International Journal of GEOMATE, Sept., 2019 Vol.17, Issue 61, pp. 149 - 155
ISSN: 2186-2982 (P), 2186-2990 (O), Japan, DOI: https://doi.org/10.21660/2019.61.8220
Special Issue on Science, Engineering & Environment

DEFECT-DRIVEN DEVELOPMENT: A NEW SOFTWARE DEVELOPMENT MODEL FOR BEGINNERS

* Wacharapong Nachiengmai[1,2], Sakgasit Ramingwong[3], Kenneth Cosh[3], Lachana Ramingwong[3], and Narissara Eiamkanitchat[3]

[1,3] Department of Computer Engineering, Faculty of Engineering, Chiang Mai University, Thailand;
[2] Graduate School, Chiang Mai University;

*Corresponding Author, Received: 30 Oct. 2018, Revised: 24 Dec. 2018, Accepted: 23 Jan. 2019

ABSTRACT: Software development is challenging. It is normal for software developers to find some problems with their software design, especially during their beginner days. This usually involves simple and repetitious defects which subtly hamper their overall productivity. Defect-driven development (DDD) is a concept proposed to tackle such problems. DDD utilizes the benefits of software defect knowledge base by collecting defects data from experienced programmers and teach beginners to avoid these problems. In this way, the beginners can proactively prevent the defects and subsequently produce more high-quality software. DDD concept can be efficiently adapted to either traditional software development such as the Waterfall and Spiral model, or the more modern concepts such as Scrum or Test-driven Development. This research implemented the DDD concept on undergraduate students and compared their performance with the generic personal software process. A total of seventy-seven undergraduate students from information technology departments participated in this experiment. The experiment was organized in 3 batches in order to minimize potential discrepancies in the results. The result unanimously reveals that the students who implemented DDD had a significantly higher yield on defect removal. Although the time spent to finish each project in the DDD group were higher as expected, they were surprisingly not statistically different from the students who implemented a generic process. This suggests that DDD is a promising concept of software development.

Keywords: Defect-driven development, Software development model, Quality software development, Personal software process, Personal process improvement.

1. INTRODUCTION

In academic, during the beginning of general software development programs, students take courses to increase their comprehension of how the software works in real-world situations [1]. Gradually, they keep on practicing to gain more understanding of the discipline. As their experience grows, they learn how to prioritize tasks, complete their job and, simultaneously, learn how to avoid causing software defects.

The software quality process focuses on controlling product quality and aims to produce non-defective or less-defective products. In real situations, defects can be created at every stage of software process [2]. For example, the defects could be originated by stakeholders, the product owner, or the software development team since the requirement engineering phase. Moreover, different environments could be the cause of errors, e.g., hardware specification, platform and the social environment, including culture and tradition, etc.

The general software process consists of 5 steps as follows: requirement analysis, design, construction, test, and delivery and maintenance. Research in 1992 [2] reported that the defects can occur in every phase of the software process. Those software defects could be avoided with increasing the experience of software developers. With a sufficient level of cautiousness and experience, engineers are more likely to develop their code without repeating their old mistakes.

As mentioned earlier, inexperienced software developers tend to create more simple defects than experienced ones. In addition, such defects may be caused by the changes in the development environment. The defect format normally occurs in the repetitious and similar format [2]. So, this research focuses on whether the beginner software developers can use the defect knowledge from experienced software developers to decrease defects in their projects.

This research attempts to introduce a new approach which facilitates the software process for software developers, especially the beginners. Its objective is to decrease potential errors of the products produced by novice developers. Several tools such as software defect pattern are used in this concept.

1.1 The Concept of Defect-driven Development

The concept of "Defect-driven Development (DDD)" that uses the knowledge of software defects

to preventively drive the software process. The knowledge base of software defects is collected from every step of the software process by experienced software developers. Then it is normalized to a standard format and rearranged as the software defects pattern for beginners.

The principle of the Defect-driven Development (DDD) focuses on proactive activities to check the design and types of error that it might lead to and how to avoid them before coding. This is done by referencing software defect knowledge that was previously collected from experienced software developers. Software developers can use defect information in the design phase to decide either to deal with those defects or redesign that software to avoid problems. DDD's objective is similar to those of Test-driven Development (TDD) hoping that developers foresee the potential problems before the coding stage. The difference is TDD involves a design of unit tests before coding which may not be a natural process for beginners; while DDD more subtly adds a defect checklist during the design. This arguably makes a slight but important change in the process and is likely to be more comfortable for beginners. Yet, both concepts can be implemented simultaneously.

This idea proposes the benefits of using the software defect knowledge from the expertise for producing a framework for beginner software developers. Moreover, this can provide some basic suggestions on how to solve common problems and beginners could learn how to develop software together with software defect management. These would entail the quality software developers in the future.

Each symbol in detail-design is mapped to the category of a software defect in the knowledge of software defect. Then, the system would show defect information that is related to the function in the designed format for example in a checklist or table, etc.

2. RELATED WORK

2.1 Test-driven Development

"Test-driven development (TDD)" [3] or "Test-first development" is one of the techniques that is proposed in "Extreme Programming Model" [4]. It has different steps from general software processes. Developers who implement TDD will create unit tests before the program coding stage. This method will drive programmers to be conscious about software defects first. So, this concept proposes to decrease the error of the products.

The research from IBM Corporation and North Carolina State University reported that projects which applied TDD can reduce defects by 40% when compared with others that use the general process [5]. The research claimed that TDD decreases the quantity of defect and it also influences a proper design of software. In addition, TDD improves communication among the development team and business as well [6].

Another research reports that TDD is less efficient in terms of defect detection compared to code inspection technique. TDD is chosen for the reason of budget because it can save costs compared with code inspection [7]. Lastly, a study reported that TDD is not different from traditional software development in 3 indexes including (i) programming speed, (ii) program reliability, and (iii) program understanding measured as proper reuse of existing methods [8].

2.2 Software Defect Taxonomy

Controlling defects is one of the most important aspects of software quality management. There are many researchers that study on the nature of the software defects, particularly in defect classification. One group of researchers [2] present their idea for classifying software defect by using cause-effect analysis. They collect feedbacks on defects from the software developers. This includes the phase of defects injection, the cause of the defect and the effect of those defects.

The result of this study demonstrates 7 classes of defect including, Function, Interface, Checking, Assignment, Timing/Serialization, Documentation and Algorithm [2]. They are distributed in all of the stages of a software process. It is defined as "Orthogonal Defect Classification (ODC)"

ODC was used to implement in many studies of software engineering areas, i.e., to classify software defect in a specific phase of software process or using for software defects prediction, etc. [2]. The example of studies that use ODC to implement in their research is the research in 2010 [9], which illustrates the new concept of defect classification for black-box testing. In addition, it demonstrates that the ODC is not applicable to black-box defects which resulted in accumulating the defects from the step of black-box testing. It is the appropriate process for their work. In the result, they represent this concept as "Orthogonal Defect Classification for Black-box Defect (ODC-BD)" [9].

2.3 Software Defect Pattern

Software defect pattern collects of software defects from the real work with an aim to reduce repetitive defects. Defects are recorded and categorized by the cause of that error, the phase of

injection, the effect of that defects and how to remove it. Another important information from the pattern is the knowledge that can guide developers on how to prevent defects.

A study in 2009 investigated implementation of software defect pattern in the software process. The purpose of that research was to increase the reliability of software design [10].

This research implements this set of defect classification in the Knowledge of Software Defect (KSD). It could identify the defect information in the right stage of the software process. The detail of KSD in this experiment is reported in the research design section.

2.4 Personal Software Process

Personal Software Process (PSP) is a tool for investigating and improving personal performance in software development [11],[12]. PSP collects and shows the statistics that are calculated from the data that engineers record. These results can be used to analyze the strength and weakness of an individual. Thus, engineers can continually improve themselves.

PSP can be applied in various areas of software engineering. There is no limitation of computer language or software process model. It can be implemented in pair programming [13] and M-V-C frameworks [14]. Researchers reported that PSP can improve the personal performance of students in both solo and team programming styles.

Research in 2015 presented an experiment of MVC-PSP to increase the reliability of defect logging [15]. Two activities including Defect Detection Capability Test (DDCT) and Defect Standard Table (DST). DDCT is a test for calculating the engineers defect detection capability. DST is a review of the team to generate and update the standard of defect detection. Based on the results, it is concluded that the defect standard table has higher reliability. As a result, this research proposes that the defect standard table can be effective for defect logging.

3. RESEARCH DESIGNS

3.1 Participants

This research was implemented in 3 batches during 3 undergraduate courses on Mobile Application Development. There were 18, 21 and 38 students who studied in the department of software engineering, department of information technology, department of computer engineering and department of business computer. These students have different programming experiences.

In each batch of the experiment, students were organized into 2 groups based on the result of the Defect Detection Capability Test. The better performance group of the student was assigned to group A as the control group and the lower group was assigned to group B. The DDD's methods were implemented in only group B.

3.2 Duration

Each batch of experiments took 6 weeks to complete. It involved 6 programming exercises. Detail of the exercises is described in the following section. This research took 4 months to complete all experiments.

The experiments were designed to provide feedback to each other. The result of the first experiment had been used as input data for the second one and later. The result of the second experiment was also input data for the third experiment.

3.3 Exercises in the Experiment

There are 6 exercises in this experiment shown in Table 1 The structure of these exercises follows the official PSP training scheme. The first exercise is easy so that the participants adjust their working process to get used to the PSP framework. Only working process, time spent on each step, errors occurred in the working process are recorded. The second exercise develops an Android application which calculates geometric shapes. It introduces resource estimation in PSP framework.

The third to fifth exercise is related to general calculations with the addition of the decision process. Full PSP process, including reviewing of design and code, are included in these exercises. The last exercise is the only exercise of this research study that must be connected to a database.

4. METHODOLOGY

4.1 Workshop Iteration

Each batch of the workshop involved 6 exercises. Control Group students were instructed to build projects based on their normal procedures. At the same time, DDD Group who displayed less capability of error detection during the test implemented DDD's activities. These activities help students to detect defects that should occur in their projects according to their design prior to the step of coding. KSD has displayed the information about those defects, how to prevent or debug that error. Finally, they could choose to implement the project with this problem or re-designing procedure.

Table 1. Workshop Exercises

Ex.#	Android Application	PSP Level.	Data Collection	Difficulty Level/Expected Development Time (min.)	Skill Needed
1	Simple Calculator	0	1, 2	1/115	• Simple Calculation
2	Areas of Geometric Shapes Calculator	1.0	1, 2, 3,4, 5, 6	1/135	• Simple Calculation • Class and Object
3	Body mass index (BMI) Calculator	2.0	1, 2, 3, 4, 5, 6, 7, 8	2/175	• Complex Calculation • Class and Object • Defensive Programming
4	Mini Horoscope	2.1	1, 2, 3, 4, 5, 6, 7, 8	2/110	• Complex Calculation • Class and Object • Defensive Programming • Logic and Decision
5	Taekwondo point calculator	2.1	1, 2, 3, 4, 5, 6, 7, 8	2/130	• Complex Calculation • Class and Object • Defensive Programming • Logic and Decision
6	To Do Listing	2.1	1, 2, 3, 4, 5, 6, 7, 8	3/220	• Class and Object • Defensive Programming • Logic and Decision • Database Programming

Note: • 1: Time, 2: Defect, 3: Time Estimation, 4: Size Estimation, 5: Actual Time, 6: Actual Size, 7: Design Review, 8: Code Review

When students of DDD group found defects during their coding, they can search the KSD for debugging guidelines. They can continuously add new information of software defect solving to KSD too. Fig.1 displayed the experiment processes.

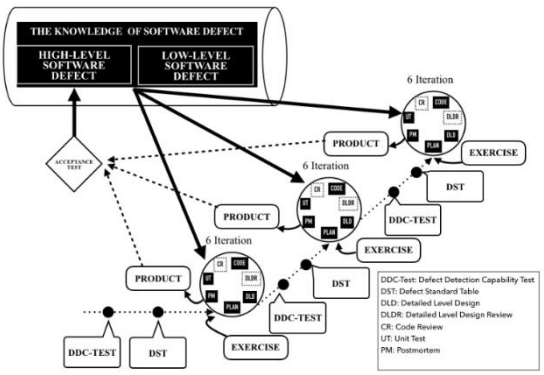


Fig. 1. The research processes.

5. RESEARCH RESULTS

After the students finished all 3 batches, their development data were compared. There are 4 indexes for investigating as follows: (i) Defect Density, (ii) Yield%, (iii) Time used and (iv) Productivity. All of those indexes can project the efficiency of DDD in this experiment. Other indexes also contribute supplement information to conclude this research.

The Defect density illustrates the intensity of defect in software building processes. It is calculated by the number of defects by the total line of code written in the project. The lower value of defect density represents the fewer defects in software. The average value of defect density of DDD group students is expected to be less than the Control group.

Yield% shows the capability of defect detection before the compile phase. The higher value of yield% means the developer detected more errors. That value of DDD group is expected to be greater than the Control group.

The DDD group may take more time to complete than those of Control group as they must complete more activities. Nevertheless, it is expected that the time spent by DDD group should not be significantly different than the control group as it would affect projects with limited time frames.

Lastly, productivity is the value showing the overall efficiency of the personal process, calculated by code size and the total time spent in completing all work. The DDD students are hypothetically expected to yield less productivity than the control group due to more activities.

5.1 The Defect Capability Test Result

Fig 2 shows the defect capability test result of all 3 batches. The defect capability test result projected the Java programming skill of all students. Students of the second batch scored 52.05 on average. It showed that they have the least skill in error detection in Java. The first batch' students had the highest skill with the score of 68.19 on average. This is not surprising since they were third-year software engineering students. The last batch had the most variety of programming skills because they

consist of students from 3 study program who had a different experience in computer programming. So, the standard deviation value of this group is the highest and their average score was 57.55.

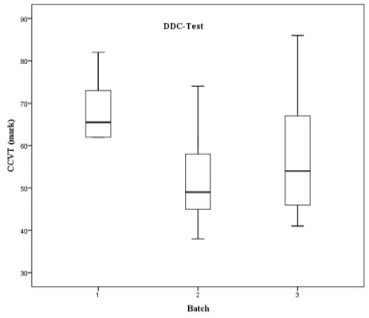


Fig. 2. The defect capability test result

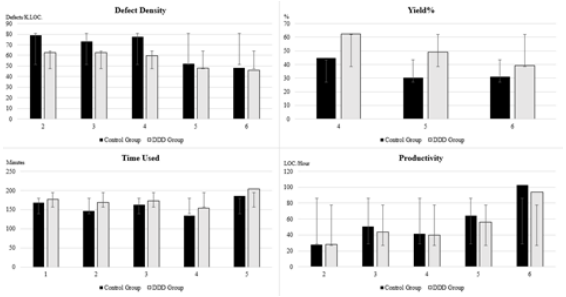
5.2 Exercise Result

The results from the study are displayed in Table 2. The software' size is not different because all batches used the same set of exercises. However, the time used is different. As mentioned above, the second batch' students had the least programming skill in Java, so they used the longest time to finish the exercises more than others.

The average value of defects amount is not different. But the sixth exercise was different because it is the biggest size of the code. The function of this project had to connect to the database engine with the Android platform. The student had to spend part of the time to manage the database structure. This exercise is not only the most time consuming but also led to the most defects.

5.3 All Batch's Result

The experiment result of all batches is shown in Fig 3. The average value of defect density of students in DDD group is significantly less than the Control group in every exercise. The average value of yield% of DDD group is significantly greater than the Control group in every project too. It is noted that the yield% value is calculated from the 4th exercise since the essential data was not previously collected because the earlier exercises used the lower level of PSP that appropriates with the easy projects.



Note: * Defect density can be calculated from the 2nd exercise.
* Productivity can be calculated from the 2nd exercise.
* Yield% can be calculated from the 4th exercise.

Fig. 3. The result of all batch's experiment

For the time used to complete workshops, the students of the DDD group spent the additional time for DDD's activities. So, they used more time than the Control group members with the same size of the program code. Then, the productivity of the Control group is higher than the DDD group in every exercise excluding the first one as expected.

5.4 The Result Comparison of Students in DDD's Group

Fig 4 shows the result that compares DDD group students with the other three batches. It illustrates the evolution of the DDD model and the KSD. This result shows that students who implemented the DDD model in software projects tended to reduce their defects. It is referenced from the downward

Table 2. Exercise result

Ex.	Average size (SD)						Average time used (SD)						Average defects (SD)					
	Batch 1		Batch 2		Batch 3		Batch 1		Batch 2		Batch 3		Batch 1		Batch 2		Batch 3	
	A	B	A	B	A	B	A	B	A	B	A	B	A	B	A	B	A	B
1	-	-	-	-	-	-	113.57 (5.26)	134.82 (5.95)	145.40 (2.33)	148.36 (3.11)	94.79 (1.32)	114.11 (2.73)	4.29 (1.39)	6.82 (1.53)	6.40 (1.28)	6.55 (1.62)	7.05 (1.76)	6.00 (1.38)
2	93.43 (1.40)	110.64 (7.04)	111.10 (1.37)	111.09 (1.88)	112.00 (1.65)	112.26 (1.65)	152.14 (2.36)	158.64 (3.31)	204.30 (1.55)	207.09 (1.56)	146.32 (1.69)	166.79 (2.40)	6.00 (1.07)	5.36 (1.82)	6.40 (1.36)	5.27 (1.66)	5.42 (1.31)	4.89 (1.02)
3	112.86 (4.58)	118.91 (3.53)	120.40 (1.28)	120.45 (1.88)	119.26 (0.91)	119.05 (0.89)	129.86 (1.64)	138.18 (3.95)	195.20 (1.17)	212.91 (1.88)	115.68 (3.01)	155.16 (4.26)	6.57 (1.05)	6.45 (1.62)	5.90 (1.97)	5.55 (2.39)	6.95 (2.76)	5.00 (1.59)
4	97.14 (2.70)	105.09 (4.21)	107.80 (2.96)	107.91 (3.26)	110.47 (1.43)	108.58 (1.31)	115.29 (4.37)	123.36 (4.60)	226.20 (1.25)	236.09 (1.16)	146.63 (2.18)	161.32 (3.37)	5.29 (1.48)	4.91 (1.44)	5.80 (2.40)	4.82 (0.83)	6.58 (1.43)	4.26 (1.02)
5	137.71 (1.67)	136.73 (1.35)	140.50 (0.81)	140.27 (1.21)	140.32 (1.30)	139.68 (1.08)	107.43 (2.06)	122.82 (6.45)	163.29 (1.45)	190.09 (5.37)	130.89 (2.27)	147.26 (1.94)	5.43 (1.29)	5.18 (1.19)	6.14 (1.42)	5.55 (1.29)	5.74 (1.41)	5.05 (1.05)
6	310.00 (6.57)	312.64 (4.07)	316.20 (1.72)	315.73 (2.38)	321.89 (2.34)	321.11 (1.74)	167.14 (3.83)	174.36 (3.47)	204.60 (6.20)	227.45 (2.78)	185.05 (2.86)	211.21 (4.16)	12.00 (3.30)	12.73 (2.53)	16.00 (1.41)	15.00 (0.74)	13.37 (3.01)	11.74 (2.51)

trend of defect density value. While the yield% tenor is increased similarly with the value of time used.

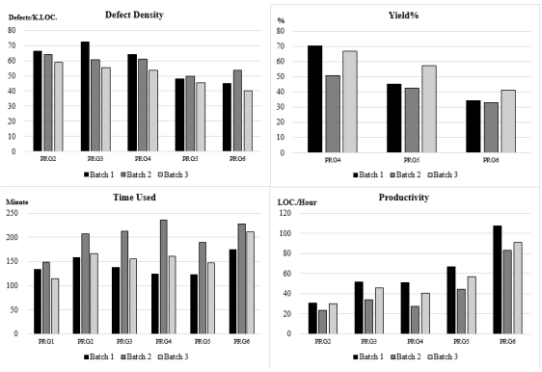


Fig.4. The result comparison of students in DDD's group

5.5 The Statistics Significant Test by ANOVA

A one-way between groups ANOVA is conducted to compare the groups for the value of defect density. There is a significant different at the $p<.05$ level [$F(1, 28) = 4.883$, $p = 0.035$] that shows in Table 3.

Table 4 displays the result of ANOVA to comparing the group for the value of yield%. It shows that is a significant different at the $p<.05$ level [$F(1, 16) = 5.199$, $p = 0.037$].

6. CONCLUSION

Defect-Driven Development (DDD) is a novel concept of the software development process which utilizes the benefit of the Knowledge of Software Defect (KSD), which collects defect data from experienced practitioners, to proactively mitigate defects in new software development. Novice developers can learn from the expert knowledge and thus effectively prevent defects, especially in the early phase of software development.

The result of the experiment shows that students who implemented the DDD concept injected significantly fewer defects than others who applied the general personal software processes. It can be measured by two main indicators including, I) defect density and II) yield%. However, the data suggest that implementing DDD may result in longer development time which is spent on extra preventive activities. This causes the productivity of DDD subjects to be slightly lower than traditional implementation.

7. REFERENCES

[1] P. Letouze, J. I. M. de Souza, and V. M. Da Silva, "Generating Software Engineers by Developing Web Systems: A Project-Based Learning Case Study," 2016 IEEE 29th Int. Conf. Softw. Eng. Educ. Train., pp. 194–203, 2016. http://doi.org/10.1109/CSEET.2016.11

[2] R. Chillarege et al., "Orthogonal Defect Classification-A Concept for In-Process Measurements," IEEE Trans. Softw. Eng., vol. 18,no.11,1992.http://doi.org/10.1109/32.177364

[3] K. Beck, Test Driven Development: By Example, 1st ed. Addison-Wesley Professional, 2002.

[4] K. Beck, Extreme Programming Explained: Embrace Change. Addison-Wesley Professional, 2004.

[5] L. Williams, E. M. Maximilien, and M. Vouk, "Test-driven development as a defect-

Table 3. The result of ANOVA analysis for Defect Density

Source of Variation	Sum of Squares	Degree of freedom	Mean Square	F	Sig.
Between Groups	766.538	1.000	766.538	4.883	0.035
Within Groups	4,395.079	28.000	156.967		
Total	5,161.617	29.000			

*note: Sig. = Significance

Table 4. The result of ANOVA analysis for Yield%

Source of Variation	Sum of Squares	Degree of freedom	Mean Square	F	Sig.
Between Groups	991.643	1.000	991.643	5.199	0.037
Within Groups	3,051.804	16.000	190.738		
Total	4,043.447	17.000			

*Note: Sig. = Significance

reduction practice," 14th Int. Symp. Softw. Reliab. Eng. 2003. ISSRE 2003., pp. 1–12, 2003. http://doi.org/10.1109/ISSRE.2003.1251029

[6] L. Crispin, "Driving software quality: How test-driven development impacts software quality," IEEE Softw., vol. 23, no. 6, pp. 70–71, 2006. http://doi.org/10.1109/MS.2006.157

[7] J. W. Wilkerson, J. F. Nunamaker, and R. Mercer, "Comparing the defect reduction benefits of code inspection and test-driven development," IEEE Trans. Softw. Eng., vol. 38, no. 3, pp. 547–560, 2012. http://doi.org/10.1109/TSE.2011.46

[8] M. M. Müller and O. Hagner, "Experiment about test-first programming," IEE Proc. - Softw., vol. 149, no. 5, p. 131, 2002. http://doi.org/10.1049/ip-sen:20020540

[9] N. Li, Z. Li, and X. Sun, "Classification of software defect detected by black-box testing: An empirical study," Proc. - 2010 2nd WRI World Congr. Softw. Eng. WCSE 2010, vol. 2, pp. 234–240, 2010. http://doi.org/10.1109/WCSE.2010.28

[10] F. Zeng, A. Chen, and X. Tao, "Study on software reliability design criteria based on defect patterns," Reliab. Maintainab. Safety, 2009. ICRMS 2009. 8th Int. Conf., pp. 723–727,2009.http://doi.org/10.1109/ICRMS.2009.5270095

[11] W. S. Humphrey, PSP(sm): A Self-Improvement Process for Software Engineers, 1st ed. Addison-Wesley Professional, 2005.

[12] W. S. Humphrey, "The personal process in software engineering," in Proceedings of the Third International Conference on the Software Process. Applying the Software Process, 1994, no. c, pp. 69–77. http://doi.org/10.1109/SPCON.1994.344422

[13] G. Rong, H. Zhang, M. Xie, and D. Shao, "Improving PSP education by pairing: An empirical study," Proc. - Int. Conf. Softw. Eng., pp. 1245–1254, 2012. http://doi.org/10.1109/ICSE.2012.6227018

[14] W. Nachiengmai and S. Ramingwong, "Implementing Personal Software Process in Undergraduate Course to Improve Model-View-Controller Software Construction," in Lecture Notes in Electrical Engineering, vol. 339, 2015, pp. 949–956. http://doi.org/10.1007/978-3-662-46578-3_113

[15] W. Nachiengmai and S. Ramingwong, "Improving Reliability of Defects Logging in MVC-PSP," in 2015 2nd International Conference on Information Science and Security (ICISS), 2015, pp. 1–4. http://doi.org/10.1109/ICISSEC.2015.7371007

[16] S. Thisuk and S. Ramingwong, "WBPS: A new web-based tool for Personal Software Process," in 2014 11th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON), 2014, pp. 1–6. http://doi.org/10.1109/ECTICon.2014.6839821

[17] C. Larman and V. R. Basili, "Iterative and incremental developments. a brief history," Computer (Long. Beach. Calif)., vol. 36, no. 6, pp. 47–56, Jun. 2003. http://doi.org/10.1109/MC.2003.1204375

[18] D. Liu, S. Xu, and W. Du, "Case study on incremental software development," Proc. - 2011 9th Int. Conf. Softw. Eng. Res. Manag. Appl. SERA 2011, pp. 227–234, 2011. http://doi.org/ 10.1109/SERA.2011.43

[19] K. Schwaber and M. Beedle, Agile Software Development with Scrum, 1st ed. Pearson, 2001.

[20] Y. Zhang and S. Patel, "Agile model-driven development in practice," IEEE Softw., vol. 28, no. 2, pp. 84–91, 2011. http://doi.org/10.1109/MS.2010.85