On the Effects of Confirmation Bias and Test Driven Development on Software Quality

Rafael Kallis

Department of Informatics
University of Zurich
Zurich, Switzerland
rk@rafaelkallis.com

Abstract. In this paper, an overview of related work reasoning how confirmation bias and test driven development affects software quality will be presented. Specifically, I will reason how confirmation bias is present in software engineering and where it emerges from. Additionally, we will see if test driven development shows any impact on confirmation bias or productivity levels. With respect to the found conclusions, some suggestions will be made on how software quality can be increased.

Keywords: software quality, confirmation bias, negative tests, test driven development

1 Introduction

Traditionally, the significance of software quality was put into perspective in terms of billions of dollars spent trying to fix bugs. Besides consumer application failures, software quality also accounts for the 1 second glitch resulting in a 3,700 meter freefall of the Mars Lander during a live mission, the Boeing Dreamliner 787 needing a restart every 21 days due to a signed 32-bit integer conversion and the security flaw in PwC's SAP module allowing an attacker to manipulate accounting documents and financial results [1]. A lot of effort has been invested researching factors affecting software quality.

Dalal et al. performed a Root Cause Analysis on software failures. The found failures have been classified into 6 groups: *Requirement, Design, Coding, Testing, Up-gradation* and *Implementation*. The majority of failures fall in the *Testing* category, which accounts for 31% of total failures [2].

Grady categorized various bugs found in 4 departments of HP with respect to when and how the bugs have been found. Bugs found in phases after the feature has been created are more expensive to fix. The majority of bugs found in later phases, especially coding related bugs, have been

discovered using tests [3]. The findings of the two contributions above imply that software testing seems to be a significant factor determining software quality, if measured in number of defects.

Calikli et al. concluded that high levels of defect rates introduced by software developers are directly related to confirmation bias and high levels of confirmation bias among software testers are very likely to result in an increase in the number of production defects [4].

In the following pages, I will further analyse the relation between confirmation bias and software defects, as Calikli et al. inferred [4][8]. Based on the findings, I will then suggest how software quality can be increased.

2 Background

Confirmation Bias: tendency of people testing hypotheses in ways that tend to confirm rather than disconfirm them [5].

Defect Rate: measure describing the number of faults observed after release per (thousands) lines of code.

Test Coverage: percentage of code covered by tests. Metric used to find untested code fragments. Commonly used in software engineering.

Test Effectiveness: measure describing the fault detecting ability of tests. Higher test effectiveness yields more faults detected by the tests.

Positive Tests: test cases aiming to confirm, support, verify the code fragment [4][6]. Let f denote a function which accepts as an input a numeric string with length of 4 or 5 characters. A test which calls f with the strings '5346' or '24160' as an input, is considered a positive test.

Negative Tests: test cases aiming to disconfirm, fail the code fragment [4][6]. Consider f from the example above. A test which calls f with the strings '420', '567432' or '12aa' as an input, is considered a negative test.

Test Driven Development: software engineering methodology in which tests are written prior writing the code. Sometimes called test-first approach. Tests are written in form of specifications and then incrementally satisfied by code. After passing all tests, new tests are written and implemented [7].

3 Overview of Related Literature

In order to lead the discussion regarding software quality, various related literature has been reviewed. The literature originates from various studies in the fields of computer science and psychology.

Calikli et al. inferred from their findings that strong logical reasoning and hypothesis testing skills lead to less defects. Experience, role or job title, company size and engineering methodology have no effect on confirmation bias levels of a developer or tester. It is still unclear if education has an effect on an individual's confirmation bias level. They showed that education stands in relation with confirmation bias but in their following work they failed to reject the hypothesis claiming education has no effect on confirmation bias. They also observed the phenomenon of users reporting bugs above average exhibiting more tendency to verify that production defects do not exist in code they test. Therefore, they exhibit more confirmation bias [4][8].

Teasley et al. showed that functional software testing can be considered as a form of hypothesis testing. They also concluded that people tend to create more positive tests than negative tests, therefore exhibiting positive test bias, a form of confirmation bias . Similar finding have been observed by Leventhal et al. Both studies observed an increase in number and percentage of negative tests with expertise level, which strengthens the hypothesis that education has an effect on confirmation bias [5][6].

Leventhal et al. discovered that the more completely the program being tested was specified, the more likely developers write negative tests. Additionally they found that the presence of errors caused subjects to increase negative test coverage, but did not affect positive test coverage [6].

Causevic et al. observed in their test driven development based experiment that negative test cases accounted for the 29% of total test cases but contributed as much as 65% to the overall test quality score concluding that test efficiency for test-first developers was highly dependant on the negative test cases. They observed that developers using the test-first approach had a higher positive to negative test ratio, which is quite thought provoking. Moreover they noted higher quality code on a test-first group in comparison to a test-last group [9].

Bhat et al. observed a significant increase in code quality in test driven development based projects. On a project for the Windows operating system, they observed a 260% decrease in defects and a 35% increase in development time. The results were more extreme on a MSN based

project, where they observed a 420% decrease in defects with only a 15% increase in development time [10].

Erdogmus et al. on the other hand did not observe any increase in quality on test-first subjects but instead claimed that test-first subjects yielded higher productivity due to better task understanding, task focus and lower rework effort. They also found an increase of tests per unit of programming effort in test-first programmers [11].

Kaufmann et al. also evaluated test driven development and also observed an increase in productivity, since the test-first group produced 50% more code than the test-last group, the code having similar complexity to the test-last group's code. Additionally he noted a significantly higher programmer confidence in the test-first group [12].

Summarizing, confirmation bias seems to be present in software engineering. It is still unclear if the test-first approach yields better quality and if the increase in quality due to test driven development is related to confirmation bias. In the following chapter, we are going to retrospect on the findings of the observations above.

4 Discussion

Reflecting on the work from the authors above, a connection between confirmation bias levels and number of software defects might exist [4][5][6][8]. Firstly, we are going to discuss if Calikli et al.'s conclusion, that high levels of confirmation bias cause an increase in defects [4][8], is plausible. After finishing our arguments regarding the relation between confirmation bias and defects, a conversation will be held trying to determine factors affecting confirmation bias. Lastly, we will examine one specific factor in more detail, the software engineering methodology. Test driven development might influence confirmation bias levels to some degree, like some authors have observed [9].

4.1 Do higher confirmation bias levels yield more defects?

Calikli et al. found that high levels of confirmation bias are likely to result in an increase in number of defects. They discovered a correlation between number of bugs reported and production defect count on developers and testers. See fig. 1 for an overview of the data. A possible explanation to their quantitative results is that testers who report bugs above average exhibit more tendency to verify that production defects do not exist in code they test, thus exhibiting confirmation bias. The relationship of production defects with reported bugs could also be explained by the fact

that above average bug reporters might be assigned code fragments with very high defect density requiring immense testing effort. Since there are also deadlines and high time pressure to end the testing procedure, this may result in the deployment of the defected code fragment [4][8].

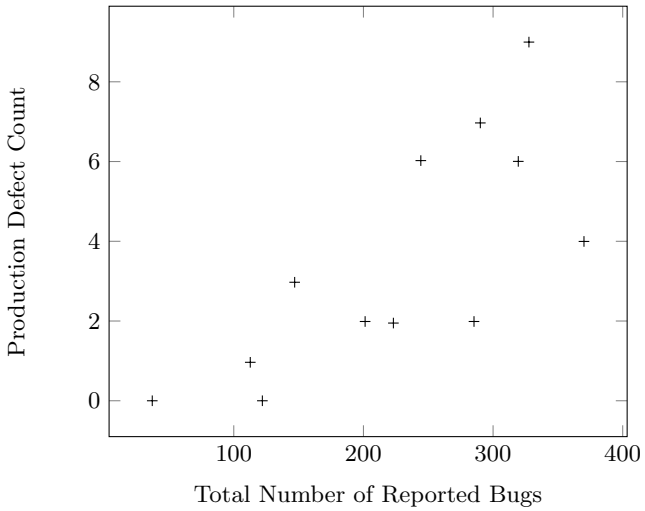

Fig. 1. Production defects and total number of reported bugs [4].

We conclude that besides confirmation bias, a lack of productivity may also explain their observations. Calikli et al. also claim that strong hypothesis testing skills lead to less defects [4][8]. Assuming strong hypothesis testing skills exhibit lower confirmation bias, their claim supports the former hypothesis.

Teasley et al. showed that hypothesis testing can be considered equivalent to functional software testing. Having a function that represents the relationship between specified input values and expected output, a functional test checks if we get the intended output from the function with respect to the specified input value [5]. But how is confirmation bias exhibited given this representation of functional testing? Teasley et al. further explain that testers may select many test cases consistent with the specifications and few inconsistent with them. This behavior is referred to as positive test bias, the tendency to create more positive tests than negative tests [5], another form of confirmation bias. This phenomenon may also be explained by the fact that requirements are sometimes not completely specified, not stating the expected behavior under failing con-

ditions. The very thought was confirmed by Leventhal et al., observing an increase in negative tests when the program being tested was more completely specified [6].

Causevic et al. have classified all test cases in their study in positive or negative tests. Negative test cases where nearly 50% less than positive, yet they detected 50% more defects [9].

Dalal et al. also point out in their root cause analysis that lack of negative testing was a cause for software defects during the testing phase [2].

Retrospecting on findings of the work above, we have seen that better hypothesis testing skills may cause an increase in negative test cases [4][8], resulting in an increase in negative test coverage. An increase in negative test coverage results into an increase in test effectiveness [9].

On the other hand, the observations of Calikli et al. may be explained by incomplete specifications, since specifications including expected failure behavior may naturally lead developers into writing more negative test cases. As Teasley et al. and Leventhal et al. found, more complete specifications yield more negative test cases [5][6], resulting in higher negative test coverage, which results in higher test effectiveness [9] as explained above.

It has to be noted that an increase in test effectiveness does not directly yield a lower defect rate, simply because high test effectiveness essentially means that more errors can be discovered. But if developers suffer under time pressure, they might not have the time to fix the bugs discovered on time, resulting in deployment of defect code fragments. Thus, Calikli et al.'s conclusion that decreasing confirmation bias yields less defects, can be true assuming the detected defects are actually fixed. Detected bugs can only be fixed if developers are productive enough.

4.2 Which factors affect confirmation bias?

We have seen that confirmation bias seems to affect test effectiveness. In this section we are going to analyse factors impacting confirmation bias levels.

Calikli et al. have gathered extensive data in order to answer the very question. In their paper, they define a set of hypotheses which claim *confirmation bias levels of software engineers are not affected by their level of education*. Their regression model failed to reject the hypotheses [8], which contradicts previous findings [4]. This is quite though provoking, since it questions the relationship between education level and hypothesis

testing skills. Almost all subjects in their study are computer science, mathematical and engineering related students, which are obliged to take theoretical courses and should in theory have strong hypothesis testing skills. Therefore the results might be skewed. Teasley et al. and Leventhal et al. found in their papers that higher education increased the number and percentage of negative tests [5][6], possibly due to lower confirmation bias.

Calikli et al. tested the hypothesis claiming *logical reasoning and strategic hypothesis testing skills are not differentiating factors in low confirmation bias levels*. In their study they compared researchers and non-researchers and found that researchers have significantly lower confirmation bias levels. Researchers consist of Computer Engineering Ph.D candidates which took theoretical computer science courses which have strengthened their reasoning skills. Therefore they successfully rejected the hypothesis [8].

They also failed to reject hypotheses claiming *confirmation bias levels of software engineers or testers are not affected by their industrial development or testing experience* [8] which confirms previous findings [4]. Teasley et al. as well as Leventhal et al. found different results. They showed that positive test bias may be mitigated by higher levels of programmer expertise since they observed a higher negative to positive test ratio on developers with more expertise which results from less positive test bias [5][6].

Calikli et al. also failed to reject the hypothesis which claims *confirmation bias is not related to position/job title*. In their regression they included a total of 152 developers, testers and analysts [8]. The subjects having a similar education background thus having similar hypothesis testing skills might explain this result. A significantly different picture is being depicted when researchers also are included in the regression model. There is a significant difference in confirmation bias between researchers and all other subjects. Researchers consist of Computer Engineering Ph.D candidates [8]. It is highly probable that theoretical computer science courses have strengthened their reasoning skills. This observation supports the claim of education level affecting confirmation bias.

Further regression models also failed to reject the hypothesis claiming *confirmation bias is not affected by software engineering methodology used* [8]. The subjects in the regression used by Calikli et al. to test software engineering methodologies affecting confirmation bias are not uniformly distributed thus the result might be skewed.

Concluding, the only hypothesis that got successfully rejected claimed *logical reasoning and strategic hypothesis testing skills are not differentiating factors in low confirmation bias levels*. It is still unclear if the software engineering methodology affects confirmation bias. In the following chapter we are going to discuss if test driven development, a commonly used methodology, has any effect on confirmation bias or software quality in general.

4.3 Does test driven development affect confirmation bias?

Reviewing our arguments above, software engineering methodologies might affect confirmation bias levels. In this section, we will argue if test driven development impacts confirmation bias levels.

The positive effects of test driven development have been observed in many instances. In some instances the test-first methodology caused a significant decrease in software defects [10]. Can the decrease in defects be explained with confirmation bias? As explained above, low confirmation bias results in less defects and maybe the **(1) test-first methodology decreases confirmation bias thus resulting in less defects**. Another plausible theory is that **(2) test-first approach yields higher productivity thus allowing developers to fix more defects**. There is also a possibility that **(3) test driven development increases confirmation bias**.

Bhat et al. also had another significant observation besides the increase in quality. They found that the test-first approach increased development time by 35% and 15% respectively [10]. It is still unclear if there has been an increase in productivity. Assuming there was no increase in productivity, the increase in quality due to test driven development might be explained by a decrease in confirmation bias. But we cannot draw definite conclusions from assumptions based on Bhat et al.'s results.

The findings of Causevic et al. disprove the first theorem and support the third theorem, since they observed a lower negative to positive test ratio while the test-first approach was used. They claim that the lower ratio is a result of the inherent positive test bias of test-driven development approach. In the test-first approach, test cases are driving a developer towards implementing required functionality in a constructive rather than destructive way [9].

Kaufman et al.'s study supports the second theorem since they observed a 50% increase in productivity. Specifically they saw that developers using the test-first approach wrote more tests per programming effort. It is worth mentioning that the test-first code showed similar cyclomatic

complexity in comparison to test-last code. The increase in productivity might be explained by the increase of programmer confidence which affirmed advantages of test driven development [12].

Erdogmus et al. support the second theorem, finding no evidence of increase in quality, but increase in productivity. A possible explanation for the non proportional increase of quality to the increase of testing, as stated by Erdogmus et al., is programmer's skill level [11]. The observed quality had a high variance and was strongly dependant on the programmers skills. His theory is possibly true if we consider the strong relationship between hypothesis testing skills and software defects [4][8], which possibly explains the high variance. Erdogmus et al. give very plausible synergistic explanations for the increase in productivity. Firstly, test driven development results in *better task understanding* since writing a test before implementing the functionality requires a programmer to express functionality unambiguously [11]. *Better task focus*, due to the limited scope of a single test case. Larger pieces of functionality are decomposed down to smaller chunks, resulting in lower cognitive load [11]. Another effect of limited scope of a single test case, is *lower rework effort*. When a test fails, the root cause is easier pinpointed [11]. Erdogmus et al.'s findings strongly support the second theorem (*test-first approach yields higher productivity*).

Calikli et al. also failed to reject the hypothesis claiming *confirmation bias is not affected by software engineering methodology used* which disproves the first and third theorem [8].

Summarizing, the claim stating *test-first methodology decreases confirmation bias* has no strong support and most observations disprove it. On the other hand, the hypotheses stating *test-first approach yields higher productivity and exhibits more confirmation bias* have strong support from all referenced studies above, which implies that it is likely to be true.

5 Conclusions

5.1 Summarization of Results

First we reviewed some related work regarding the relationship of confirmation bias and software defects. The results shown, imply that **better hypothesis testing and logical reasoning skills yield higher test effectiveness** [5][6]. Higher test effectiveness does not imply less defects. Calikli et al. claimed better hypothesis testing skills yield less defects [4][8], which is likely assuming developers are productive enough. If bugs

are detected but the developer does not have the time to fix them, the higher test effectiveness will not improve software quality.

Continuing the discussion regarding confirmation bias, we have seen some insights regarding factors affecting it:

- *Level of education* seems to be a plausible factor differentiating confirmation bias levels [4][5][6], but some research found non supporting evidence [8].

- There also seems to be a weak relationship between *experience level* and confirmation bias [5][6], although Calikli et al. did not find supportive evidence in their work [4][8].

- It is unlikely that *position/job title* has any effect on confirmation bias, as related work has shown [4][8].

- It is possible that *software engineering methodology* has an impact on confirmation bias [4][9][11].

Retrospecting on our discussion regarding test driven development, we can conclude that **the test-first approach yields higher productivity** due to better task understanding, better task focus and lower rework effort because of the limited scope of each text case [11][12]. Software developed using the test-first approach had a lower negative to positive test ratio in comparison to the test-last approach, possibly due to the inherent positive test bias of test driven development[9], which might imply that **test driven development increases confirmation bias**. The positive test bias of the test-first approach might be explained by the fact that test cases are driving a developer towards implementing required functionality in a constructive rather than destructive way [9].

5.2 Improving Software Quality

We can give some general guidelines in order to improve software quality based on the findings we found above.

Test Driven Development: an increase in productivity can always benefit software quality, since more defects can be fixed in the same amount of time. Various studies have observed an increase in productivity while using test driven development, due to better task understanding, limited test case scope and lower rework effort [11][12]. Therefore it is recommended to use the test-first approach.

Specifications: it has to be noted that when using the test-first approach, the number of negative tests is highly dependant on the completeness of specifications. The completeness of specifications will determine the number of negative test cases [5][6]. The number of negative test cases correlate with test effectiveness [9]. Therefore it is recommended to include relatively high number of specifications describing expected failure behavior of the software.

Confirmation bias: better hypothesis testing skills and logical reasoning impact confirmation bias levels. Lower confirmation bias levels increase test effectiveness. Therefore it is recommended that developer training is focused on personal skills, such as logical reasoning, instead of tasks. Organizations should find ways to improve basic logical reasoning and strategic hypothesis testing skills of their software engineers [4][8]. It is also important to make software engineers aware of the existence of confirmation bias and the importance of negative tests [5][6]. Organizations could also introduce a standard, in order to enforce balancing every positive test case with a negative one [6].

It has to be noted that not all of the stated suggestions may increase software quality in every organization. These methods worked on them, which doesn't imply that they will work on you.

References

1. Neumann, P. G., 2017, January. Risks to the Public. In *ACM SIGSOFT Software Engineering Notes*, Volume 42 Issue 1
2. Dalal, G. and Chhillar, R. S., 2013. Empirical Study of Root Cause Analysis of Software Failure, In *ACM SIGSOFT Software Engineering Notes*, Volume 38(4)
3. Grady, R. B., 1993, November. Practical Results from Measuring Software Quality, In *Communications of the ACM*, Volume 36 Issue 11
4. Calikli, G. and Bener, A., 2010, September. 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*. Article No. 10
5. Teasley, B. E., Leventhal, L. M., Mynatt, C. R., Rohlman, D. S., 1994, February. Why Software Testing Is Sometimes Ineffective: Two Applied Studies of Positive Test Strategy In *Journal of Applied Psychology*. Vol 79(1), 142-155.
6. Leventhal, L. M. , Teasley, B. E., Mynatt, C. R., Rohlman, D. S., 1994, November. Analyses of Factors Related to Positive Test Bias in Software Testing. In *International Journal of Human-Computer Studies*. Volume 41, Issue 5, Pages 717-749
7. Beck, K., 2003. Test Driven Development- by Example. In *Boston: Addison-Wesley*.
8. Calikli, G. and Bener A., 2015. Empirical Analysis of Factors Affecting Confirmation Bias Leves of Software Engineers. In *Software Quality Journal*. Volume 23(4), Pages 695-722

9. Causevic, A., Punnekkat, S. and Sundmark, D., 2012, September. Quality of testing in test driven development. In *Quality of Information and Communications Technology (QUATIC), 2012 Eighth International Conference* (pp. 266-271). IEEE.
10. Bhat, T. and Nagappan, N., 2006, September. Evaluating the efficacy of test-driven development: industrial case studies. In *Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering* (pp. 356-363). ACM.
11. Erdogmus, H., Morisio, M. and Torchiano, M., 2005. On the effectiveness of the test-first approach to programming. In *IEEE Transactions on software Engineering* 31(3), pp.226-237.
12. Kaufmann, R. and Janzen, D., 2003, October. Implications of test-driven development: a pilot study. In *Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications* (pp. 298-299). ACM.

Word of honor

I, Rafael Kallis, hereby declare that I have produced this work independently and have used no other than the listed tools and sources.