IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
Impact Factor: 7.301
Volume 3, Issue 2, September 2023

A Critical Analysis of Software Defects Ability to Understand using of Software Testing

Samundra Singh[1] and Dr. Shivani[2]
Research Scholar, Department of CS[1]
Assistant Professor, Department of CS[2]
Bhagwant University, Ajmer, India

Abstract: *Remaining defects are the cause of significant issues in the software development industry and not fixing them sooner increases the risk of negative effects such as system crashes, customer dissatisfaction and higher costs. Despite these observations, companies do not always remove newly found defects due to varying factors. These factors, which lead to longer term defects, are not well studied and therefore require more research.*

Keywords: Cause, Defect, Software, Testing, and Effect etc

I. INTRODUCTION

In today's world people's jobs, comfort, safety, entertainment, decisions and their lives depend on computer software, so it is better to get it done right [1]. This is one reason why software testing is extremely important. Software testing is as essential as any other phase of the software development life cycle. Testing should be done before deploying the software for use, it helps to discover errors in time and ensure the working of the software as required, it also helps to correct the defects discovered after the product is put into use. reduces. Software testing is an important part of software quality assurance (SQA), an activity used to evaluate and improve software quality [2]. Software testing includes a set of activities performed with the sole purpose of finding errors in software. It verifies and verifies whether the software or product is working correctly without any errors or defects, enabled bugs. In the testing phase, errors from previous phases should be detected, this ensures software reliability and quality assurance [3]. Software testing tools facilitate generated information management, communication, test execution, generation and production [6]. Therefore, the use of appropriate software testing tools effectively and efficiently enhances the testing process. Currently, a large amount of software testing tools are available to assist at any stage of the testing process [7]. Even though many software testing tools are primarily useful in managing and keeping track of scheduled or performed software tests; However, some software testing tools provide automation for core testing tasks [4] ; This reduces costly, time-consuming and error-prone manual testing [8]. Test automation facilitates the identification of errors and defects in specially developed software effectively and efficiently; Because it increases the reliability factor, saves time and increases productivity in human efforts and also reduces costs in the long run [4].

II. LITERATURE REVIEW

Software defect prediction is a hot topic in the field of software engineering. Many fault prediction algorithms have been proposed [9], which are mainly based on machine learning, such as linear regression model (LRM) [10], decision tree [11] and ensemble learning [12], etc. The goal of software defect prediction mainly includes two categories: classification and ranking. The classification task focuses on the correct classification of software parts. Many algorithms [13, 14] have been proposed, but no conclusion has been drawn on the most optimal algorithm. This paper focuses on the ranking task. Different from classification, it focuses on testing priority of software parts. By ranking the prediction results, high-risk software parts can be identified. Elberg and Ohlson proposed an Elberg diagram to estimate the accuracy of the predictive model. They proved that most software defects were contained in 20% of the modules. Zimmerman [15] demonstrated that high-complexity modules carry high risks. To improve the efficiency of ranking prediction, several different approaches have been proposed. Ostrand et al. [16] developed a negative binomial regression as a predictor for predicting defects in the next release based on the code in the file in the current release.

ISSN 2581-9429 IJARSCT

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
Impact Factor: 7.301
Volume 3, Issue 2, September 2023

Yang [17, 18] introduced a learning-to-rank (LTR) approach with overall difference evolution to obtain the coefficients of a linear model. Most of the research in defect prediction mainly focuses on algorithmic improvement and performance comparison, while little attention is paid to the optimization of forecasting strategy. The novelty of this paper is to propose an optimization strategy that combines defect prediction and STP to improve the prediction accuracy. Cross-project defect prediction (CPDP) is another research point related to defect prediction in this paper. CPDP is often used when automatic prediction is impossible because of not enough training data or poor data quality [16]. Some experiments [20] have shown that CPDP can provide acceptable prediction results if the training data is carefully chosen. In addition, Herbold [21] proposed a distance-based strategy for selecting training data. The results demonstrated that their strategy could significantly improve the effectiveness of CPDP, but it was still not competitive with defect prediction within the project. Transfer learning [22] was introduced to build a more effective CPDP model based on weighted data transferred from cross-project data.

OBJECTIVES

To identify the factors that companies consider when deciding whether to allow and retain defects.

METHODS

A two-step method is applied where a snowballing based literature review is used to identify research gaps in order to gain an understanding of the research area. These results are then complemented by an empirical and industrial case study with 16 interviews. Interviews were analyzed with an approach inspired by thematic coding, which led to the main findings of the study.

III. RESULT ANALYSIS

What factors are considered in industrial practice when deciding not to remove a defect?

The results of the interviews identified 11 different factors that lead to slack defects, in short the factors. The results also indicate that known and unknown defects are treated differently, as factors are considered in different ways. To find out what factors participants use to perpetuate defections, interviewers were first asked how they define defections, listed in question 4.5, also to check whether the results match Let's look at how the literature defines defects [13]. The results indicate a heterogeneous view of what is at fault within the sample. While some interviewees define it as when the system stops working, others said it is when the system does not respond as expected. These statements are also related to factors such as the priority of the defects as well as their severity, e.g. minor or major. However, some of the interviewees also stated that they do not have any known flaws on the system yet. This information was considered during the analysis to help ensure that the interview results were interpreted correctly. Following, each of the identified factors is described. Cost: The cost of removing a defect is affected if the defect is fixed, where higher cost leads to lower priority and thus less likely to be fixed. As Interviewee P5 said, "Um, it could be something that's a trifle and fixing it will cost more development time than maintaining it". However, this statement is only true when considering the perceived value of addressing the defect. As such, high value, but costly, defects are more likely to be addressed than low value defects. This factor is generally considered important, as most of the interviewers (N=10) mention it during the interview. However, a subset (N=6) either did not mention it, or did not find it significant. Time: Time, as a complementary metric to cost, is also commonly considered by the interviewers (n = 7) and in relation to the time required to fix the defect. Time is considered to influence the priority of the defect in the same way as cost, ie cost is also considered while prioritizing the defects. As stated by interviewee P7 “Some defects take a long time to fix” Severity: Severity of a defect is one of the main factors mentioned by some of the interviewees (N=4). Interviewees say that even though they knew that less severe defects could cause problems later in development, many interviewees chose not to take action. This is mainly the case for defects of low perceived severity, while defects of high perceived severity are generally addressed. As stated by interviewee P10 when explicitly asked about the factors that are considered; "How serious is the defect" impact: Impact, defined as the impact on the system or its development, is one of the more commonly reported factors by most interviewees (N=5). High impact defects are more likely to be addressed than low impact defects. Difficult to fix "[when] the impact [of the defect] is low" as stated by interviewee P14: The perceived difficulty of fixing a defect was also expressed by interviewees (N=3). As stated by interviewee

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
Impact Factor: 7.301
Volume 3, Issue 2, September 2023

P15 "we don't know how to solve this [defect]". This factor is related to time/cost and so is related to the value provided by addressing the defect. Therefore, if a defect is considered difficult to fix with little value gain, it is discarded and may become languishing. Outdated functionality: Defects associated with out-of-date, or legacy, functionality are considered less important to address by some interviewees (n = 1). Therefore, defects found in new functionality, or functionality under development, are more likely to be addressed than defects in older, established functionality. Security: This factor was specifically brought up by interviewees (n = 1) working in companies with systems that deal with sensitive data. Interviewees stated that defects with a high perceived impact on security are given higher priority than defects with a low impact. As stated by interviewer P4 "exposing user credentials, or, um, being able to manipulate data you shouldn't be allowed to do these kinds of things". Affecting the business: Defects affecting the business rather than the system are given a lower priority by some interviewees (n = 2). It should be noted that this factor was presented by the developers, meaning their perspective is more on development, whereas a manager may provide a different answer for priority. As stated by interviewee P4 "But, uh, the impact on the business is always, kind of the first [thing we consider]". This result also indicates that different roles may value the same factor in different ways. Impact traffic: This factor was only mentioned by one interviewee (N=1), who was in a company in a domain where system traffic is important. For this factor, the impact on traffic is related to its assigned priority, which means that higher impact implies higher priority. This factor also highlights that there is variation in what factors are considered in different domains. As stated by interviewee P13 "So for us, the first question is always where do these faults affect traffic". Risk: Risk, mentioned by one interviewee (n = 1), relates to the impact of defects on the project rather than the product itself. The amount of risk is correlated with the priority of the defect, such that higher risk implies higher priority. As stated by interviewee P13 "And this is because when you have a defect, you must consider how much of a risk it is, or if it is a minor or a major defect".

Effect of chronic defects

To what extent do long-lived defects affect a software product? As presented, there are a number of factors that go into the decision if a defect should be prolonged or not. Additionally, as such, it was determined that there were multiple reasons for the defects to persist. However, long-standing defects are always likely to have an impact on software development or its organization. According to the interview results, the impact of defects depends on the severity of the defect, i.e. if it is a minor or major defect [2]. A synthesis of the interviewers' perceptions on severity leads us to the following definitions: • Major defect: Any defect that affects the functionality of a system to such an extent that it leads to a system crash is considered a major defect. Is. • Minor Defect: Any defect which has only minor or no impact on the functionality of the system is considered as minor defect. For example, performance defects are considered minor defects by most participants (n = 10). Note that the severity in these definitions focuses on the functional characteristics of the system. This result could be due to several factors, e.g. Companies policies/preferences, interviewer's knowledge of non-functional attributes, etc. The purpose of RQ2 was to identify the impact of defects on the system. This was of interest as related research highlighting defects have negative effects on software systems and therefore assumed that supporting evidence would be observed [13]. However, the results indicate the opposite as the majority (n = 13) of the interviewees stated that minor lingering defects have no effect on the system. Interviewee P4 said that "they are living for us. It may as well be that, because they are minors, they are not causing any problems". In fact, this lack of effectiveness was cited as one of the reasons why the defects are allowed to persist. According to some of the more experienced interviewees (n = 5), some of the defectors said that the defects could remain in the system for more than 10 years and would probably never be fixed. However, this observation was not entirely conclusive, as statements from some interviewees indicated that lingering faults could change and become more dominant over time. Interviewee P9 said that "a defect can last for 20 years if it is not causing problems". For example, it was observed that defects classified as minor in the past turned into major defects for future releases of the software. As stated by interviewee P2 "It may happen that minor defects become problems later". This change resulted in a major and immediate effort to rectify the defects. As a result, estimating the impact and impact of defects over time is difficult and can lead to unforeseen circumstances and additional costs. However, it should be noted that the perceived future impact of a defect was not obtained as a factor. One reason for this observation may be the perceived complexity of such estimates, which necessitates future research into the effects of defects over time.

ISSN 2581-9429 IJARSCT

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
Impact Factor: 7.301
Volume 3, Issue 2, September 2023

IV. CONCLUSION

The study provides industrial practitioners with insight into factors to consider when deciding whether a defect will persist. Contrary to the body of knowledge, the study also provides evidence, that lingering defects may not be a major problem in practice. However, due to the size of the study and its interview sample, future research is needed.

BIBLIOGRAPHY

[1]. Pressman, R. S. (n.d.). Software engineePressman, R. S. (n.d.). Software engineering (2nd ed.). New York: McGraw-Hill Book Company.ring.

[2]. Chauhan, R. K. & Singh, I. (2014). Latest Research and Development on Software Testing Techniques and Tools, 4(4), 2368–2372.

[3]. Wala, T. & Sharma, A. K. (2014). Improvised Software Testing Tool, 3(9), 573–581.

[4]. Merina, C. (2019). Tool Usability Parameter in Determining the Performance of Software Testing Tool, IJTB (International J. Technol. Business), 3(1), 8–18.

[5]. Tassey, G. (2002). "The Economic Impacts of Inadequate infrastructure for software testing". Natl. Inst. Stand. Technol.

[6]. Raulamo-Jurvanen, P., Mäntylä, M. & Garousi, V. (2017). Choosing the Right Test Automation Tool, in 21st International Conference on Evaluation and Assessment in Software Engineering (EASE 2017), 21–30.

[7]. Jain, V. & Rajnish, K. (2018). Comparative Study of Software Automation Testing Tools: OpenScript and Selenium, Int. J. Eng. Res. Appl., 8(2), 29–33.

[8]. E. Arisholm, L. C. Briand, E. B. Johannessen, A systematic and comprehensive investigation of methods to build and evaluate fault prediction models, Journal of Systems and Software 83 (1) (2010) 2–17.

[9]. R. Malhotra, A systematic review of machine learning techniques for software fault prediction, Applied Soft Computing 27 (C) (2015) 504–518.

[10]. G. Denaro, L. Lavazza, M. Pez, An empirical evaluation of object oriented metrics in industrial setting, Journal of Object Technology Vol.

[11]. M. A. D. Almeida, S. Matwin, Machine learning method for software quality model building, in: International Symposium on Foundations of Intelligent Systems, 1999, pp. 565–573.

[12]. Z. Sun, Q. Song, X. Zhu, Using coding-based ensemble learning to improve software defect prediction, IEEE Transactions on Systems Man and Cybernetics Part C 42 (6) (2012) 1806–1817.

[13]. T. Menzies, J. Distefano, A. Orrego, M. Chapman, Assessing predictors of software defects, Proceedings Workshop on Predictive Software Models.

[14]. S. Lessmann, B. Baesens, C. Mues, S. Pietsch, Benchmarking classification models for software defect prediction: A proposed framework and novel findings, IEEE Transactions on Software Engineering 34 (4) (2008) 485–496.

[15]. T. Zimmermann, R. Premraj, A. Zeller, Predicting defects for eclipse, in: International Workshop on Predictor MODELS in Software Engineering, 2007. Promise'07: ICSE Workshops, 2007, pp. 9–9.

[16]. T. J. Ostrand, E. J. Weyuker, R. M. Bell, Predicting the location and number of faults in large software systems, IEEE Transactions on Software Engineering 31 (4) (2005) 340–355.

[17]. X. Yang, K. Tang, X. Yao, A learning-to-rank approach to software defect prediction, IEEE Transactions on Reliability 64 (1) (2015) 234–246.

[18]. B. A. Kitchenham, E. Mendes, G. H. Travassos, Cross versus withincompany cost estimation studies: A systematic review, IEEE Transactions on Software Engineering 33 (5) (2007) 316–329.

[19]. Z. He, F. Shu, Y. Yang, M. Li, Q. Wang, An investigation on the feasibility of cross-project defect prediction, Automated Software Engineering 19 (2) (2012) 167–199.

[20]. S. Herbold, Training data selection for cross-project defect prediction, in: International Conference on Predictive MODELS in Software Engineering, 2013, pp. 1–10.

[21]. Y. Ma, G. Luo, X. Zeng, A. Chen, Transfer learning for cross-company software defect prediction, Information and Software Technology 54 (3) (2012) 248–256.

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
Impact Factor: 7.301
Volume 3, Issue 2, September 2023

[22]. B. Turhan, T. Menzies, A. B. Bener, J. D. Stefano, On the relative value of cross-company and within-company data for defect prediction, Empirical Software Engineering 14 (5) (2009) 540–578.

[23]. T. Zimmermann, N. Nagappan, H. Gall, E. Giger, B. Murphy, Crossproject defect prediction:a large scale experiment on data vs. domain vs. process, in: Joint Meeting of the European Software Engineering Conference and the ACM Sigsoft International Symposium on Foundations of Software Engineering, 2009, pp. 91–100.

[24]. E. Kocaguneli, G. Gay, T. Menzies, Y. Yang, J. W. Keung, When to use data from other projects for effort estimation, in: International Conference on Automated Software Engineering, 2010, pp. 321–324