ISSN 1846-6168 (Print), ISSN 1848-5588 (Online)
https://doi.org/10.31803/tg-20191104214708

Original scientific paper

THE EMPIRICAL STUDY: ENCOURAGING STUDENTS' INTEREST IN SOFTWARE DEVELOPMENT USING TEST-DRIVEN DEVELOPMENT

Aziz NANTHAAMORNPHONG, Stephane BRESSAN

Abstract: The supply is not matching the demand on the market for software developers. While the enrolment in undergraduate computer science courses is increasing, few students are interested in and committed to becoming software developers. It could be that students are overwhelmed by the software development methodology that they are taught. We are consequently looking for a constructivist approach to software engineering able to effectively engage learners. We empirically evaluated whether test-driven development (TDD) is able to improve the quality of both learning and of software development in the classroom. Although numerous studies have outlined the benefits and effects of TDD in the classroom, none of those studies have focused on measuring students' interest in and attitudes toward using TDD in the classroom. We present a study evaluating the impact of TDD on the engagement and focus of learners of software development in the classroom. The results illustrate that the use of TDD in the classroom encourages learners to engage and focus.

Keywords: empirical software engineering; software engineering education; test-driven development

1 INTRODUCTION

"By 2020, one of every two jobs in science, technology, engineering, and mathematics will be in computing," according to Bobby Schnabel, executive director and chief executive officer of the Association for Computing Machinery [1]. The enrolment in undergraduate computer science courses is increasing. Yet, the demand, the number of jobs in computing, outruns the supply, the number of students graduating in computer science and related fields, according to the report "Assessing and responding to the Growth of Computer Science Undergraduate Enrolments" by the American National Academies of Science, Engineering and Medicine [2]. Stack Overflow 2019 Developer Survey Results [3] suggest that the majority of software developers are Web developers (although this may be a bias inherent to the Stack Overflow community) and that they are not necessarily computing graduates. This highlights the existence of a market for trained software developers able to work on challenging applications such as distributed and embedded systems.

If well managed, this phenomenon is an opportunity for all countries with the right education policies and the right academic curricula and the right methodologies. Developing countries, in particular, can diversify their industry and workforce and further tune into the growing service and knowledge economy. Illustratively, Oichi Okoshi, president and chief executive officer of Toyota Tsusho Nexty Electronics (Thailand) Co., Ltd. believes that "Thailand has a lot of potential to become a hub of automotive software development" [4].

This is however even more challenging than it looks as, while the number of students wanting to study computing and pursue a career in computing is increasing, it is not necessarily the case that a sufficient number of these students are interested in coding and are actually motivated and able to effectively learn to code. In its report [2], the American National Academies of Science, Engineering and Medicine further acknowledges that academically trained software developers end up not being properly equipped to meet the requirements of the software industry.

Typically, the teaching pattern of software development starts with programming fundamentals such as variable declaration and assignment, control structures (e.g., if-then-else, for and while statements) and other language structures (e.g., functions), after which exercises or problems are assigned to the students. This pattern parallels a well-known software development process: The Waterfall Model [5]. In this model, software development begins by acquiring Software Requirements for Software Design. Implementation then starts with code writing. Finally, Testing is performed once the code is complete. Both the teaching methodology and the Waterfall model emphasize a conceptually top-down approach from the abstract to the concrete. Consequently, students focus on executing software with no runtime error. Diving from the height of the software requirement into the code without anything to guide them, students can only rely on a trial and error approach [6] in which debugging is the main activity. They actually generally test their programs with only a handful of test cases after the code is finished. Testing of all operational conditions is rarely performed. Paradoxically, testing is assumed to be completed when the program runs without error. As a result, software development projects fail in practice. In addition, the time spent on software development is increasing because fixing bugs is time consuming, albeit the main activity.

In this top-down approach, students misunderstand system requirements or cannot find a solution to the assigned global problem. They cannot write code to meet the requirements and get hopelessly lost in meaningless debugging of program errors and random attempts to fix failures to run.

Piaget's theory of cognitive development [7] was the seminal impulse to constructivist pedagogy [8]. Constructivist instructional approaches try and actively involve learners in the construction process of meaning and knowledge. Clearly, constructivism suggests bottom-up approaches. We look among the modern agile software engineering methods one that can provide both the active involvement and the rigor needed.

Test-Driven Development (TDD) is a method of software development and is part of Extreme Programming, which is an agile software development method [9]. Agile software development has been a popular method in the global software development community since 2001. Because of the complexity of software systems, the popularity of agile development has been increasing. Agile development can synchronize communications among developers, users and customers regarding software requirements. Essentially, agile software development operates in a short cycle, and the developer selects numerous functionalities to develop in each cycle. Once the chosen functionality has been developed and delivered to the customers, those customers can rapidly perceive the tangible outcomes that satisfy their requirements. The customers can give feedback and suggestions from the beginning of the development process, significantly decreasing the risk that the system will not satisfy their requirements. The result is higher software quality [10]. Additionally, TDD is recommended for teaching students about software development because TDD helps improve the software development process [11], [12]. The researchers seem to agree that TDD is an enjoyable way to work on the programming task [13]. The TDD approach involves writing the test code before the functional code. Therefore, the developers write the code for testing before writing the actual functional code and test the production code using the created testing code. Programming is done in cycles for every function. After passing the test, the developers can improve the quality of the code, since the work in this stage is only to ensure that the code works properly; the result can be a poor-quality program. Updating the code is called Refactoring. The refactoring process changes the internal structure of the software but does not alter the functionality and is done to enhance the quality of the software – for example, by addressing feature envy – which can be refactored using 'move method' [14, 15]. Repetitive tasks can be implemented as a superclass; any function can then call this super-class from the regular class, helping prevent duplication and reduce lines of code.

As far as we are aware, there is a lack of empirical evidence presented in the literature as to whether TDD can be instrumental in sparkling student, particularly undergraduate student, interest in software development. Therefore, we are interested in TDD adoption by students in software development studies. The main research question of this study is as follows: *Can test-driven development encourage student interest in software development?*

2 RELATED WORK

Janzen suggested that academics benefit from adopting the TDD teaching approach to teach testing. Textbooks and instructional materials are expected to incorporate this approach, since TDD can help improve software development [16]. In later years, Janzen and Saiedia [17] investigated whether the use of TDD in teaching and learning can lead to better software systems. The study indicates that although software quality is better, the research still lacks an examination of students' feelings and attitudes about the use of TDD for assigned tasks. Another study by Pencur et al. [18] involved two groups of developers. Group 1, Iterative test-last (ITL), used the traditional testing method, which is to test the program after the code was finished. Group 2 incorporated TDD. The comparison shows that TDD is not substantially different from ITL on code coverage but is significantly different in quality in that the use of TDD results in better code quality.

Edwards [19] evaluated the effects of using TDD when teaching undergraduate students in the classroom and found that students demonstrated the effectiveness and accuracy of the test. The students obtained higher scores and encountered fewer programming errors. Another study was conducted by Buffardi with the goal of motivating new programmers to use TDD [20]. The result indicates that TDD helps produce higher-quality code.

The study of TDD use among professional developers shows that it is a well-respected tool in the software industry [21]. Preliminary reports on TDD testing with experts in software development illustrate that TDD helps improve software quality and encourages rigorous testing. This evidence suggests that the trend among developers in the use of TDD enhances testing capabilities in the software industry [22], [23].

A comparative study of TDD use among university students and expert groups that utilized TDD in real-world environments showed that compared to the student group, the expert group paid more attention to and preferred to use TDD [24].

Desai et al. [11] suggested that TDD is a possible solution for improving students' software testing skills. They also showed that TDD exposes students to analytical and comprehension skills, both of which are needed in software testing. Mäkinen and Münch [25] performed a comparative analysis of empirical studies on the impact and potential of using TDD through observation. The results from the code quality comparison suggested that TDD can help reduce errors and improve both quality and maintenance time.

Causevi et al. [26] reviewed the literature on the limitations of adopting TDD in the software industry. Based on empirical evidence from 48 studies, the researchers identified the following factors as limiting the industrial adoption of TDD: 1) Increased development time; 2) Insufficient TDD experience/knowledge; 3) Insufficient design; 4) Insufficient developer testing skills; 5) Insufficient adherence to TDD protocol; 6) Domain- and tool-specific limitations; and Legacy code.

3 RESEARCH METHODOLOGY

To address the research question described in Section 1, we identify the following hypotheses: H_{null}: the student's interest with the TDD method is not different from student interest with the waterfall model; $H_{alternative}$: the student's interest with the TDD method is different from student interest with the waterfall model.

In this study, we primarily conducted controlled experiments and interviews. The following subsections will detail the experimental setting and data-collection method.

3.1 Experimental Variables

Based on our hypotheses, we have set an independent variable as the *software development* method, including 1) the waterfall model and 2) the TDD method. The dependent variable is *student's interest* in software development.

The fields of psychology and education students have very different definitions of "interest." In this study, we defined interest as follows: *Interest is a feeling of satisfaction or attitude toward anything, any idea or any situation with a tendency of wanting to approach and know about that thing. This leads to perseverance to achieve the goal. Interest is the driving force that motivates people to do anything or represents the tendency for the person to choose. In terms of academics, interest is linked to academic success and is a key element in career skill development* [27], [28]. We used this definition to build the interest indicators for measuring the dependent variable. We have identified 5 indicators of interest as follows:

1) Enthusiasm about what is interesting: The purpose of this metric is to identify what students learned before, during, and after the experiment. Being enthusiastic about what interests the students is a driving force that motivates them to choose what they want to study. For example, if students are interested in programming, their interest might motivate them to pursue a career in software development.
2) Self-study: This metric provides an overview of what students learned more by themselves, which might be related to software development. It might also be related to software development but is not in the curriculum. For example, students can learn more about Python, which is within the scope of software development but is not taught in any courses at the university.
3) Exercises: This indicator studies information from student exercises. The topics for the exercises may be those in which the students are interested. For example, some students might choose to do the system analysis exercise but not the programming exercise.
4) Exercise duration: The purpose of this indicator is to study the time spent on the exercises because the amount of time spent by each student is different. For example, a student who takes more time to do exercises or does more exercises than others might want to understand the course material.
5) Self-study duration: This metric is intended to study the time used for self-study. For example, students who are interested in learning more about software development in Java may spend more time studying this topic by themselves than other subjects.

Once the interest indicator has been identified, we adopted metrics to design the questionnaire, which consists of open- ended questions and Likert-scale questions.

3.2 Data Collection

The data were collected using two methods: 1) questionnaires and 2) interviews. The questionnaires and interview questions include both quantitative and qualitative data collection. The data from the students before and after the experiment were analysed both to determine whether the use of TDD in software development can encourage student interest and to study the impact of TDD on education.

Each questionnaire consists of both Likert-scale questions and open-ended questions. The interview questions are based on the open-ended questionnaire because we required in-depth information from the students that might not have been present in their questionnaire answers. This strategy was used because the interviewees are free to answer questions, and answering in their own words is easier than writing formal responses; additionally, they can offer more comprehensive information. Details about the design of the questionnaires and interviews questions are described in the paragraphs to follow.

Questionnaires We developed the questionnaire to provide pre-test questions, post-test questions and TDD exercises. The questions in the questionnaire were examined by a software engineering expert with experience in software development and TDD research. There are two formats of the questions: 1) The open-ended question collects the student's basic information and software development experience in object-oriented programming (OOP) courses and Java projects developed using the waterfall model. It also gathers information about the students' interests in the field of software development, and 2) The Likert-scale question collects information about students' interests in software development. There are five levels of answers: 1) Not at all interested, 2) Slightly interested, 3) Somewhat interested, 4) Very interested, and 5) Extremely interested.

This questionnaire was developed from our defined indicators of interest. Tab. 1-3 show the developed questions with indicators.

Interviews The interview was designed to be semi-structured. In a semi-structured interview, the interviewer can append or alter the questions during the interview. The interview questions are similar to the open-ended questionnaires but are intended to collect additional information from the students' pre-test and post-test. These interviews collect more comprehensive information than the questionnaire since speaking is much easier than writing. In addition, the interview questions are more flexible; thus, the interviewer and interviewee can communicate better, especially when using technical terms.

3.3 PARTICIPANTS

The samples in this study consist of 52 students in the second-year of a software engineering program. We selected these students because this study involves testing and refactoring, which is part of the students' prior software engineering classes. Additionally, the second-year students have been taught basic programming and OOP. As a result, the students could understand the questions from interviews and questionnaires.

3.4 Controlled Experiment

The controlled experiment consists of four stages, including the pre-test, TDD training, experiment treatments, and post-test.

Pre-test The pre-experiment was performed using questionnaires and interviews. The questionnaire survey was on paper, and the students were asked to answer the survey questions in the classroom beforehand. The students had 30 minutes to complete the questionnaire, which consists of 16 open-ended and closed-ended questions.

The interview questioning was in a semi-structured format in which the interviews were planned in advance. The interviewers included four researchers who were trained by the first author. The content of the interview concerned the students' interest in software development and the software development model used before attending the TDD presentation. If necessary, each interviewer could ask questions in addition to the specified questions. The interview was conducted in a one-to-one format. The interviews were recorded via audio recording and note taking.

Prior to performing the experiment, we conducted the pilot study using designed questionnaires and interviews with 32 students who were not included in the main experiment. The students spent an average of 15 minutes to complete the questionnaire and 20 minutes to be interviewed. The pilot study aimed to examine the internal consistency reliability of the Likert scale questionnaire. We checked the reliability by calculating the Cronbach's alpha coefficient based on the 32 respondents. The calculated coefficient was 0.89, which shows that the magnitude of the consistency of the questions was in the exact same direction (greater than 0.7). Based on this pilot study, we did not find any problems stemming from the questionnaire and interview.

TDD training One week after completing the questionnaire and interview, the second author taught TDD to the students. The topics covered in the lecture were as follows: 1) basic knowledge of TDD, 2) TDD and unit testing, 3) TDD software development tools, and 4) TDD software development examples. All 52 students participated in the lecture session.

Experiment treatments The students were assigned to develop the FizzBuzz program [29], which is an easy program that enabled students to understand the steps in the TDD process. FizzBuzz is a game that helps one learn division by assigning one integer value in four cases as follows:

- **Case 1** for an integer that can divided by 3, display the word Fiz;
- **Case 2** for an integer that can divided by 5, display the word Buzz;
- **Case 3** for an integer that can divided by 3 and 5, display the word FizzBuzz;
- **Case 4** for an integer that cannot be divided by either of the three cases above, display the input integer.

We asked each participant to implement the FizzBuzz game as two versions. For the first version, each participant developed the program using the waterfall model (non-TDD version). In the second version, the program was developed with TDD process (TDD version).

To implement the non-TDD version, the participants could choose any programming software development tools (e.g., NetBean, Eclipse, IntelliJ). For the TDD version, the participants were required to develop a FizzBuzz program using the TDD method with a JUnit plugin (version 4.0) integrated with Eclipse. The participants were asked to install JUnit within Eclipse on their laboratory computers by themselves and test whether JUnit worked properly. Once JUnit was successfully installed, the participants began implementing the program with the TDD process.

The participants developed the non-TDD and TDD versions in the morning (9:00 AM-11:00 AM) and afternoon (1:00-3:00 PM), respectively.

Post-test One day after the assigned tasks were completed, we conducted a survey using questionnaires and interviews. The post-test questionnaire was similar to the pre-test questionnaire, but there were additional questions about the effect of using TDD. The participants were asked to complete the questionnaires within 30 minutes.

4 RESULTS AND DISCUSSION

To examine how well the Likert-scale questions represented the research construct, we employed a statistical method known as confirmatory factor analysis (CFA [30]). The 8 Likert-question scores from the pre- and post-experiments were analysed with the CFA.

First, we measured how our data are suited for a factor analysis. To do so, we used Kaiser-Mayer-Olkin (*KMO*) and Bartlett's test. Typically, if the value of $KMO < 0.5$, it is considered unsuitable to use a factor analysis. Barlett's test is a statistical hypothesis test. In this study, we assume Barlett's test in the questionnaire as follows:

H_{null}: The questions are not related

$H_{alternative}$: The questions are related

Based on an analysis of the Likert-scale questionnaire, the *KMO* result equals 0.858, which is greater than 0.5; thus, the data are suitable for use with factor analysis techniques.

Additionally, the value of Barlett's test is less than 0.05, so H_{null} is rejected and $H_{alternative}$ is accepted. As a result, it can be concluded that the indicators for the questions are correlated, and therefore, the factor analysis can be used. Next, we used principal component analysis (PCA) to examine whether all the questions included all measures of interest of the students. With the recommendation of Kaiser [31], components with eigenvalues greater than 1 should be retained. In this case, we retain only component 1 as the principal factor.

Table 1 Pre-test questions

Questions	Indicators
1. Grade in OOP subject	Enthusiasm about topics of interest
2. What is your project in OOP about?	
3. What is your reason for enrolling in software engineering?	Self-study
4. What do you know about "software developers"?	
5. Are you interested in software development? Why? (Interest is a feeling of satisfaction, curiosity, pursuit of enjoyment, personal satisfaction toward an activity that leads to perseverance to achieve the goal)	
6. What is your language of interest?	
7. How did you test the system in your previous class project?	
8. Based on your past studies, which subjects are significant in the study of software engineering? (Significant subjects can affect your later education or may be required in your software engineering career)	
9. Which subjects have interested you in your previous software engineering classes? (Interest is the satisfaction of learning a subject that you would like to learn more about or study by yourself)	
10. If training is provided as a seminar or workshop, which topic would like to study? Why?	
11. What is your expected career after graduation?	
12. What is the average amount of time you spend on self-study? (average time per week)	
13. What is your preferred subject when you study by yourself?	
14. If training is provided as a seminar or workshop, which topic would you like to study? Why?	
15. Which kind of an assigned exercise?	Exercises
16. What is the average amount of time you spend to develop software from each assigned exercise? (average time per week)	Exercise duration

Table 2 Likert-Scale Questions

Questions	Indicators
1. You are interested in learning software development.	Enthusiasm about topics of interest
2. You are interested in practicing software development from the exercises (Software development practice is in the form of a coding exercise).	
3. You are interested in practicing software development via self-study (Software development practice is in the form of a coding exercise).	
4. You are interested in learning about software engineering by yourself (Studying software engineering topics such as learning a new method of software development or a new programming language, etc.).	
5. You are interested in modifying the interface from the assigned exercise.	
6. You are interested in participating in questions and answers during class.	
7. You are interested in self-learning.	Self-study and self-study duration
8. You are interested in spending time to complete the exercise.	Exercises and exercise duration

We then calculated a factor loading that illustrates the relationship between variables (questions) and the principal component (component 1). The loading factor values of all variables are greater than 0.5; thus, all variables have relationships within the principal component. Additionally, the Cronbach's alpha values of all questions are 0.929, suggesting that all five indicators were within the same factor and that the variables were highly correlated.

Table 3 Post-Test Questions

Questions	Indicators
1. Does the use of TDD affect your interest in software development? Why?	Enthusiasm about topics of interest
2. How does the use of TDD affect your programming experience in the laboratory?	
3. When using the waterfall model or TDD in the same exercise, which method produces fewer errors? Why?	
4. In the future, if you are required to create a project in class, which test method will you use?	
5. Which method do you believe provides more details in program testing? Why?	
6. From your perspective, does TDD contribute to software development? Why?	

4.1 Likert-Scale Questionnaire

Based on the 52 responses, the average score of each question was summarized by comparison of the mean scores of the pre- and post-experiments, which is shown in Fig. 1.

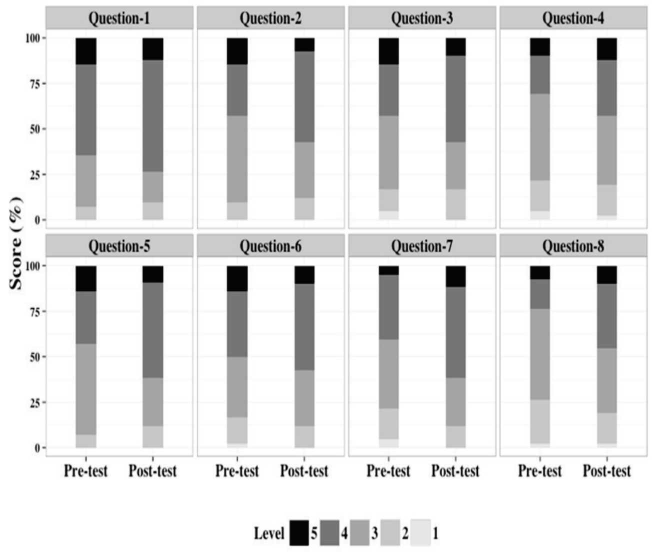

Figure 1 The Likert-scale questionnaire's results

The highlighted results of each question are as follows:

Question 1 Students are interested in learning software development. Average scores of 4 for TDD (very interested) were higher than the waterfall model by 11.9%.

Question 2 Students are interested in software development practice with exercises (The software development exercise serves as programming practice). The study found that after TDD, average scores of 4 (very interested) were higher than the waterfall model by 21.30%.

Question 3 Students are interested in software development practice with self-study (Software development exercises serve as the programming practice). From the questionnaire after TDD study, average scores of 4 (very interested) were higher than the waterfall model by 19.05%, and average scores of 1 (not at all interested) decreased by -4.76%.

Question 4 Students are interested in learning more about software engineering by themselves (additional software engineering studies). Average scores of 5 (extremely interested) increased by 2.38%, and average scores of 4 (very interested) increased by 9.52%. Average scores of 1 (not at all interested) decreased by -2.38%.

Question 5 Students are interested in modifying the interface display from the exercise. Average scores of 4 (very interested) increased by 11.91% after using TDD. Average scores of level 1 (not at all interested) decreased by -2.38%.

Question 6 Students are interested in participating in the question & answer session in class. The results after TDD, average scores of 5 (extremely interested) were higher than the waterfall model by 2.38%, and scores of 4 (very interested) were higher than the waterfall model by 19.04%.

Question 7 Students are interested in self-learning. The results after TDD study shows that average scores of 5 (extremely interested) increased by 7.14% and scores of 4 (very interested) increased by 14.29%. Average scores of 1 (not at all interested) decreased by -4.76%.

Question 8 Students are interested in spending time working on the exercises. From the results of the questionnaire after TDD study, average scores of 4 were higher than the waterfall model by 23.81%.

In addition to the analysis with the Likert-scale questionnaire, we performed a statistical analysis with the paired Student's T-test to prove our research hypotheses.

The paired Student T-tests were analysed to compare the results of the pre-test and post-test total score of each participant (1 point denotes not at all interested and 5 denotes "extremely interested"). The results obtained from the paired Student's T-test showed that the mean scores of the post-test are statistically significantly higher than the pre-test (p-value = 0.00365 at a confidence level of 95%). Therefore, H_{null} was rejected, whereas $H_{alternative}$ was accepted, which means that the interest of the students who learned about TDD is different from their interest associated with the waterfall model. Based on the prior results, this may imply that the use of TDD can encourage a sample group to become more interested in software development.

4.2 Open-Ended Questions and Interviews

The answers for the open-ended questionnaires and interviews were the descriptive data, which were analysed using a coding analysis method with ATLAS.ti software (http://atlasti.com/). We summarized the findings as follows:

Finding 1 Effect on students' interest in using TDD in software development. From data collected from 52 questionnaires and interviews, TDD affects student interest in software development by 86%. Based on the responses of students who found that TDD affected their interest in software development, we classified the benefits of using TDD into 3 areas:

1) Errors found in the program. The respondents reported that TDD helped them easily check for program errors; thus, the number of errors was reduced. Additionally, they could find program errors from the beginning of the development process.
2) Program quality. In terms of the quality of the code, the respondents indicated that TDD helped them reduce the complexity because refactoring methods were used during the development. Additionally, refactoring reduced the number of lines of code. Some respondents mentioned that the code they wrote was more understandable after refactoring.
3) The time spent using TDD for programming. The respondents felt that TDD saved time because there were fewer steps. However, some of respondents mentioned that they were more familiar with the waterfall model than with TDD.

Finding 2 The selection of software development methods between the waterfall model and TDD. Based on the analysis, we divided the selection of software development methods into 4 groups: 1) Using TDD in the exercise (73%); 2) Using the waterfall model in the exercise (15%); 3) Writing the test code and actual code without refactoring (11%); and 4) The methods are not different (1%).

Finding 3 Limitations of using TDD in software development from a student perspective. We have included several limitations that some students mentioned with regard to using TDD in the lab during and after the experiment as follows: the students thought that the developers who write the code should not write the test code; there are several steps in TDD, so additional study may be required, but they are more interested in programming; they cannot use the experimental tool (Eclipse); if they learn more about TDD, interest will increase with their confidence; they are familiar with the waterfall model for writing the code and fixing the errors at the run time; they think of TDD as only a new software development process and is not interesting as new technology; they have a better understanding of both traditional software development and the waterfall model method; and they need to reorganize their programming to use the TDD method.

4.3 Discussion

We have set all five indicators of interest and verified the indicators with confirmatory factor analysis through the Likert-scale questionnaires. The results of the factor analysis show that the questionnaires can be used to ask questions to measure the interest of the experimental groups.

Based on the questionnaires and interviews from the pre-experiment, students who used the waterfall model of software development were not interested in system testing and project testing in class. The students performed the tests by entering random data for testing purposes, but they did not test the execution process of the code internally. The students made modifications to the system when the errors were encountered at run time. Additionally, based on the responses to the questionnaires and interviews, the students understand that using Exception try-catch in Java is one way to test a system.

From the questionnaires and the interviews about additional study, 70% of the students were interested in additional programming studies. However, the students were not interested in studying system testing and the programming process or development methods for software.

The study found that the use of TDD in class contributed to students' interest in software development. The questionnaires and interviews in pre- and post-experiment showed an increasing number of students who were interested in software development. The results of this study indicate that TDD could help encourage students' interest in software development since today's students are less interested in being software developers. The use of TDD could help encourage student interest and enables students to produce more efficient software and better quality code. This evidence paves a way to changing the learning style to produce better software developers into the industry.

These results are similar to those in previous works. For example, previous studies [16], [18], [20], [32] suggested that TDD should be used in a class to provide students with software development skills because it can help improve the quality of software development. In addition, the use of TDD shows that students make fewer programming errors [19].

5 THREATS TO VALIDITY

The threats to this study's validity are divided into *Construct*, *Internal* and *External* threats.

5.1 Construct Validity

Construct validity is a consideration if the concepts being studied are correct and all selected studies can answer the research questions. In this study, we measured student interest using questionnaires, and we studied various theories related to the definition of "interest". Nevertheless, there are various definitions of interest in the existing literature, and there is no accepted definition of interest in software engineering research. Therefore, we combined existing definitions as the indicators to formulate the questions in the questionnaire. To reduce this threat, the research questionnaire was reviewed by a software engineering expert who works in software engineering education. Additionally, the questionnaire was determined with confirmatory factor analysis, and the results suggested that the questions involved the same factor.

5.2 Internal Validity

Internal validity focuses on the results of experiments. This study is an experimental study that uses designed questionnaires and interviews. Prior to the experiment, a pilot study was conducted to check the questionnaires and interview questions in terms of whether the students could understand the exact questions. In addition, before the actual experiment, we informed the participants that the results of the questionnaire and the interview had no effect on their scores in the studied subjects. Since the interview was semi-structured, the participants' questions might have been different. We attempted to reduce this risk by asking the interviewers to discuss and practice together. Additionally, the lessons learned from the pilot study mitigated this threat.

5.3 External Validity

External validity focuses on the generalizability of the results of this study. The experiments in the study used a sample of students in software engineering who have completed basic programming classes in OOP and Java programming. Additionally, the program we used in this study is simple. The replicated experiment using larger programs and other student fields can help mitigate these threats.

6 CONCLUSION

This study aimed to enable students to learn and use TDD to encourage their interest using experimental methods and data collection from questionnaires and interviews. The results of the experiment have proven the research hypothesis: the students who were taught to use TDD were interested in software development at a different level than those who used the waterfall model. The students who used TDD in their studies were more interested in software development. Based on this study, we believe that the results of the research will help investigators in software engineering encourage interest in software development. Consequently, it also focused on software development methods and the emotional attitudes of software developers. The study is beneficial to educational personnel, especially in software engineering education, to change teaching methods and motivate students to become more interested in software development. Finally, it provides empirical evidence in software engineering regarding the use of TDD in the classroom, including constraints and recommendations for other researchers or educational personnel using TDD in teaching and learning.

Remark

The article was orally presented at the 23rd International Computer Science and Engineering Conference (ICSEC2019).

5 REFERENCES

[1] Kaczmarczyk, L. & Dopplick, R. (2014). ACM Report: Preparing Students for Computing Workforce Needs in the U.S. *ACM SIGCSE Bulletin, 46*(2), 8-8. https://doi.org/10.1145/2620668.2620675

[2] https://www.nap.edu/catalog/24926/assessing-and-responding-to-the-growth-of-computer-science-undergraduate-enrollments

[3] https://insights.stackoverflow.com/survey/2019

[4] http://www.nationmultimedia.com/detail/Economy/ 30344732

[5] Royce, W. W. (1987). Managing the development of large software systems: concepts and techniques. In *Proceedings of*

the 9th international conference on Software Engineering (pp. 328-338)

[6] Edwards, S. H. (2004, March). Using software testing to move students from trial-and-error to reflection-in-action. *ACM SIGCSE Bulletin, 36*(1), 26-30. https://doi.org/10.1145/1028174.971312

[7] Piaget, J. (1977). *The role of action in the development of thinking*. In Knowledge and development. Springer, Boston, MA, 17-42.

[8] Cakir, M. (2008). Constructivist approaches to learning in science and their implications for science pedagogy: A literature review. *International journal of environmental and science education*, *3*(4), 193-206. https://doi.org/10.1007/978-1-4684-2547-5_2

[9] Beck, K. (2003). *Test-driven development: by example*. Addison-Wesley Professional.

[10] Karac, I. & Turhan, B. (2018). What Do We (Really) Know about Test-Driven Development? *IEEE Software, 35*(4), 81-85. https://doi.org/10.1109/MS.2018.2801554

[11] Desai, C., Janzen, D., & Savage, K. (2008). A survey of evidence for test-driven development in academia. *ACM SIGCSE Bulletin, 40*(2), 97-101. https://doi.org/10.1145/1383602.1383644

[12] Munir, H., Moayyed, M., & Petersen, K. (2014). Considering rigor and relevance when evaluating test driven development: A systematic review. *Information and Software Technology, 56*(4), 375-394. https://doi.org/10.1016/j.infsof.2014.01.002

[13] Scanniello, G., Romano, S., Fucci, D., Turhan, B., & Juristo, N. (2016, April). Students' and professionals' perceptions of test-driven development: a focus group study. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, 1422-1427. https://doi.org/10.1145/2851613.2851778

[14] Mumtaz, H., Alshayeb, M., Mahmood, S., & Niazi, M. (2018). An empirical study to improve software security through the application of code refactoring. *Information and Software Technology*, 96, 112-125. https://doi.org/10.1016/j.infsof.2017.11.010

[15] Tosun, A., Dieste, O., Fucci, D., Vegas, S., Turhan, B., Erdogmus, H., & Juristo, N. (2017). An industry experiment on the effects of test-driven development on external quality and productivity. *Empirical Software Engineering, 22*(6), 2763-2805. https://doi.org/10.1007/s10664-016-9490-0

[16] Janzen, D. & Saiedian, H. (2005). Test-driven development concepts, taxonomy, and future direction. *Computer, 38*(9), 43-50. https://doi.org/10.1109/MC.2005.314

[17] Janzen, D., & Saiedian, H. (2008). Test-driven learning in early programming courses. *ACM SIGCSE Bulletin, 40*(1), 532-536. https://doi.org/10.1145/1352322.1352315

[18] Pancur, M., Ciglaric, M., Trampus, M., & Vidmar, T. (2003, Septe). Towards empirical evaluation of test-driven development in a university environment. In *The IEEE Region 8 EUROCON 2003. Computer as a Tool*, 2, 83-86.

[19] Edwards, S. H. (2003, August). Using test-driven development in the classroom: Providing students with automatic, concrete feedback on performance. In *Proceedings of the international conference on education and information systems: technologies and applications EISTA* (Vol. 3).

[20] Buffardi, K. (2012, September). Understanding and persuading adherence to test-driven development. In *Proceedings of the ninth annual international conference on International computing education research*, 155-156. https://doi.org/10.1145/2361276.2361308

[21] Kropp, M., Meier, A., Anslow, C., & Biddle, R. (2018, June). Satisfaction, practices, and influences in agile software development. In *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering*, 112-121. https://doi.org/10.1145/3210459.3210470

[22] Bissi, W., Neto, A. G. S. S., & Emer, M. C. F. P. (2016). The effects of test driven development on internal quality, external quality and productivity: A systematic review. *Information and Software Technology*, *74*, 45-54. https://doi.org/10.1016/j.infsof.2016.02.004

[23] Munir, H., Wnuk, K., Petersen, K., & Moayyed, M. (2014, May). An experimental evaluation of test driven development vs. test-last development with industry professionals. In *proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering* (p. 50). https://doi.org/10.1145/2601248.2601267

[24] Müller, M. M. & Höfer, A. (2007). The effect of experience on the test-driven development process. *Empirical Software Engineering, 12*(6), 593-615. https://doi.org/10.1007/s10664-007-9048-2

[25] Mäkinen, S. & Münch, J. (2014, January). Effects of test-driven development: A comparative analysis of empirical studies. In *International Conference on Software Quality*, Springer, Cham, 155-169. https://doi.org/10.1007/978-3-319-03602-1_10

[26] Causevic, A., Sundmark, D., & Punnekkat, S. (2011, March). Factors limiting industrial adoption of test driven development: A systematic review. In *2011 Fourth IEEE International Conference on Software Testing, Verification and Validation*, 337-346. https://doi.org/10.1109/ICST.2011.19

[27] Dwyer, K. K. (1993). Using the 4MAT System Learning Styles Model to Teach Persuasive Speaking in the Basic Speech Course.

[28] Charters, W. W. & Good, C. V. (1945). The Dictionary of Education. *The Phi Delta Kappan, 27*(1), 5-7.

[29] Fenton, S. (2018). Automated Testing. In *Pro TypeScript*. Apress, Berkeley, CA, 245-256. https://doi.org/10.1007/978-1-4842-3249-1_10

[30] Fox, R. J. (2010). *Confirmatory factor analysis*. Wiley International Encyclopedia of Marketing. https://doi.org/10.1002/9781444316568.wiem02060

[31] Kaiser, H. F. (1960). The application of electronic computers to factor analysis. *Educational and psychological measurement, 20*(1), 141-151. https://doi.org/10.1177/001316446002000116

[32] Romano, S., Fucci, D., Baldassarre, M. T., Caivano, D., & Scanniello, G. (2019). An Empirical Assessment on Affective Reactions of Novice Developers when Applying Test-Driven Development. *arXiv preprint arXiv:1907.12290*. https://doi.org/10.1007/978-3-030-35333-9_1

Authors' contacts:

Aziz NANTHAAMORNPHONG
College of Computing, Prince of Songkla University,
80 Moo1 Vichitsongkram Road, Kathu, Phuket, 83120, Thailand
aziz.n@phuket.psu.ac.th

Stephane BRESSAN
School of Computing, National University of Singapore,
21 Lower Kent Ridge Road, Singapore 119077
steph@comp.nus.edu.sg