Why Do Software Developers Practice Test-Driven Development?

Patrick Kayongo, Wallace Chigona, ZwelithiniMabhena

Department of Information Systems, University of Cape Town, South Africa
pat.kayongo@gmail.com,wallace.chigona@uct.ac.za,zwe5184@gmail.com

***Abstract*—Test driven development (TDD) is a crucial phase for any software developer. Most software developers using TDD observed an increase in software quality and reduced defects in the software. For that reason, it is our belief that it is critical to understand factors influencing software developers' intention to perform TDD. The Theory of Planned Behaviour (TPB) proposes that intention to perform behaviour is influenced by three factors: attitude towards the behaviour, subjective norm, and perceived behavioural control. The article reveals how the model is tested based on data collected from an online survey distributed around the world. This psychographic study showed that data collected does not conform to the model, and recommendations are made for a future study to form a more comprehensive model.**

***Keywords*— Software Developers, Test-driven development, Theory of Planned Behaviour, V-Model, eXtreme Programming**

I. INTRODUCTION

With the growing prevalence of software, rising use and dynamism has become critical for software development projects. There is a growing need to produce quality software which meets with the constantly changing user requirements whilst preventing defects from being introduced. With the pressure to deliver on quality, software development teams face the challenge of increased volatility of the requirements from customers and organisations while a project is still in progress. The changes in requirements are a result of changing customer needs over the lifecycle of a project, and increased understanding of the domain by the development team [1]. So, what are the factors and determinants to performing TDD?

The biggest determinant to performing TDD is not necessarily the level of skill of the developer, but subjective beliefs about the outcomes of performing the practice. Studies have also found that increased volatility of requirements lead to an increased amount of defects introduced into the software, especially when requirements are changed closer to the release date [2]. Firstly, this paper reveals that TDD can benefit the teams and eliminate the challenges associated with software testing. Secondly TDD has shown to decrease the number of defects appearing, increase the maintainability of the codebase, and in some cases increase developer efficiency, all which contribute to better software produced [3]–[5].

The paper discusses the psychographic motivations behind the practice of TDD. Motivated by this there is a need to therefore understand the factors within the Theory of Planned Behaviour (TPB) theoretical model as well as the intention of software developers in performing TDD. The study is a psychographic study; implying that it is the developer being studied who holds the knowledge about themselves.

II. LITERATURE REVIEW

TDD has guided the strategy of code functionality. We discuss TDD in light of other development models and benefits and challenges that have been realised.

A. TDD

Test driven development is a practice where tests are written before the functionality. It is done to guide the design of the code for the functionality, as well as to ensure test coverage for all functionality. Unit tests are a form of testing, where tests are written for small units of software being developed. Unit testing involves writing small tests to confirm that a piece of written programming language is working as intended. This involves the setting up of test data, the running of the piece of the software that manipulates the data, and the confirmation that the result of the manipulation is as intended [6].

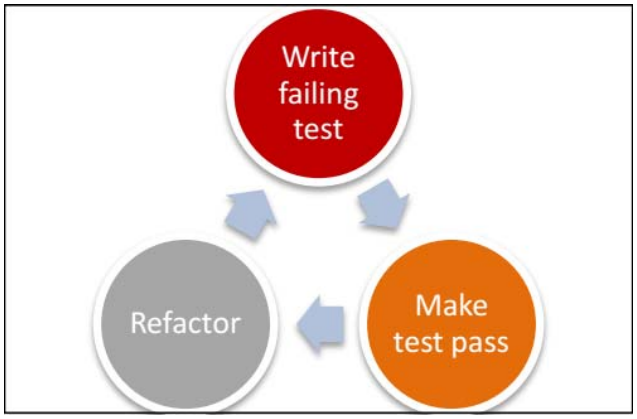

Fig. 1 TDD Cycle

Fig. 1 shows the TDD developments in practice where unit test cases are written first. If the code implementing the functionality does not exist yet, the unit test is expected to fail. The implementation code is then written by the developer, which causes the test to pass. After which, the developer can refactor the implemented code, simply meaning; making alterations to the structure of the code to

978-1-5090-2576-3/16/$31.00 ©2016 IEEE

increase efficiency or readability, and use the test to ensure that the production code functions as is intended [3]–[5].

B. AGILE SOFTWARE

Agile software development methods have been used as alternatives to TDD as a way of mitigating some of the challenges, such as constantly changing requirements, resulting in constantly changing code, increasing the potential for the introduction of defects. Agile software development is an approach to software development projects that has aimed to overcome some of the challenges posed by traditional process-driven software development approaches such as the waterfall method of software development [7].

The traditional, process-driven approaches of software development assumed a level of consistency in the operating environment over time. This assumption resulted in long periods of planning large releases of software to be developed, followed by the analysis of user requirements and the implementation of these user requirements. Yet, if there were changes in the environment that the software was being developed in, or changes in requirements, such a fixed approach found difficulty adjusting to these changes, or being 'agile'[8]. Software development firms sought to remove a lot of the scaffolding that came with the traditional approaches, to make it easier to quickly respond to changes in requirements within the environment they are developing for. [9]. This then gave way to the V-model.

C. V-MODEL

As the waterfall model for software development gained maturity, a V-model was developed which linked different stages of the software development lifecycle to different testing activities, in terms of level of detail[10], [11].

With the rise of object-oriented programming, it was assumed that many of the advantages that it brought such as modularity would lessen the need for software testing, but it was found that the very same principles that were applied beforehand were still required to ensure the quality of the developed software [10].

D. BENEFITS

Numerous generalizable studies have been done to assess the benefits of TDD. Most of the studies which have been done are case studies within organizations, or experiments in controlled environments that measure the differences in various measures when TDD has been used to develop software compared to when it has not been used.

In various case studies, a common finding is that there is an increase in software quality shown by a reduction of the defects found. When compared to cases where TDD is not used, it has been found that the number of defects in the software was less than what was initially expected, or less than the defects found in a similar control project where other factors were kept similar [3]–[5].

A second benefit that has been found in case studies is how easy it is to make changes to the code at a later stage, and the prevention of adding defects on existing functionality (regression defects) when new functionality is developed on an existing codebase. As has been mentioned above, test-driven development forms part of many methods found in agile software methodology, which seeks flexibility and the ability to change with a changing environment and changing requirements. Tests may also increase the ease of changing as existing functionality is tested, and therefore changes in the code can be made to adapt to the changing environment with less risk of introducing defects into existing functionality.

Thirdly, the maintainability and changeability of the code has been recognised as a benefit. At a study done at IBM, developers found it easier to make changes to code, and reduced the risk of regression defects when using TDD compared to when TDD was not used [5].

Several other benefits have been found when using TDD, such as improved quality of the software due to reduced defects, improved maintainability of the codebase, and less regression defects introduce0d with the development of new software. Lastly a common benefit that has been found is an increase in developer efficiency with shorter pieces of focus, allowing the developer to focus on one piece of functionality at a time. It is easier to find and rectify the relevant fault because of smaller scope being done at a time [4], [5].

E. CHALLENGES

The biggest challenge of TDD that is evident in literature is the additional time taken to write the test cases. When compared to a case where no tests are written or only a few test cases were written, it has been found that there is a statistically significant increase in time writing these tests, as they are written for every piece of functionality, and for various scenarios of the feature [3], [4].

Some studies have also found that writing unit tests first do not lead to an increase in the readability and maintainability of the code base [12]. A cause of this could be that in an effort to write the code in such a way that tests can be written, a developer may over-complicate the code and introduce unnecessary complexity, as was mentioned in the introduction.

III. THEORETICAL FRAMEWORK

The theoretical model of the TPB has been used to understand the factors affecting intention, namely attitude towards the behaviour, subjective norm and perceived behavioural control as shown in Fig 2. The TPB is an enhanced version of the Theory of Reasoned Action (TRA) [13]. This popular theory used in psychological research proposed that the intention to perform behaviour was affected by the beliefs about the outcomes of performing that behaviour.

These beliefs are separated into two distinct kinds: behavioural, which form the basis of an individual's attitude towards the belief, and normative, which are the

individual's beliefs about what is considered 'normal' in their environment, or their subjective norm [14]

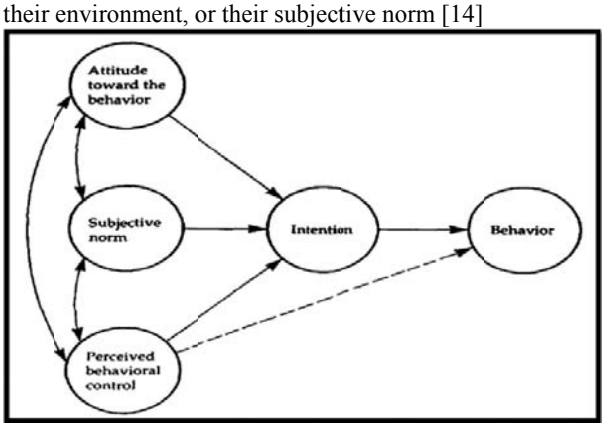

Fig. 2 Theory of planned behaviour (Ajzen, 1991).

F. Model Validation

To test whether the theoretical conceptual model that was constructed through the literature supported the data from the responses that we were collected, structured equation modelling (SEM) was used. SEM has been growing in social science, as it provides a tool for theoretical model testing, validations and modifications using sample data. Relationships between indicators and factors are determined using the covariance between the various indicators. In its simplistic form, if there is high covariance between indicators, it is then assumed that there is a relationship between these indicators, and they could represent a latent variable (or factor). The numerical values of these relationships are known as factor loadings, which can be seen as regression coefficients [15]

There are two types of factor analysis: confirmatory factor analysis and exploratory factor analysis. Confirmatory factor analysis seeks to test whether data supports a pre-specified theoretical model. This is done by comparing the theoretical covariance matrix (which represents the relationships between the constructs of the model), and the covariance matrix of the actual data received. For the purposes of this study, confirmatory factor analysis was done it was conducted using R for statistical computation.

G. Attitude and Perception Hypothesis

H_0: There is no relationship between a software developer's attitude towards TDD and the individual's perception of the difference in the quality of the software developed.
H_1: There is a relationship between a software developer's attitude towards TDD and the individual's perception of the difference in the quality of the software developed.

H. Attitude and Perception of Difference in Time Hypothesis

H_0: There is no relationship between a software developer's attitude towards TDD and the individual's perception of the difference in time taken to develop software.
H_1: There is a relationship between a software developer's attitude towards TDD and the individual's perception of the difference in time taken to develop software.

I. Attitude and Perception of Efficiency

H_0: There is no relationship between a software developer's attitude towards TDD and the individual's perception of the difference in individual efficiency.
H_1: There is a relationship between a software developer's attitude towards TDD and the individual's perception of the difference in individual efficiency.

IV. Research Methodology

A realist ontological perspective and positivist epistemological perspective has been taken, as this aligns with both the assumption of an objective. Data collection was conducted through an online survey which was available at the URL http://research.patkayongo.co.za.The survey was distributed through the researcher's LinkedIn connections that are software developers, various LinkedIn groups and posted several times on the social media website Twitter generating a total of 799 responses. The focus was mainly on software developers.

J. TPB as a Conceptual Model

At the same time, the TPB used as a conceptual model in this research assumes a causal nature between the phenomena. It posits that attitudes, subjective norms and perceived behavioural control determine an agent's intention to perform a behaviour, which in turn influences whether or not the agent performs the behaviour. Therefore, an objective reality is assumed in which there is an objective causal nature between the phenomena.

K. Type of Study and Sample

Based on the goal of the research to test hypotheses and deductively explain a developer's intention to practice TDD, as well as the ontological and epistemological assumptions discussed, a quantitative methodological approach is to be used to conduct the research. These will provide standardised and generalizable responses that can be used to test the hypotheses and deduce conclusions.
The population that we were focusing on are software developers who regularly create software.

L. Research Instrument

A survey was used as the research instrument for data collection. A survey is a common instrument for data collection where TPB is a conceptual model behind the questions, and has been used in various fields to understand psychological motivations to perform behaviour. All ordinal data refers to responses on the Likert scale. These responses were recorded on a scale from 1 to 7 where 1 represented 'Strongly Disagree' and 7 represented 'Strongly Agree'. Measures with negative responses as higher numbers (AttitudeTime,

SubjectiveNormTime, UnitTestDifficulty and TDDDifficulty) were reversed to ensure that higher numbers reflect positive responses while lower numbers reflect negative responses. Two questions were asked to determine if users had the intention to practice TDD: whether they are practicing TDD, and if not, whether they would like to practice TDD. If they answered positively to any of these questions, the respondent is assumed to have the intention to practice TDD.

M. ETHICAL CONSIDERATIONS

Individuals were being surveyed within this research; ethical considerations have to be made. Respondents were made aware of the nature and the purpose of the research before they proceeded in answering the survey questions.

The instrument was distributed online to willing parties; respondents were not coerced or forced into completing the questionnaire. Perceptions were then recorded; and we had to keep these confidential. The respondents were informed that their responses would be kept confidential, and had the option of remaining anonymous. Also, in all additional comments given by the software developers, we ensured that there were no way comments mentioned in the research could be linked back to an individual software developer.

N. LIMITATIONS

While there was an intention to have a representative sample of software developers around the world, the nature of the media through which the survey was distributed and the random nature of the responses that were received making it difficult to assume that the sample represents the entire intended population under study. This was mitigated by using t measure of whether individuals had the intention to practice TDD before responding.

V. DISCUSSION OF FINDINGS

The main measure was on whether individuals had the intention to practice TDD. Out of all respondents, 94.6% had the intention to perform TDD. While the number of those who have an intention to perform TDD is not representative of the entire population of software developers worldwide, the sample could give us a good understanding of the motivations and

psychometric driving factors of those software developers

who do practice TDD.

O. HYPOTHESIS TESTING

The conceptual model proposed included four different influences of a software developer's attitude towards practicing test-driven development: the developer's attitude towards the differences in quality of software developed, time taken, efficiency and maintainability of the underlying code.

P. ATTITUDE

When attempting to conduct an ordered logistic regression on the Attitude variable and all its determinants, the procedure could not be conducted in R because of the data provided, which indicates a wrong specification of the model. We can conclude that the relationships between attitude and its determinants are therefore wrongfully specified, but other tests were conducted to ensure that this conclusion is correct.

Findings indicate that many respondents a preference for practicing TDD, highlighting how it has improved their software development practice. A common theme that kept recurring was how it has improved the quality of the software developed and reduced the number of defects that have been found. Some also mentioned how it helped them in improving the design of the code because it forces one to "think about behaviour before thinking about implementation". An example of such a comment was a respondent who said:

"Using outside-in TDD has had the biggest impact on the way I design and write code."

This relates to the attitude towards quality construct proposed in the theoretical model. Because there is a perception that practicing TDD leads to improved quality, this leads to a positive attitude towards TDD, and influences the software developer to intend to practice TDD.

Q. ATTITUDE AND PERCEPTION OF QUALITY

We realised that there seemed to be a positive relationship between an individual developer's attitude towards the perceived difference in quality, and a developer's attitude towards TDD. To statistically test the relationship, first a Kruskal-Wallis test was conducted. The Kruskal-Wallis test resulted in a p-value of $2.2 * 10^{-16}$ which would lead one to reject the null hypothesis of no relationship between attitude towards TDD and perceive difference in quality of software developed. The only concern is that the Kruskal-Wallis chi-squared result came to 485.29, which is fairly high, and may indicate a model which hasn't been specified correctly.

The chi-squared contingency table test from the condensed contingency table resulted in the same p-value of $2.2 * 10^{-16}$ but an even higher chi-squared value of 862.2. So again, while this leads us to conclude that the null hypothesis should be rejected, it still represents a misspecification of the conceptual model.

R. ATTITUDE AND PERCEPTION OF DIFFERENCE IN TIME

In statistically testing the relationship between the two measures, the Kruskal-Wallis test resulted in a p-value of $2.2 * 10^{-16}$ leading us to reject the null hypothesis, and conclude there is a significant relationship between the two variables. Again, there was a high Kruskal-Wallis chi-squared value of 186.9 making implying a model that hasn't been specified correctly. The chi-squared contingency table test of the condensed contingency table resulted in a p-value of $2.2 * 10^{-16}$ and a chi-squared value of 94.901. Again, this leads to the rejection of the null hypothesis, but the high chi-squared value also indicates that there may be an incorrect specification of the relationship in the conceptual model.

VI. CONCLUSION

The literature revealed fascinating results about the influences behind the performance of TDD by developers

around the world. Firstly, what we found intriguing is the literature highlighted numerous advantages and disadvantages of performing TDD. The data that was collected however did not fully support what was posited in the literature review and the conceptual model. On the models; the attitude construct and its determinants (the attitude towards the time taken, the effect on quality, and the maintainability) support the conceptual framework. Similarly, the subjective norm construct and its determinants (perceived attitude of the environment (e.g. the software development team) towards quality, time taken and maintainability) support the conceptual framework. The perceived behavioural control constructs of difficulty and experience do not support the theoretical framework in a statistically significant way.

Secondly, we found that within the comments submitted by the respondents, many mentioned that the difficulty of TDD may be a barrier for people, but as can be seen from the hypotheses, experience and perception of difficulty were actually independent of intention to perform. This shows that perceived difficulty of the task may not be the biggest barrier, to performing TDD, but he attitude of the developer, and their perception of what is acceptable in the environment they work in.

Further, the findings show that there is a relationship between the proposed determinants of attitude and attitude, as well as a relationship between the proposed determinants of subjective norm and the subjective norm.

In the case of attitude though, the ordinal logistic regression that was used could not specify a regression function, indicating that the model was not specified correctly, and that more items may be needed. There was also a significant relationship between attitude and intention to perform TDD. In the case of subjective norm, a relationship was shown between the determinants of subjective norm and subjective norm, and the ordinal logistic regression confirmed that the proposed determinants did indeed have the predicted effects on the subjective norm. Similar to attitude, there exists a statistically significant relationship between subjective norm and intention to perform TDD. Lastly, it is interesting to note that the factors that were substituted for subjective norm, perceived difficulty of TDD and experience, did not have a statistically significant relationship with the intention to perform TDD. This shows that the TPB does not align with the data, and the data ismore suited to the Theory of Reasoned Action, the precursor to TPB, which did not include thePerceived Behavioural Control factor.

REFERENCES

[1] N. Nurmuliani, D. Zowghi, and S. Powell, "Analysis of requirements volatility during software development life cycle," in *Software Engineering Conference, 2004. Proceedings. 2004 Australian*, 2004, pp. 28–37.

[2] T. Javed, M. e Maqsood, and Q. S. Durrani, "A study to investigate the impact of requirements instability on software defects," *ACM Softw. Eng. Notes*, vol. 29, no. 4, pp. 1–7, 2004.

[3] N. N. Bhat, Thirumalesh, "Evaluating the Efficacy of Test-Driven Development : Industrial Case Studies," in *Isese'06,* 2006, pp. 356–363.

[4] B. George, L. Williams, and W. L. George B., "An initial investigation of test driven development in industry," in *Proceedings of the ACM Symposium on Applied Computing*, 2003, pp. 1135–1139.

[5] L. Williams, E. M. Maximilien, and M. Vouk, "Test-driven development as a defect-reduction practice," *ISSRE*, vol. Janua, pp. 34–45, 2003.

[6] S. Wappler and F. Lammermann, "Using Evolutionary Algorithms for the Unit Testing of Object-Oriented Software," in *GECCO'05*, 2005, pp. 1053–1060.

[7] J. Highsmith, "What Is Agile Software Development ?," *Crosstalk, Def. Softw. Eng.*, vol. 15, no. 10, pp. 4–9, 2002.

[8] T. Dybå and T. Dingsøyr, "Empirical studies of agile software development: A systematic review," *Inf. Softw. Technol.*, vol. 50, no. 9–10, pp. 833–859, 2008.

[9] J. Erickson, K. Lyytinen, and K. Siau, "Agile Modeling, Agile Software Development, and Extreme Programming: The State of Research," *J. Database Manag.*, vol. 16, no. 4, pp. 88–100, 2005.

[10] A. Bertolino, "Software Testing Research : Achievements , Challenges , Dreams Software Testing Research : Achievements , Challenges , Dreams," in *FOSE'07*, 2007, no. September.

[11] S. Mathur and S. Malik, "Advancements in the V-Model," *Int. J. Comput. Appl.*, vol. 1, no. 12, pp. 29–34, 2010.

[12] L. Madeyski, "Agile Development Practices-The Impact of Pair Programming and Test-Driven Development on Package Dependencies in Object-Oriented Design–An Experiment," *Lecture Notes in Computer Science*, vol. LNCS 4034. pp. 278–289, 2006.

[13] I. Ajzen, "The Theory of Planned Behavior," *Organ. Behav. Hum. Decis. Process.*, vol. 50, pp. 179–211, 1991.

[14] T. J. Madden, P. S. Ellen, and I. Ajzen, "A Comparison of the Theory of Planned Behavior and the Theory of Reasoned Action," *Soc. Personal. Soc. Psychol.*, vol. 18, no. 1, pp. 3–9, 1992.

[15] V. Savalei and P. M. Bentler, "Structural Equation Modeling," in *Handbook of Marketing Research: Uses, Misuses, and Future Advances*, 2006, pp. 330–364.