Scientific Research Publishing

Journal of Service Science and Management, 2018, 11, 13-25
http://www.scirp.org/journal/jssm
ISSN Online: 1940-9907
ISSN Print: 1940-9893

Ethics as a Quality Driver in Agile Software Projects

Hisham Abdulhalim[1], Yotam Lurie[1], Shlomo Mark[2]

[1]Department of Management, Ben-Gurion University of the Negev, Be'er Sheva, Israel
[2]Department of Software Engineering, SCE-Shamoon College of Engineering, Ashdod, Israel
Email: hishama@post.bgu.ac.il, yotam@som.bgu.ac.il, marks@sce.ac.il

How to cite this paper: Abdulhalim, H., Lurie, Y. and Mark, S. (2018) Ethics as a Quality Driver in Agile Software Projects. *Journal of Service Science and Management*, **11**, 13-25.
https://doi.org/10.4236/jssm.2018.111002

Received: November 8, 2017
Accepted: January 9, 2018
Published: January 12, 2018

Abstract

Comparing two software development teams working on similar projects in a large software company, this research study focuses on the question as to whether the implementation of ethical tools in the software development process serves as a quality driver. Is ethics a quality driver in Agile development processes? The findings of the present study indicate that there is a significant correlation between the inclusion of ethical tools in the process of planning in Agile methodologies and the achievement of improved performance in three quality parameters: schedule, product functionality and cost. Theoretically, the connection between ethics and quality is important. Practically, this study's findings show that the inclusion of ethical tools in Agile software projects can improve the quality of a project.

Keywords

Ethics, Software Engineering, Quality, Agile Methodologies, Software Quality, Innovation, Engineering Ethics, Ethical Framework

1. Introduction

This research study focuses on the question as to whether ethical tools act as a quality driver in Agile software projects. By comparing two development teams each having similar initial and ongoing development conditions, we show how ethical tools serve to drive software quality (*i.e.*, QVD or quality value drivers). We present a detailed experimental model showing that an ethical framework supports the production process in its early stages. We then show the positive impact on three quality parameters: time to market, functional defects and total cost of ownership. For educational theorists and researchers interested in portfolios, it is hoped that the study will also contribute to a better understanding of

the importance of software engineering as a regulated profession.

Modern civilization runs on software. Given this reality, the quality of software is a significant matter, since it is so widely used and is very important [1]. Over the years, developers and researchers of many developmental methodologies have allocated a large amount of time and effort to the refinement of those methodologies for better usage and for continuous improvement. Most of the methodologies have reached a mature and stable level and are referred to as traditional software development methods, e.g., the waterfall model [2] [3] [4]. Agile [5] [6] [7] [8], a modern software development methodology, became a controversial software engineering topic, when studies were made as to whether some of the supposed and promised strengths of Agile do actually improve software quality [8]. Some studies have reported that Agile methodologies do improve product quality, some studies have been inconclusive, and some studies have argued that Agile methodologies do not improve product quality. Dyba and Dingsoyr [9] [10] show that the evidence for the benefits or limitations of Agile methods is not very strong with regard to design, quality, consistency and directness.

A number of definitions have been proposed for a definition of software quality based on various parameters. In this paper, the quality of software is defined as comprising three parameters: schedule, product functionality and cost. The term "value driver" refers to a factor that leverages and achieves a higher level of value by means of a practiced technique [11]. Quality value drivers (QVD) are a managerial practice, reflecting a process approach rather than a product approach and promising quality value.

The professional literature shows that software development practices do not formally leverage ethical tools prior to, during or following developmental phases; thus, the study of professional ethics in software engineering is still largely unexplored territory. In 2015, Y. Lurie and S. Mark introduced an ethical framework called EDSD, derived from the concept of a framework in software engineering and textured with the principles of professional ethics.

This study aims to investigate if and how ethical tools drive to better quality in Agile software projects without conflicting with or contradicting the Agile standards defined in the Agile manifesto.

2. Literature Review

2.1. Quality Management

There are two basic approaches to quality: the product-based approach and the process-based approach. The general assumption in the product-based approach is that software quality requires that the testing of a product in terms of its requirements and expresses the basic desire that we want things to work as expected. Thus, quality engineers usually review a product's requirements, develop tests and examine the test records to check whether the testing has been properly carried out and whether the actual results are congruent with the expected results.

The subjective quality of a software product is largely based on its non-functional attributes [12]. Juran, who is widely considered to be a quality guru, defines quality as fitness for a given purpose, accompanied by customer satisfaction [13]. Pressman defines software quality in terms of specific measures, claiming that software quality is measured by the product's level of compliance with functional and performance requirements and development standards; in his definition, the focus is more on the engineering process [14].

Quality testing sets up measurement benchmarks to evaluate quality processes and identifies weaknesses in those processes so that quality issues will not repeat themselves. It requires the analysis of the tools, metrics, and processes used by management (including specification processes); analysis of the product's development; and testing activities. The classic functional testing and QA approach of verifying the functionality, load and performance of developed features in the post-delivery phase conflicts with the fast-shipment orientation that Agile relies on and can cause delays and uncertainty prior to a product's release. Hence, a common practice is to plan testing activities during development and to allocate the capacity of developers for the correction of future defects [15] [16]. It is clear that software quality is a vague concept-and one that is difficult to define. That is why there are multiple approaches for the measurement of the quality of software products [17].

2.2. Quality Value Drivers (QVD)

In the business world, managers seek various kinds of business value drivers, such as financial value drivers. The term "value driver" refers to a factor that leverages and achieves a higher level of value for a practiced methodology [16].

Quality value drivers target the improvement of the quality parameter. This parameter is controlled by a software development team, a project leader, or an organization. Petersen, Andersen, Heilesen, Klim, and Schmidt conclude that quality indicators are parameters that may be monitored and tracked during the development cycle in order to keep a project on the right track, namely the one leading toward a planned goal [18]. These indicators focus only on product and process attributes. The present research study focuses on three quality indicators: time to market, functional defects and total cost of ownership. Ethical tools are the drivers being investigated here as instruments for the attainment of higher scores for these three defined quality parameters.

2.3. Ethical-Driven Software Development (EDSD)

Over the years, software ethics evolved in different ways and formats [19] [20] [21] [22]. Ethical-Driven Software Development (EDSD) promotes an ethical framework that serves software engineers as a guideline for the entire development process [23]. The incentive behind Lurie and Mark's EDSD framework is the promotion of a proactive approach aimed at ensuring the quality of the final product by raising the awareness of ethical tools prior to, and during, the prod-

uct's development cycle. The EDSD framework formats YES/NO questions addressing each of the phases in the development process and demands that all the relevant stakeholders, including team members, users and sponsors be aware of the ethical issues involved in the product's manufacture, and that they have aligned expectations, prior to each phase. Thus, the EDSD framework can increase the commitment of developers to produce quality products and can reduce the potential for tension developing between the various stakeholders [23]. In practice, to create the necessary ethical framework, EDSD has index cards as a tool to help to facilitate the development process while sticking to the framework's questions. Each phase in this process has its own unique set of cards and each card contains only one ethical question that is specific and focused for that particular phase and which all the relevant stakeholders are supposed to answer. EDSD's testing and verification section asks five fundamental ethics-related questions:

1) Has the level of allowed errors been determined before there is a transfer of the product to acceptance testing been set?

2) Is a mechanism in place that distinguishes between a quality product versus a correct product?

3) Is a mechanism in place that determines whether the required tests were performed?

4) Is there a set of mandatory tests?

5) Are the metrics for determining the level of required tester training defined?

2.4. Agile Software Development

Appearing on the market in 2001, Agile offers a very contemporary approach in its principles and guidelines for software development [6]. Agile leverages more than one format: for example, it is applied to Scrum, one of the well-known methodologies being used today in the software industry. One of the main fundamentals in Scrum is that the product is developed in a series of fixed length iterations called sprints, ranging between one and four weeks and thus allowing the team or scrum a platform for delivering software in a closely orchestrated rhythm. A Scrum team comprises three to nine members. For software projects, a typical team will include at least one software engineer, at least one software architect and at least one quality assurance engineer [6]. Short iterations boost the importance of accurate effort estimation and a fast feedback cycle, which are well-known factors in waterfall projects. Scrum relies on self-organized teams. Thus, there is no manager who allocates tasks to team members or assists in designing solutions; those are issues that are decided by the team as a unit. The team is supported by two key players, the Product Owner (PO), the person who sets the product's vision and who has the authority to determine the sprint's backlog, and the Scrum Master (SM), who is responsible for managing the scrum's processes, including meetings and artifacts, and the terminology of the

Agile scrum. In addition, Scrum has four ceremonies that format each sprint, focusing on planning, progress, summarizing and feedback [6].

This section has reviewed the relevant professional literature on quality management, QVD, EDSD and Agile methodologies, highlighting current findings and trends. From the perspective of quality enhancement of Agile methodologies, it is apparent that conflicting views still exist, with no apparent consensus having yet emerged. Our central proposition is that a software organization's development model is an important pivot and a crucial source of value creation. Since EDSD is an adaptable ethical platform for Agile projects, it seems logical to have it act as a value driver. The present study seeks to provide information on the development of quality processes using ethical tools. For educational theorists and researchers interested in portfolios, it is hoped that our study will promote a better understanding of the importance of software engineering as a regulated profession.

3. The Present Study's Design

3.1. Sample and Procedure

This study sought to compare the impact of ethical considerations on selected software quality parameters in Agile projects. Using an experimental design, we chose a large global software organization that had hundreds of Agile scrum teams and was involved in projects on a worldwide scale. We decided to focus on one project that comprised multiple teams with the same scope of work in terms of size, complexity and location. The differences in terms of time zones or other technical and cultural aspects were controlled and were not considered relevant to the experiment.

To minimize external distracting factors that might have impacted our experiment's results (e.g., location, size, seniority), we concentrated on 14 engineers who were located in the same region and who were targeted to work on the same project, prior to the formation of the scrum teams. Next, we categorized them into two groups: senior and junior engineers. Whereas the senior engineers group comprised engineers with more than five years of experience, the junior engineers group included engineers with less than five years of experience. This step was taken to ensure that the seniority level would be distributed equally among the teams. Eventually, we ended up having eight senior engineers and six junior engineers, who gave their consent to participate in the experiment. Using a random selection process, we split the members into two teams. Each team had the same number of junior and senior engineers, and we named them Group A and Group B. Each group had seven members: four senior engineers and threes junior ones. The team members were not given any choice as to which group to belong to, and each engineer had the same chance of being chosen for either of the two teams. Group A was designated as our test group, in which intervention would occur, as opposed to the control group, Group B, where no intervention was to occur.

3.2. The Experiment's Design and the Nature of the Intervention

The variable we were trying to investigate was the effect of the implementation of ethical considerations or tools, namely, the impact of EDSD, especially its testing and verification section. Thus, in accordance with the experiment's design, Group A would have to answer five additional questions during the planning session of each sprint, and for each user story. In order to collect enough data, the entire experiment lasted for six sprints, or 12 weeks. Group B did not benefit from EDSD's testing and verification sections and executed tasks via the known Agile scrum development methodology without any additional requirements or enhancements. This arrangement assured that any quality measurement impact on Group A's deliveries would have a strong connection with EDSD's implementation. In every sprint planning, the scrum PO in Group A went over the backlog with the team. For every user story, the members of the group answered the five questions from EDSD's testing and validation section. The idea was to make sure that all questions would receive a "YES" answer and team members were asked to fill in gaps in order to get a "YES" for every question. For instance, question 3 ("Is there a set of mandatory tests?"). In case one of the user stories did not have a set of mandatory tests, the team would have to define the set prior to the sprint's kickoff. Only when all the user stories met the requirements of EDSD's questions was the team allowed to start working on the actual content delivery.

The PO in Group B went over the backlog before each of the six sprints, during the sprint's planning session, making sure that every user story was allocated the required resources and that the stories were clear to everyone. The team was allowed to start working on content delivery once the planning meeting was over.

3.3. Measurements

When it comes to defining the key performance indicators for quality measurement, there is no single answer that can cover what all organizations believe and practice in their day-to-day work because of the industry's particular specialty and because of the organization's size, resource allocation strategy, etc.; these technical aspects impact on the organization's success factors and milestones. To avoid confusion and to provide a focused approach, this study based itself on three main quality parameters.

Story points are an arbitrary measure used by Agile teams, especially in Scrum. They are used to measure the effort required to fully implement a user story. These numbers tell the team how difficult the story is, the particular difficulty being related to complexity, dependency, prerequisites, etc. Story points usually range in T-Shirt sizes: small, medium, large and extra-large. The baseline size is determined by the team [16] [24].

3.4. Product Functionality

Product functionality refers to the congruence between software requirements and actual delivery on the one hand and the percentage of customer require-

ments met on the other, with the attention being focused on significant cases. We measured the number of functional defects or the total number of non-working software pieces in each iteration, during and after delivery phases.

3.5. Cost of Ownership

Cost of ownership refers to many aspects of the development cycle. We measured the cost of ownership, the percentage of resource capacity in human resource (HR) days for every user story, the actual allocation to maintain story content delivery after release, the addressing of changes in code without the introduction of new innovative content, etc.

3.6. Schedule

This parameter means the delivery of the product in accordance with the committed timeline. We measured the time to market using HR day units for every user story, indicating the overall time from user story readiness to product/feature final delivery.

Time to market and cost of ownership are measurements that are relative to user stories, whereas the number of functional defects is measured per sprint or per iteration since defects are not always correlated to one user story and can potentially occur because of more than one user story.

For each quality indicator, measurements were categorized, based on the size or type of the user story or defect. The following provides more details of each indicator's measurement scale:

- Time to Market (TTM) + Cost of Ownership (COO) in story points: Small, Medium, Large and Extra-Large.
- Functional defects-Minor (A defect that will not cause a failure in the execution of the product), Major (A defect that will cause an observable product failure or a departure from requirements), Fatal (A defect that will cause the system to crash or close abruptly or which might impact other applications).

4. Results and Data Analysis

4.1. Time to Market (TTM)

While the definition of time to market (TTM) can vary depending on the organization, industry and product, for this study and in collaboration with the organization we were examining, TTM was defined as the period of time between the point where a user story is ready-in other words, all involved team members understand and are aware of all the requirements and of the definition of what must be done-to the final delivery time. TTM and the search for ways of optimizing it are critical factors that directly impact revenue and cost when we holistically view the end-to-end status of a product. Rationally, TTM is very dependent on a controlled process, on tracking abilities and on the early reduction of the risk of sudden unknowns. Hence, an ethical framework, especially EDSD, delimits and raises prerequisites and potential risks by leveraging questions at

the planning phase.

User stories vary in size. There are four different category sizes: From small to extra-large. Therefore, it makes sense that, the bigger the size of a user story, the longer its TTM. According to the organization's records, user stories belonging to the small size bucket usually take between 5.5 to 6 HR days, depending on the project and the team. Table 1 describes the TTM for the four size categories after one iteration, the measurement being made in HR days.

Looking at the different user story sizes, we can see that there is a delta between both groups after the first iteration. This fact does not necessarily prove that EDSD's testing and validation had a direct impact, but it should be noted that differences were obtained for all the sizes.

The trend of lower TTM for all story sizes was continuous after three iterations (Table 2) in the test group. Moreover, the deltas were becoming bigger over time. For example, in iteration 1, the delta in large sized stories was 0.3 HR days, while in iteration 3 it was 1.4 HR days. Looking at the results in the control group, we can see that the measurements after iteration 3 were similar to previous ones.

After six iterations, we could see significant TTM measurement differences between the test and control groups, in addition to a constant reduction in our test group from the 1st to the 6th iteration, leading to ~20% - ~40% less TTM across different story sizes. On the other side of the experiment, measurements were almost the same in our control group. Table 3 describes the full results after six iterations. The findings show that EDSD's testing and validation section was impacting TTM for all user story sizes, especially in small and extra-large stories, with a reduction of 38.46% and 33.81% respectively, compared to the traditional Agile scrum methodology used in our control group.

Table 1. TTM after 1 iteration.

Story size	TTM test group (O1)	TTM control group (O2)
Small	5.2	5.8
Medium	7.5	7.1
Large	10.1	9.8
Extra-Large	13.9	14.3

Table 2. TTM after 3 iterations.

Story size	TTM test group (O1)	TTM control group (O2)
Small	4	6
Medium	6.9	7.1
Large	8.8	10.2
Extra-Large	11.4	14.1

Table 3. TTM after 6 iterations.

Story size	TTM test group (O1)	TTM control group (O2)
Small	3.2	6.1
Medium	6.1	6.9
Large	7.2	10.15
Extra-Large	9.2	14.4

4.2. Cost of Ownership (COO)

The Cost of Ownership (COO) for a software project is the sum of all direct and indirect costs incurred for that software. COO is a critical part of the return of investment calculation [25]. In our case, COO related to the user stories, meaning how many HR days the team needed to spend on "maintaining" user stories developed in previous sprints. Thus, COO has a direct link to the nature of good design and development.

COO applies to both the organization and the customer since the allocation of resources to maintain a piece of software instead of work on new content eventually leads to longer time to market and higher costs. Obviously, high quality achieved prior to the release of the product prevents the risk of the allocation of resources to maintain quality or correct problematic issues.

Like TTM, COO relates to user stories. Thus, the four categories of small, medium, large and extra-large were relevant in our measurement. **Table 4** describes the COO for each user story size after the second iteration.

The numbers reflect the percentage of HR days out of the entire team capacity that was required to be allocated in order to support user stories that were delivered from earlier sprints. Support could have included correction of discovered problematic issues, or a change in the delivery date due to a variety of reasons, such as change in customer requests, the incongruence of the design and the scale, etc.

For example, there were seven members in each team and the assumption was that the second iteration had full resource capacity in both teams, the result being 70 HR days for each team (7 engineers × 10 working days for a period of two weeks). Our test group allocated 6.3 HR days on extra-large user stories delivered from previous sprints, while our control group spent 7.08 HR days.

Looking at the entire COO allocation in the second iteration, we found that the numbers added up to 25% of resource allocation in our test group and almost 28% in our control group. Clearly, high COO is expensive and impacts overall quality. **Table 5** paints a fuller picture of the impact of ethical considerations on the COO of user stories in scrum teams.

In our test group, we managed to reduce the total COO for all the user story categories in sprint 6% to 17.4% when it originally was ~30%, the delta totaling almost nine HR days in each sprint and thus becoming available for the development of new content instead of support for old content. Our control group

Table 4. COO after 2 iterations.

Story size	COO test group (o1)	COO control group (o2)
Small	1.80%	2.00%
Medium	4.61%	5.00%
Large	9.60%	10.66%
Extra-Large	9.00%	10.12%

Table 5. COO after 6 iterations.

Story size	COO test group (O1)	COO control group (O2)
Small	0.06%	2.07%
Medium	3.36%	4.78%
Large	6.97%	10.91%
Extra-Large	7.03%	10.20%

started with 27.99% HR days for COO and ended up with 27.96%. Clearly we can see that EDSD's testing and validation section is correlated with the reduction of COO overtime.

4.3. Functional Defects

The number of functional defects is the total number of non-working software pieces in the pre and post-delivery phases that were reported in the project tracking system by team members. Team members test features by feeding them input, examining the output and comparing it to the original requirements from the PO or business owner. Functional testing ensures that the requirements are properly met by the product [25].

As described in a previous section, functional defects in this study had three different severity levels: minor, major and fatal. Hence, we had three different measurements for each iteration. **Table 6** presents the results after the first iteration. From our earlier COO measurement, the findings showed that the use of EDSD during the planning process lowered the percentage of COO for the various user stories. As already noted, COO also includes defect correction. Thus, it makes sense that one of the root causes of a reduction of the COO is the reduction of the number of functional defects. **Table 7** lists the number of found defects in both groups after five iterations.

All in all, there was a reduction of almost 60% in minor defects and 100% in major and fatal defects in our test group. The opposite happened in our control group. The figures were higher for minor defects and remained almost the same for the other two categories. Clearly, we saw a reduction in the number of functional defects in our test group from one iteration to the next, the result being a lower COO and a lower TTM.

Table 6. No. of functional defects after 1 iteration.

Defect severity	No. of functional defects in test group (O1)	No. of functional defects in control group (O2)
Minor	12	17
Major	5	8
Fatal	1	2

Table 7. No. of functional defects after 5 iterations.

Defect severity	No. of functional defects in test group (O1)	No. of functional defects in control group (O2)
Minor	4	15
Major	0	10
Fatal	0	0

5. Significance and Contributions of the Research

The main contribution of this research study is the major findings regarding the positive effect ethical tools have on quality, as we have discussed in detail in the section on the results and on the data analysis. In addition, this research study has investigated the application of new practical approaches and methodologies to quality and ethics. The investigation of new approaches will hopefully stimulate debate and provide opportunities for further research as discussed in the previous section.

6. Limitations and Recommendations for Future Research

This study has focused on two Agile scrum teams in a large global software organization. The teams started working on the same project, measured specific quality parameters and were escorted by an ethical framework. The assumption that the same results will be obtained for every Agile team can be mistaken; therefore, we have provided additional research suggestions.

Several Agile methodologies exist, all postulating different practices regarding the manner for building software systems. They all have their strengths and weaknesses. There has also been considerable competition among them in terms of popularity and suitability in different environments. Building on a common ethical tool for Agile methodologies, the best features of each methodology should be leveraged and should be combined into a consolidated, integrated and unifying meta-framework. Several definitions exist to help improve the software quality process. Like the methodologies, each definition has its strengths and weaknesses. Similarly, as in the case of the methodologies, the best attributes of each definition should be leveraged and combined into a consolidated, integrated and unifying software quality definition.

7. Conclusion

Our research study clearly indicated that ethical tools lead to higher quality in

Agile software projects. The findings of our study showed that the use of EDSD's validation and testing section as a part of scrum methodology and during sprint planning reduces time to market, cost of ownership and the number of functional defects. Thus, it can be concluded that ethical tools are a quality driver in Agile software projects. This point was brought home in the experiment that was carried out and the findings of our study clearly indicate that software development escorted by ethical tools results in better quality as compared with software that has been developed without ethical tools.

References

[1] Wong, W.E., Debroy, V. and Restrepo, A. (2009) The Role of Software in Recent Catastrophic Accidents. Department of Computer Science, University of Texas, Dallas.

[2] Avison, D. and Fitzgerald, G. (2006) Information Systems Development: Methodologies, Techniques and Tools. 4th Edition, McGraw Hill, London, 395-418.

[3] Schach, S. (2010) Object-Oriented and Classical Software Engineering. 8th Edition, McGraw-Hill, New York, 154-294.

[4] Abrahamsson, P., Oza, N. and Siponen, M.T. (2010) Agile Software Development Methods: A Comparative Review. In: *Agile Software Development*, Springer, Berlin Heidelberg, 31-59. https://doi.org/10.1007/978-3-642-12575-1_3

[5] Beck, K. (2000) Extreme Programming Explained: Embrace Change. Addison-Wesley, Reading, MA, 17-34.

[6] Cunningham, W. (2001) Manifesto for Agile Software Development. http://Agilemanifesto.org

[7] Highsmith, J. (2004) Agile Project Management. Addison-Wesley, Boston, MA.

[8] Santos, M.A., Bermejo, P.H.S., Oliveira, M.S. Tonelli, A.O. (2011) Agile Practices: An Assessment of Perception of Value of Professionals on the Quality Criteria in Performance of Projects. *Journal of Software Engineering and Applications*, **4**, 700-709. https://doi.org/10.4236/jsea.2011.412082

[9] Dyba, T. and Dingsoyr, T. (2008) Strength of Evidence in Systematic Reviews in software Engineering. *Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ACM, Kaiserslautern, 9-10 October 2008, 178-187. https://doi.org/10.1145/1414004.1414034

[10] Donald, M. (2016) Agile and Conventional Methodologies: An Empirical Investigation of Their Impact on Software Quality Parameters, University of South Africa.

[11] Brandenburger, A.M. and Stuart, H. (1996) Value-Based Business Strategy. *Journal of Economics & Management Strategy*, **5**, 5-25. https://doi.org/10.1111/j.1430-9134.1996.00005.x

[12] Boehm, B.W., Brown, J.R., Kaspar, H., Lipow, M., Macleod, G. and Merrit, M. (1978) Characteristics of Software Quality. North-Holland, Amsterdam.

[13] Juran, J.M. (1992) Juran on Quality by Design: The New Steps for Planning Quality into Goods and Services. Free Press, New York.

[14] Pressman, R. (2010) Software Engineering: A Practitioner's Approach. 7th Edition, McGraw Hill, New York.

[15] Huo, M., Verner, J., Zhu, L. and Babar, M.A. (2004) Software Quality and Agile Methods. *Proceedings of the* 28*th Annual International on Computer Software and Applications Conference*, Hong Kong, 28-30 September 2004, 520-525.

[16] Schwaber, K. (2004) Agile Project Management with Scrum. Microsoft Press, 102-336.

[17] Mnkandla, E. and Dwolatzky, B. (2006) Defining Agile Software Quality Assurance. *International Conference on Software Engineering Advances*, Tahiti, 29 October-3 November, 36.

[18] Petersen, P., Andersen, O., Heilesen, J.H., Klim, S. and Schmidt, J. (1989) Software Quality Drivers and Indicators. *Proceedings of the 22nd Annual Hawaii International Conference on System Sciences*, Washington DC, 3-6 January 1989, Vol. 2.

[19] Peslak, A.R. (2007) Improving Software Quality: An Ethics Based Approach. Ph.D. Thesis, Penn State University, State College, PA.

[20] ACM (2015) Software Engineering Code of Ethics and Professional Practice.

[21] National Society for Professional Engineers (2007) Code of Ethics.

[22] Gotterbarn, D., Miller, K. and Rogerson, S. (1997) Software Engineering Code of Ethics. *Communications of the ACM*, **40**, 110-118. https://doi.org/10.1145/265684.265699

[23] Lurie, Y. and Mark, S. (2015) Professional Ethics of Software Engineers: An Ethical Framework. *Science and Engineering Ethics*, **22**, 417-434.

[24] Sommerville, I. (2011) Software Engineering. 9th Edition, 618-668.

[25] Sommerville, I. (2007) Software Engineering. 8th Edition, Addison-Wesley, Boston, MA.