World Academy of Science, Engineering and Technology
International Journal of Computer and Systems Engineering
Vol:16, No:5, 2022

A Study of Agile-Based Approaches to Improve Software Quality

Gurmeet Kaur, Jyoti Pruthi

***Abstract*—**Agile Software development approaches and techniques are being considered as efficient, effective, and popular methods to the development of software. Agile software developments are useful for developing high-quality software that completes client requirements with zero defects, and in short delivery period. In agile software development methodology, quality is related to coding, which means quality, is managed through the use of approaches like refactoring, pair programming, test-driven development, behavior-driven development, acceptance test-driven development, and demand-driven development. The quality of software is measured using metrics like the number of defects during the development and improvement of the software. Usage of the above-mentioned methods or approaches reduces the possibilities of defects in developed software, and hence improves quality. This paper focuses on the study of agile-based quality methods or approaches for software development that ensures improved quality of software as well as reduced cost, and customer satisfaction.

***Keywords*—** Agile software development, ASD, Acceptance test-driven development, ATDD, Behavior-driven development, BDD, Demand-driven development. DDD, Test-driven development, TDD.

I. Introduction

ALONG with the exposure of dynamic societies or evolution in the software industry, the development of software projects is facing dramatic changes. As software requirements are not clear and prediction to complete a project within a limited expense and limited period is not possible using traditional software development strategies, agile-based software development strategies that work on the basis of finding a reasonable solution to such problems have been employed since 2000 [1], [2]. Agile approaches are featured by the easy and quick techniques to modify the product as per a user requirement which helps to customer satisfaction and delivery of product in time. There are many agile approaches and methodologies that are approved, and used in the software engineering discipline. Adaptive Software Development, Crystal family, FDD, XP, and SCRUM are the most well-known methodologies. Generally, agile software development (ASD) encourages a management system that supports teamwork, frequent review, modification, framing the best methods, and allowing for fast delivery of high-quality product [3]. Refactoring and testing are critical events in ASD. In these techniques, there must be continuous modification to the members of a project: designers, and developers, etc. The progress of software projects lies in the satisfaction of functionalities as well as quality characteristic, such as reliability, adaptability, and performance. Besides, there are various corresponding points between the serviceability, and quality aspects [4]-[6]. Hence, also quality aspects must be examined and the design of software development must be assembled in such manners that there are connections between the quality, and performance aspects. Of course, when a software project fails or modifies, the reason is not its incomplete performance but its faulty quality aspects such as its low performance, applicability, and improvement [7]. This paper focuses on agile methodology-based quality approaches for high quality of software development.

Gurmeet Kaur, Research Scholar, is with the Manav Rachna University Faridabad, Haryana, India (e-mail: grmtkaur02@gmail.com).

Dr. JyotiPruthi, Associate Professor, is with the Manav Rachna University Faridabad, Haryana, India.

II. Related Study

The main concern in the software development industry is managing the defect. The primary objective of defect management is the satisfaction of the customer. The delivery of a high-quality software product leads to customer satisfaction [8].The distinct characteristics of a high-quality software product are that, they are less defective, and produce predicable results and delivery in time and cost. Software inspection, review, and testing are some of the most common strategies to detect defects before the release of a software product in the market in the traditional software development life cycle [9], [10]. But ASD guarantees higher project satisfaction at lower cost with efficient resource usage. In agile techniques, the development of software takes place in iterations repeatedly to progressively define steps. In a software development process, there are phases such as planning, design, coding, testing, and customer feedback, and there are various errors that identify in these phases. As ASD does not concentrate externally on formal review and inspection techniques, there is a need to add implicit techniques for defect detection, and quality enhancement procedures in the development life cycle. ASD uses a non-formal method of quality management. It is the responsibility of team members to assure quality preservation, as well as the implementation of best quality approaches [11]. ASD strategies are adaptable means, and remain open to change requirements over time. Organizations that prefer ASD methods are associates with standard tools and techniques [12]. The analysis performed for the development of software includes identification of prime parameters with their levels for ideal defects acquiring that will help engineers and managers to make better decisions for improvement of the quality of products. The software quality process focuses on controlling product quality and aims to produce non-defective or adequate products. In real situations, defects can be originated at every stage of the software process [13], for example the defects could be originated by stakeholders, the product owner, or the software development team from 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 five steps as follows: requirement analysis, design, construction, test, and delivery, and maintenance. Research in [13] reported that defects can occur in every phase of the software process. A software defects could be avoided with increasing the experience of software developers or by using the different defect reducing techniques as discussed in the next section.

III. Agile-Based Quality Approaches or Methods

ASD techniques and methods include several approaches that assure quality. This section describes various approaches which are practical and employed to perform quality assurance based on agile.

A. On-site Customers

This is a general method of quality assurance in which the customer helps the developers to filter and correct the requirements. Therefore, customer involvement is much stronger in ASD as compared to traditional development. In ASD, the involvement of the customer lies in every phase of development such as planning, designing, coding, and testing, whereas in traditional development methods their involvement in the definition, system and software design may be limited. In traditional development methods, customer input is generally limited to inspection and review stages as outlined in the development plan, and therefore their involvement is less intensive than in ASD techniques.

B. System Metaphor

A metaphor gives a description of the working system using examples on analogies. It can help overcome any issues among designers and customers by providing a model and guide to the dialogue between the relevant parties. As well, it includes classes and patterns that are useful for coding. It is used for the common system of names of classes, patterns, functions, and methods, etc.; thus, every member of a team can understand the working of code and the right way to modify the functionality of the system. In this way, system metaphor aids the team in the evaluation of the design [14].

C. Pair Programming

In pair programming, work is done by two programmers, who have to sit next to each other on the same workstation. One programmer writes the code and other reviews it. The programmers can interchange their jobs after a certain period. This means that every member of team is fully aware of the processes of the system at each interval of time. This method of programming helps reduce defects and improve quality. Research shows that pair programming is more effective concerning quality [15]. This shoulder-to-shoulder technique aids in the process of the continual design and review of code and results in a reduction in defect rates. As well, this process has been widely considered as continuous code inspection [15].

D.Refactoring

This approach is termed as continuous design improvement. It is a process in which duplicate code is removed to improve design. In refactoring, the code is restructured changing its internal configuration without changing its external functionality. The action of restructure of code delivers code inspection functionality and increases the probability of detecting errors during development. The process is composed of a set of small conversions. Each conversion (called a 'refactoring') makes a small change, but a sequence of conversions can lead to considerable restructuring. Refactoring is useful for improving the design of existing code as well as reducing the chance of error [16].

E. Continuous Integration

This approach involves merging code developed by all the developers across the company into one single common place. Integration takes place many times and helps to detect many errors and defects. It saves time in development through detection of errors and helps to expose compatibility problems early. In traditional development modeling, integration is done at the final stage, and the change for error detection is lower but in agile-based modeling, integration is performed numerous times, so there is more chance to detect errors at an early point. Continuous integration is a dynamic technique of quality assurance [17].

F. Test Driven Development (TDD)

TDD is the main concern of the agile manifesto and extreme programming. Many positive outcomes have been reviewed by TDD. TDD is not for testing; instead, it is a structure and improvement strategy in which tests are composed before the creation code. This technique progressively works in small cycles of adding and implementing a test case. Although the test case passes, the code is refactored to improve the internal structure of the code. This process is repeated until whole functionality is executed [18].

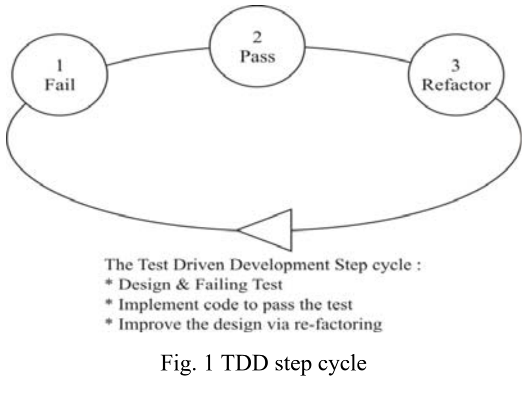
1
Fail
2
Pass
3
Refactor
The Test Driven Development Step cycle :
* Design & Failing Test
* Implement code to pass the test
* Improve the design via re-factoring

Fig. 1 TDD step cycle

Use of TDD in industry is revealed by its constant high ranking from members of the development's teams [19]. Research in [20] described their work performed on TDD for a period five and half month under contemporary industry conditions, and presented a successful application of TDD. The support of the designer to the TDD approach aids the easy maintenance of code, reduction in residual defects and rapid delivery of a product with reference to productivity. The requirement analysis and initial coordination is more time-consuming in TDD, but the functional testing takes less time due to increased unit testing [20]. Research on utilization of agile methodologies describes that over 80% software professionals utilize Scrum practices whereas 18% of professionals utilize Extreme programming. However, there is industrialists' enthusiasm in the investigation of TDD and its impact on productivity, process, cost reduction and product quality [21]. An investigation at IBM between 2001 and 2006 revealed that TDD takes more time at the initial phase, however it is compensated with more robust high-quality code [22].

The procedure of the TDD cycle consists of six basic steps:

1. Structure a test for a user story or piece of code;
2. Execute test and produce a failed test;
3. Write code for functionality that passes the tests;
4. Execute the test to confirm and the code passes;
5. Refactor the code;
6. Execute a set of tests to check that refactoring does not make changes to the external functionality.

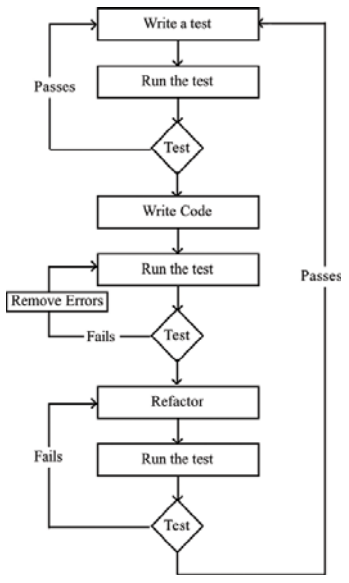

Fig. 2 Flow chart for TDD

Advantages and Disadvantages of TDD

The benefits of TDD:

i. Prevents defects.
ii. Allows code documentation with executable examples.
iii. Helps programmers really to understand their code.
iv. Supports refactoring as needs, and design changes.
v. Encourages better design (more cohesive modules that are loosely coupled).
vi. Creates basically a free automated regression test.

TDD promotes small steps and a simple working development system. It supports the programmer's skills in designing different kinds of tests. It presents advance warning to detect design problems early in the design process. The disadvantages of TDD:

i. A challenge to learn.
ii. Hard to apply to legacy code.
iii. Lots of misconceptions that keep programmers from learning it.

G.Behavior-Driven Development (BDD)

BDD based on TDD is a methodology that evolved into a process that does not concern only programmers and testers, but also deals with the entire team and all-important stakeholders, both technical and non-technical. Business stakeholders and domain experts can often determine engineers for high-level tests that would be useful to deal with important business aspects. BDD reserves the word, "test" for low-level technical checks such as data validation. The important aspect of BDD is that while tests can only be created by developers and testers, they can be collected and analyzed by designers and analysts.

BDD is a synthesis and refinement of software engineering application that supports teams to generate and distribute higher quality software quickly. The BDD process is similar to TDD and follows these steps:

1. Write a scenario;
2. Run the scenario that fails;
3. Write the test that corresponds to the specifications of the scenario;
4. Write the simplest code to pass the test and the scenario, and lastly;
5. Refractor to eliminate duplication.

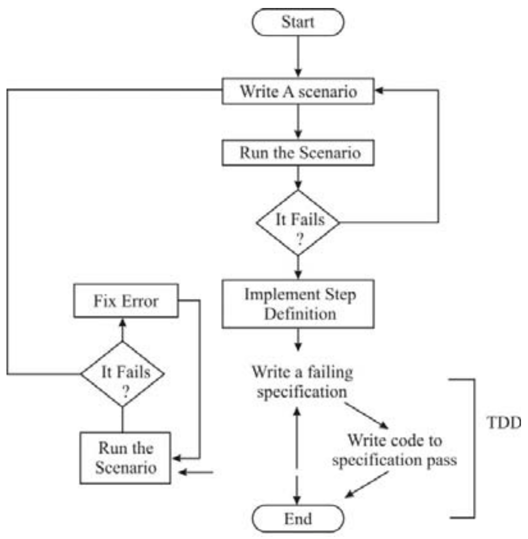

Fig. 3 Flowchart for BDD

Advantages and Disadvantages of BDD

If a software team plans to implement BDD, there are a few points to consider that will benefit them.

i. The software team is no longer defining the 'test' instead they are defining 'behavior'.

ii. It generates a best exchange of information between product owners/stakeholders, tester and developers.
iii. It covers a wider audience since it is non-technical by nature.
iv. The behavioral approach specifies acceptance guidelines and rules prior to the software development.

Even the best development approaches can have problems and BDD is no exception. Some of them are:

i. Prior experience of TDD is required to work with BDD.
ii. BDD is incompatible with the waterfall approach.
iii. If the requirements are not properly specified, BDD may not be effective.
iv. Testers using BDD need to have sufficient technical skills.

H.Acceptance Test-Driven Development (ATDD)

It is an approach that represents different methods for solving software engineering challenges [23].

1. ATDD provides instruction to update every part of a complete software development phase.
2. ATDD offers a shared medium of communication to enhance exchange of information between product owner and stakeholders.
3. ATDD ensures that the project under development continuously satisfies its requirement by using testing and refactoring [24].

The basic reasons for lack of the success of many software projects is the delayed identification of mismatch between functionality executed in the delivered system and the customer requirements. The cause of the mismatch is a set of under specifying, poorly defined, and inconsistent requirements. The technologies that have been used to upgrade the quality and usability of the software project in a traditional development life cycle are mostly proposed and are used in the testing phase (67.48%), while few methods are used by the developers in the early stages (15.98% design phase and 13.70% analysis phase) [25]. However, interest in researching ASD has risen in recent years [26] and adds approaches of ASD such as ATDD. ATDD focuses on the use of direct ATDD from the initial phase of the development [27]. ATDD is a development technique based on communication between the developers, the tester and the business customers. It is useful to encourage reusability in the software enhancement phases and to satisfy customer requirements. As well, ATDD encompasses acceptance testing by writing acceptance tests before coding.

The ATDD process follows the following steps:

1. Select user story;
2. Write acceptance test;
3. Implement user story;
4. Run acceptance test; and,
5. Make change/Refactor.

The advantages of ATDD include:

i. Improved communication and collaboration between project stakeholders.
ii. Shared understanding of what successful implementation means.
iii. Better coverage of business expectations.
iv. Faster feedback.

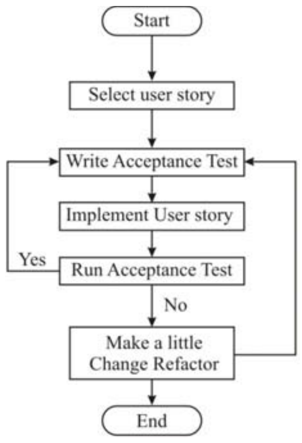

Fig. 4 Flowchart for ATDD

The disadvantages of ATDD are few and include:

i. New methodology that requires rigor and discipline.
ii. Find the right balance between people/process/tool.

I. Defect-Driven Development (DDD)

The concept of DDD uses the knowledge of software defects to proactively drive the software process. The knowledge base of software defects is collected from every step of the software processed by experienced software developers. Then, it is normalized to a standard format and the software defect pattern is rearranged for novice developers. The main principle of DDD focuses on proactive activities to check the design and types of errors that it might lead to, as well as how to avoid them before coding. This is done by referencing software defect knowledge that was previously collected from experienced software developers. A software developer 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 TDD hoping that developers foresee the potential problems before the coding stage. The difference is that TDD involves a design of unit tests before coding which may not be a natural process for a novice; while, DDD more conveniently adds a defect checklist during the design process. 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.

Software Defect Taxonomy

Controlling defects is one of the most important aspects of software quality management. There are many researchers that have studied the nature of the software defects, particularly defect classification. The research in [28] presents a framework for classifying software defects using cause-effect analysis. This is done by collecting feedback on defects from the software developers, which include the phase of defects injection, the cause of the defect, and the effect of those defects. The results of this study identify seven classes of defect including, Function, Interface, Checking, Assignment, Timing/Serialization, Documentation, and Algorithm [28]. These classes are distributed in every stage of a software process, and are termed as "Orthogonal Defect Classification (ODC)". ODC can be used in many studies of software engineering areas, i.e., to classify software defects in a specific phase of the software development process, and the prediction of defects [28]. The research by [29] is an example of a study of the implementation of ODC, 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 results in the accumulation of defects from the step of black-box testing. Finally, this concept is described as "Orthogonal Defect Classification for Black-box Defect (ODC-BD)" in [29].

Software Defect Pattern

Software defect pattern is the collection of software defects with an aim to reduce repetitive defects. Defects are recorded and categorized by the cause of the error, the phase of injection, the effect of the defect, and how to remove it. Important information from the pattern is the knowledge that can guide developers on how to prevent defects. A study examined the implementation of software defect pattern design in the software development process. The purpose of that research was to increase the reliability of software design [30]. This research implements the set of defect classification in the Knowledge of Software Defect (KSD), and could identify the defect information at the right stage of the software process.

Personal Software Process

Personal Software Process (PSP) is a tool for investigating, and improving personal performance in software development [31], [32]. PSP collects, and shows the statistics that are calculated from the data of the engineer's records. These results can be used to analyze the strengths, and weaknesses of an individual; thus, engineers can continually improve themselves. PSP can be applied in various areas of software engineering, since there are no limitations with regard to the software process model or computer language types. It can be implemented in pair programming [33], and M-V-C frameworks [34]. Research [31] represents that PSP can improve the personal performance of engineers in team, and solo programming styles.

Research described an experiment of MVC-PSP to increase the reliability of defect logging that includes two activities: Defect Standard Table (DST) and Defect Detection Capability Test (DDCT) [35]. DDCT is a test for calculating the engineers defect detection capability while DST is an analysis of the development team to produce and update the standard of defect detection. Based on the results, it was concluded that the defect standard table has higher reliability. As a result, this research proposes that the defect standard table can be used effectively for defect logging.

IV. Conclusion

Due to changes in software industry trends, organizations have had to rapidly update their approaches for quality to develop high-quality software products that are in demand by consumers. This paper describes many agile-based approaches to improve software quality as well as customer satisfaction. The main points of concern are as follows; refactoring provides code inspection functionality during reconstruction of code and reduces the generation of errors. The TDD approach upgrades the quality of software and enhances client satisfaction by permitting thorough unit testing before coding. BDD evolved from TDD to eliminate the shortfalls of TDD, since it does not work on behavior. As well, BDD describes the method of developing a feature based on behavior in a simple language like English, which can be understood by all members of the development team. ATDD makes the implementation process more effective by writing code using requirements, reducing developer efforts and continuously testing the product until it meets the customer's expectations. The DDD approach utilizes the benefit of knowledge of software defect (KSD), which gathers defect data from experienced professionals to proactively eliminate deficiency in novice developers. In this way, the novice developer can learn from expert knowledge and effectively prevent defects, especially at an early stage of the software development. Finally, this paper aimed to describe the feature of various approaches for improving software quality with their usefulness.

References

[1] P. Abrahamsson, J. Warsta, M. T. Siponen and J. Ronkainen," New Directions on Agile Methods: A Comparative Analysis", 25th ICSE, 2003.
[2] A. Cockburn, "Agile Software Development", 1st Edition, Addison-Wesley Professional, 2001
[3] J. Eckstein, "Agile Software Development in the Large: Diving Into the Deep". New York: Dorset House, 2004.
[4] M.R. Barbacci, R. Ellison, A.J. Lattanze, J.A. Stafford, C.B. Weinstock and W.G. Wood, "Quality Attribute Workshops (QAWs)", Technical Report, 3rd Edition, SEI, CMU, 2003.
[5] R. Wojcik, F. Bachmann, L. Bass, P. Clements, P. Merson, R. Nord and B. Wood, "Attribute-Driven Design (ADD)", Technical Report, 2nd Edition, SEI, CMU, 2006.
[6] R. Nord, M. Barbacci, P. Clements, R. Kazman, L. OBrien and J. Tomayko, "Integrating the Architecture Tradeoff Analysis Method(ATAM) with the Cost Benefit Analysis Method (CBAM)", Technical Report, 1st Edition, SEI, CMU, 2003.
[7] L. Bass, P. Clements, and R. Kazman, "Software Architecture in Practice", 2nd Edition, Addison-Wesley, 2003.
[8] GalinD" Software quality: concepts and practice". Wiley, NJ,2017
[9] Tian J "Software quality engineering testing, quality assurance, and quantifiable improvement". Wiley, New Jersey,2005
[10] Kandt KR "Software engineering quality practices". Auerbach Publications, Philadelphia,2006
[11] Dingsøyr T, Dyba ˚ T, Moe NB (Eds) (2010) "Agile software development: current research and future directions". Springer, Berlin
[12] Sommerville I "Software engineering", 10th edn. Pearson, India,2017
[13] 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/10.1109/32.1773 64
[14] "How Userful Is the Metaphor Component of Agile Methods"? A Preliminary Study,
[15] A. Cockburn and L. Williams, "The Costs and Benefits of Pair Programming," in Extreme Programming examined, G. Succi and M. Marchesi, Eds. Boston: Addison-Wesley, 2001, pp. xv, 569 p.

[16] M. Fowler, "Information about Refactoring," 2004.
[17] “Continuous Integration”, http://www.martinfowler.com/articles/continuousIntegratio n.html.
[18] Shrivastava and Jain, "Metrics for Test Case Design in Test Driven Development", International Journal of Computer Theory and Engineering, Vol.2, No.6, December, 2010, Pg: 1793-8201
[19] Emam K “Finding success in small software projects, agile project management executive report. Technical report”, Cutter Consortium, Arlington, Massachusetts, 2003.
[20] Latorre R “A successful application of a test-driven development strategy in the industrial environment”. EMPSoftwEng 19:753–773, 2014.
[21] Rodriguez P, Markkula J, Oivo M, TurulaK “Survey on agile and lean usage in Finnish software industry”. In: six international symposiums on empirical software engineering and measurement, 2012.
[22] Sanchez JC, Williams L, Maximilien EM “On the sustained use of a test-driven development practice at IBM”. In: AGILE conference, pp 5–14, 2007.
[23] N. Koudelia, “Acceptance test-driven development”. 2011.
[24] L. Koskela, “Test Driven: TDD and Acceptance TDD for Java Developers”, Edición: 1. Greenwich, CT: Manning Publications, 2007.
[25] W. Silva, N. M. Costa Valentim, and T. Conte, “Integrating the Usability into the Software Development Process,” in Proceedings of the 17th International Conference on Enterprise Information Systems - Volume 3, Portugal, 2015, pp. 105–113.
[26] K. Curcio, R. Santana, S. Reinehr, and A. Malucelli, “Usability in agile software development: A tertiary study,” Computer Standards & Interfaces, Jan. 2019.
[27] M. Leotta et al., “An acceptance testing approach for Internet of Things systems,” IET Software, vol. 12, no. 5, pp. 430–436, 2018.
[28] 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/10.1109/32.1773 64
[29] 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
[30] 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
[31] W. S. Humphrey, PSP(sm): “A Self Improvement Process for Software Engineers”, 1st ed. Addison-Wesley Professional, 2005.
[32] 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
[33] 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.
[34] 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
[35] 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.