Empirical Studies on Quality in Agile Practices: A Systematic Literature Review

Panagiotis Sfetsos
Department of Informatics,
Alexander Technological Educational Institution,
Thessalonki, Greece
sfetsos@it.teithe.gr

Ioannis Stamelos
Department of Informatics,
Aristotle University
Thessalonki, Greece
stamelos@csd.auth.gr

Abstract— **Nowadays one key question for most organizations is which of the agile practices should be implemented to improve product quality. This systematic literature review surveys studies published up to and including 2009 and attempts to present and evaluate the empirical findings regarding quality in agile practices. The studies were classified into three groups: test driven or test first development, pair programming, and miscellaneous agile practices and methods. The findings of most studies suggest that agile practices can improve quality if they are implemented correctly. The significant findings of this study, in conjunction with previous research, could be used as guidelines for practitioners on their own settings and situations.**

Keywords- quality; ISO/IEC 12207; ISO/IEC 9126; empirical studies; agile methods; agile practices; test-driven development; test-first development; pair programming; systematic literature review

I. INTRODUCTION

Although software quality is critical for the success of a software product, as a concept it is difficult to define, describe, understand and measure [61]. Quality, according to ISO 8402, is: *'The totality of characteristics of a product or service that bear on its ability to satisfy stated and implied needs'* [62]. The Institute of Electrical and Electronics Engineers (IEEE), defines quality as *'the degree to which a system, component, or process meets specified requirements and customer/user needs or expectations'* [63]. Both definitions are focused on satisfying the customer's need for the software product.

To address the issues of software process - and product quality in agile methods, we considered two well known industry standards, the ISO/IEC 12207 [68] and ISO/IEC 9126 [64-67] respectively. The ISO/IEC 12207 standard provides a framework for software life-cycle processes. We focused only on the *development process area* of this standard, because most of the agile practices could be mapped directly on activities in this process area. Planning game and sprint planning include the activities that could be mapped on the requirements definition activities in development process area, while test driven development, pair programming and continuous integration belong to the implementation and testing activities in this area.

The ISO/IEC 9126 standard, intending to ensure the quality of all software products, specifies software product quality characteristics and sub-characteristics and associated metrics. The standard is divided into four parts: quality model, external metrics, internal metrics and quality in use metrics. The quality model [64] classifies software quality in a structured set of characteristics and sub-characteristics. It provides a framework for organizations to define a quality model for a software product, by specifying target values for quality metrics. External metrics [65] are applicable to running software, while internal metrics [66] are those which do not rely on software execution (i.e. measure the software itself - static measures). The 'quality in use metrics' [67] are only available when the final product is used in real conditions. In this systematic review we focused on all three types of metrics (external, internal and quality in use metrics).

Agile methodologies promote evolutionary changes within software development processes. They rely on a set of best practices that are considered to increase quality assurance and control. It can be stated that the bunch of these best practices forms a disciplined process with built-in quality [2]. The quality assurance and control procedures are integrated across the entire life-cycle development, from requirements to the final release. Agile methods build quality into the product through a combination of best practices, inducing a different perspective on quality management. Many studies support and evangelize the advantages of agile practices with respect to quality.

Agile development completely redefines quality assurance work, from formal roles to day-to-day activities. Value is created and quality is assured through all the development phases, when all the parts are integrated into one cohesive whole. The developers, following a set of engineering best practices, such as pair programming [3][4] and test-driven development (test-first development and refactoring) [5][6], deliver software of higher quality, faster, with a higher acceptance to end-users. *Pair programming,* as an intensely social and collaborative activity, capitalizes on developers' unique skills, experiences, idiosyncrasies, and personalities [7]. This practice serves as a continual design and code review process resulting in the reduction of the defects [4] and in the improvement of design and code quality. *Test-driven development*

(TDD) or *Test-first development (TFD)* and *refactoring*, is an iterative and incremental approach to programming. In TDD, developers write automatically executable tests (test cases) prior to writing the code they test. Developers do detailed design and think about new functionalities before writing code. In combination with acceptance tests, which are being used as requirement-artifacts, and code refactoring [8] developers are expected to achieve high quality levels. *Refactoring* is a disciplined way to make small changes to source code improving its design without changing its external behavior. During refactoring, developers reconstruct the code through code inspection, and achieve error reduction. Agile methods require that the *customer* be involved in all the development phases, a practice that provides vision in the form of high-level requirements, basic acceptance criteria and perceived satisfaction for the final product. By employing such practices in an evolutionary, iterative, and incremental development process, the key business users become strong partners in assuring quality. In agile methods quality, it is not a single persons' job; all key business users are responsible for ensuring that the application is fit for purpose.

The objective of this systematic review is to evaluate according to the ISO/IEC 12207 and ISO/IEC 9126 standards, synthesize, and present, the empirical findings on quality in agile methods. This review can help practitioners to improve their agile practices implementation and researchers to understand the current "state of the art" of quality approaches and metrics in agile practices.

The rest of the paper is organized as follows: Section II presents the systematic review method, section III presents the results of the survey, including limitations and section IV concludes the paper.

II. SYSTEMATIC REVIEW METHOD

A. *Introduction*

We undertook this systematic review following the established review process in [1] and [9] for identifying, assessing and interpreting all available related research evidence about quality approaches and metrics in agile practices. The study was conducted in the following distinct stages: development of review protocol, formulation of the research questions, identification of inclusion and exclusion criteria, identification of relevant literature by conducting a comprehensive and exhaustive search, selection of primary studies based on inclusive/exclusive criteria, data extraction and synthesis of evidence, and interpretation of results. In the rest of this section, we describe such stages in detail.

B. *Protocol development*

We developed a protocol for the systematic review by following the guidelines and procedures as described in [1] [9]. In this protocol we specified the research questions, search strategy, inclusion and exclusion criteria, quality assessment, data extraction, and methods to synthesize the evidence in order to answer the research questions.

C. *Research Questions*

The objective of the review is to answer the following research questions:

1. What is the current state of knowledge on quality in agile practices?
2. Which are the most significant practices for achieving quality in agile development?

D. *Inclusion and exclusion criteria*

To select the primary studies for the review we considered the following inclusion criteria:

– Studies had to provide empirical data on quality issues and metrics in agile practices and passed the quality assessment procedure (see subsection G).

– Studies could come from both Academia and Industry.

– Quantitative and qualitative research studies should be published up to 2009.

– Studies should be written in English.

Exclusion criteria were:

– Studies did not focus on quality issues in agile practices.

– Studies did not present empirical data.

– Studies presented only the opinion of the researcher(s), ''lessons learned" studies (papers without a research question and research design) and simulation studies.

E. *Literature sources and search criteria*

The search process included electronic databases and manual searches of conference proceedings. The following electronic databases were searched:

- IEEE Xplore,
- ACM Digital Library,
- Kluwer Online,
- SpringerLink,
- ScienceDirect – Elsevier,
- ISI Web of Science,
- CiteseerX Library,
- Wiley Inter Science Journal Finder.

Moreover, all volumes of the following conference proceedings were searched: XP, Agile Development Conference, and XP/Agile Universe.

In the 1st stage of the search process (see next section) the titles, abstracts, and keywords of the articles in the included electronic databases and conference proceedings were searched using the following search terms:

1) agile practice AND empirical AND quality

2) agile software development AND quality

3) pair programming AND empirical AND quality

4) test driven development OR test first development AND empirical AND quality

5) refactoring AND empirical AND quality

6) planning game AND empirical AND quality

7) on site customer AND empirical AND quality

These search terms were also combined by using the Boolean ‘‘OR” operator, ensuring that an article needed to include any one of the terms to be retrieved.

F. *Selection of primary studies*

In our preliminary search, using all possible combinations of search terms, we retrieved 535 articles (see Fig. 1). The search concerned firstly the titles, abstracts, and keywords of the articles in the inspected electronic databases and conference proceedings. All those papers were inserted, stored and organized in a spreadsheet for further evaluation. Secondly, the relevant citations for the selected papers were recorded and sorted separately in the tool EndNote X3. At stage 2, both authors, in a collaborative assessment process, excluded those studies that did not include empirical data and were outside the scope of this systematic review. At this stage, 123 papers were considered for detailed quality assessment and were inserted in a new EndNote database and spreadsheet.

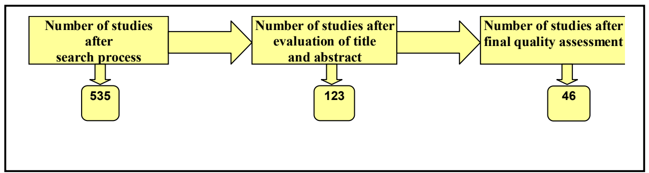
Number of studies after search process
535
Number of studies after evaluation of title and abstract
123
Number of studies after final quality assessment
46

Figure 1. Primary studies after each stage of screening process.

G. *Quality assessment*

To assess the quality of the 123 studies, we developed a quality criteria checklist screening the major important quality criteria expected from the review. The list contains 11 criteria based on quality criteria adapted from [1], [10], [11] and [12]. These criteria cover three main quality characteristics that need to be considered when appraising studies: ***rigor, credibility*** and ***relevance***.

Rigor answers the question: “Does the study follow a rigorous and appropriate approach in the implementation of the various methods used?” Three criteria were developed to assess a study’s rationale, aims and context, answering the following secondary questions:

1. Does the paper present an empirical study?
2. Are the aims of the research clearly stated?
3. Is the context of the study adequately described?

Another five criteria were used to assess the validity of data collection, the analysis methods, and the trustworthiness of the findings. These criteria answer the following sub-questions:

4. Is the research design appropriate to address the aims of the research?
5. Is the recruitment strategy appropriate to the aims of the research?
6. Is there a control group with which to compare treatments?
7. Is the data collected in a way that addressed the research issue?
8. Is the data analysis sufficiently rigorous?

Credibility answers the question: “Are the findings of the study valid, meaningful and well-presented?” Two other criteria were developed to assess whether the findings of the study are valid and meaningful. These criteria answer the following sub-questions:

9. Does the involvement of researcher affect the results (“causing bias”)?
10. Is there a clear statement of findings?

Relevance answers the question: Are the findings relevant and useful for the software industry and the research community? One further criterion was developed to assess the relevance of a study, answering the following sub-question:

11. Does the study provide value for research or practice?

Each of the 11 criteria was graded on a dichotomous ‘yes/no’ or ‘1/0’ scale. The first of these criteria was used as the basis for the inclusion or exclusion of a study. The total sum of grades for the 11 criteria was used as a confident measure for grading the quality of each assessed study. Of the 123 studies assessed for quality, only 46 passed the assessment. All disagreements were resolved by discussion that included both researchers. The results of the quality assessment are shown in Table II (Appendix A). In this table a ‘1’ indicates ‘yes’ (or OK) to the question, while ‘0’ indicates ‘no’ (or not OK).

H. *Data Extraction*

Data extraction was carried out on the 46 studies that passed the quality assessment process. The data from each one of the studies were first recorded using a data extraction form and then saved as a new textual document file. The following data were recorded in the data extraction form of each paper: title, abstract, type of empirical study (experiment, case study, survey, mixed, etc.), research environment, population, agile practice in use, research question (if any), main results - evidence on quality in the practice (if any) and conclusions. This form helped us extract, in a table, all details we needed for each of the studies (see Table III - Appendix B).

The data extraction process faced some difficulties because the quality achievements were not clearly reported in certain studies. Due to this fact, all data from the studies were extracted by both authors in consensus meetings. All disagreements were resolved by discussion during these meetings.

I. *Synthesis of findings*

The study followed the *conceptual synthesis method* which brings together different understandings or concepts with purpose to create a new concept or concepts. The most known approach of this method is the *Meta-ethnography approach* which combines the results of different studies to create an understanding of the phenomena under study greater than the individual ethnographic studies [13]. Following this approach, in the first phase we identified and recorded in tables the main concepts concerning findings about quality issues, from each study. In the second phase the findings were interpreted and compared. This phase revealed the different approaches used in these studies for measuring and confirming quality (see section III). It seems that the difference in findings between some studies is mainly caused by the difference in methods, characteristics, or metrics they utilized. Finally, the findings were translated, compared, and synthesized in ordered to answer our research questions.

III. RESULTS

Of the 46 empirical studies considered in the systematic review, 24 were experiments, 17 were case studies and 5 were mixed studies (experiment/case study, experiment/survey, case study/survey and survey/qualitative). The selected studies cover a wide range of the researched topics and conducted in different settings, varying from professional projects to university courses. We classified studies in three main categories: quality in TD/TF development, quality in pair programming and quality in other practices. The types of studies per category are summarized in Table I.

TABLE I. TYPE OF STUDIES

	Experiments	**Case Studies**	**Mixed**	***Total***
TDD/TFD	8	8	2	**18**
Pair Programming	14	3	2	**19**
Other agile practices	2	6	1	**9**
Total	**24**	**17**	**5**	**46**

We now address our research questions, starting by discussing what we found regarding quality in agile practices.

A. *Quality in TD/TF development*

Recent empirical studies, included also in this systematic review, re-considered test-driven development as the most critical enabling practice for quality in the agile software development. Most of the experiments [17], [19], [20], [23], [28], [30], [31] and case studies [14], [15], [16], [21], [22], [24], [26], [27] showed extensive improvement in external quality. External quality in a control setting (i.e. experiment) was usually measured by the number of passed acceptance tests or the total number of defects or number of defects/KLOC (defect density). In the case studies or mixed studies, external quality was usually measured by the number of defects found before release or defects reported by customers. The defects were decreased from 5% - 45% [21], [22], [24], [14], [26] [28] up to 50% - 90% [14], [16], [26], [27], [30]. Case studies showed a stronger improvement in external quality than the experiments, and this may be due to the controlled settings and the limitations of time. Only two experiments showed non significant differences in the external quality [25], [29].

Internal quality was usually measured by different code metrics such as code size, cyclomatic complexity, coupling and cohesion. Improvement in internal quality was reported in [15], while no significant differences were found in [18] and [20]. Code and design complexity were reported to decrease [15], [20], especially for small design units. Code reusability was reported to increase [29], while code cohesion did not improve [18]. One study showed that the total development cost was decreased because of the decrease of the avoidable fault cost [21]. Two studies included effort as a research variable [17], [19]. One of these studies showed that effort for testing was increased [17], while another showed that the total development effort was decreased [19]. Two case studies showed that development time was increased [14], [16]. Productivity studies reported contradictory results. Two experiments reported increased productivity [17], [19], one case study reported decreased productivity [15], while another one showed no significant differences [27].

B. *Quality in Pair Programming*

Pair Programming, already used with success in both industry and academia, has been exhaustively researched during the last years. Design and code quality improvement, varying from 15% [35] up to 65% [33], is one of the most significant results reported by most of the reviewed studies [32], [33], [34], [35], [36], [37], [38], [40], [41], [42], [44], [45], [49]. For unknown, complex and challenging programs, pair programming proved to provide better code than solo programming [32], [46], [47], [48], [50]. Fewer defects in code were reported in [33], [47], and no significant difference in code defects was reported in [39]. The results for productivity [43], [44] and time spent [32], [33], [34], [39], [40] were contradictory. Many other benefits were reported such as improved quality of teamwork and communication [45], [49], code spreading and understanding [33], [34], better information and knowledge transfer [41], [49], better and faster design of algorithms [44], increased morale [38], and more confident programmers [50]. Disadvantages that were reported include increased effort [35] and cost [40], especially for higher quality products [46], minor loss in efficiency [34], schedule problems and the personality conflicts [33]. Compatibility issues of pair programmers were researched in some of the studies included in this review [37], [45], [48]. Certain combinations of pair programmers concerning skills, knowledge and experiences can increase productivity [48]. Moreover, certain personality traits ensure higher quality code, namely Openminded and Responsible [37], or heterogeneous personalities/temperaments [45].

C. *Quality in other practices*

Most of the studies applied XP - key practices, such as planning game, pair programming, test driven development and refactoring, in combination, were found to lead in higher quality [51], [52], [53], [54], [55], [56], [59]. Only in one study no difference was reported in either internal or external quality between the XP and the traditional teams [57]. Planning game was found to lead in better work estimation of the work size [55]. Refactoring was found to increase quality [51], [59]. Productivity was increased in [52], [53], [56], [59].

XP practices were found to produce better results for small teams [59]. Scrum process was used in one case study; in this study an improvement in both the product quality (30%) and in customer satisfaction was reported [58].

D. *Limitations of this systematic review*

The internal validity (credibility) and external validity (generalizability) of the study results are defined in Lincoln and

Guba [60]. The two limitations of this systematic review are bias in the selection of the studies and inaccuracy in data extraction. To minimize the bias in the selection process, we developed a research protocol which defined the research questions, and review process. Based on this protocol we developed the search terms for the identification of the relevant literature. However, it must be emphasized that the search process was a difficult process, because many quality key terms were not standardized according to some international quality standard (i.e. ISO/IEC 12207 or ISO/IEC 9126) or were used as context- or language- specific. To minimize the selection bias we utilized a multistage process that involved both researchers. Data extraction process was hindered by the way some studies reported their context, their research questions, their sampling process, the methods and the metrics used, the data collection process and their results. Therefore, there exists a possibility that some of the extracted data are inaccurate. The generalizability of the results may be hindered also by the uncontrolled variables and metrics used in some of the studies. It is impossible to directly compare the results of these studies.

IV. CONCLUSION

This paper presented the initial results of a systematic review evaluating quality approaches and metrics, according to ISO/IEC 12207 and ISO/IEC 9126 standards, in agile practices. We identified 535 studies of which 46 were found to be empirical studies with acceptable rigor, credibility and relevance. The studies were categorized in three main groups concerning quality: test driven or test first development, pair programming and other practices.

A number of reported benefits and limitations of agile practices, concerning quality, was identified. A wide range of improvements were reported for TDD or /TFD and refactoring, including among others improvement of external quality. TDD helps significantly in the improvement of software quality, in terms of decreased fault rates, when employed in an industrial context. Such effect was not so clear in the studies conducted in Academia, even though none of those studies reported decreased quality. The productivity effects of TDD were contradictory, and the results varied for different contexts.

Pair programming was found to be a successful agile practice. The main finding for this practice is that it strongly improves code and design quality. In addition, it improves the quality of teamwork, improving communication, understanding and knowledge transfer. In combination with TDD / TFD and refactoring, pair programming becomes a key practice for quality improvement. Planning game and on-site customer are also some of the XP-practices that contributed in quality improvement.

The initial findings of this study are expected to help managers and researchers, in the field of agile methods, to better understand how to approach quality issues when implementing the agile practices.

REFERENCES

[1] B. Kitchenham. “Guidelines for performing Systematic Literature Reviews in Software Engineering”, Version 2.3, Keele University and University of Durham, EBSE Technical Report, 2007.

[2] I. Stamelos and P. Sfetsos, “Agile Software Development Quality Assurance”, IGI Publishing, 2007, ISBN: 978-159904216-9.

[3] K. Beck. “Extreme programming explained: embrace change”, Addison-Wesley, Boston, USA, 2000.

[4] 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.

[5] D. Astels. “TestDriven Development: A Practical Guide”. Upper Saddle River, New Jersey, USA, Prentice Hall, 2003.

[6] K. Beck. “Test-driven development: By example”. Boston: Addison-Wesley, 2003.

[7] P. Sfetsos, I. Stamelos, L. Angelis, I. Deligiannis. “An Experimental Investigation of Personality Types Impact on Pair Effectiveness in Pair Programming”. Empirical Software Engineering, Volume 14, Number 21/April, 2009 p.p. 187-226.

[8] M. Fowler. “Refactoring: Improving the design of existing code”. Menlo Park, Calif.: Addison-Wesley Longman, Inc, 1999.

[9] J.P.T. Higgins, S. Green (Eds.), Cochrane Handbook for Systematic Reviews of Interventions, Version 5.0.0 (updated February 2008), The Cochrane Collaboration, 2008. Available from: <www.cochrane-handbook.org>.

[10] P. D. Leedy, J. E. Ormrod, Practical Research Planning and Design, Pearson Merril Prentice Hall, 2005.S

[11] L. Spencer, J. Ritchie, J. Lewis, L. Dillon, Quality in qualitative evaluation: A framework for assessing research evidence. London: Government Chief Social Researcher’s Office, 2003.

[12] Critical Appraisal Skills Programme (CASP), Available from: www.phru.nhs.uk/Pages/PHD/CASP.htm. .

[13] G.W. Noblit, R.D. Hare, Meta-Ethnography: Synthesizing Qualitative Studies, Sage Publications, London, 1988.

[14] N. Nagappan, E. M.Maximilien, T. Bhat and L.Williams. “Realizing quality improvement through test driven development: results and experiences of four industrial teams”, Empirical Software Engineering, Volume 13, Number 3, June, 2008.

[15] C. Sanchez, L. Williams, E.M. Maximilien. “On the Sustained Use of a Test-Driven Development Practice at IBM”, Proceedings of the AGILE 2007, pp. 5-14, 2007.

[16] T. Bhat, and N. Nagappan. “Evaluating the efficacy of test-driven development: industrial case studies”. Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering, ACM: 356-363, 2006.

[17] L.Huang and M. Holcombe. “Empirical investigation towards the effectiveness of Test First programming”. Information and Software Technology, 51(1):182-194, 2009.

[18] D. Janzen and H. Saiedian. Does Test-Driven Development Really Improve Software Design Quality? EEE Software, 25(2):77-84, March-April 2008.

[19] A. Gupta and P. Jalote. An experimental evaluation of the effectiveness and efficiency of the test driven development. In ESEM '07: Proceedings of the First International Symposium on Empirical Software Engineering and Measurement, pages 285-294, Washington, DC, USA, 2007.

[20] C.Desai, D. S. Janzen. Implications of integrating test-driven development into CS1/CS2 curricula. Proceedings of the 40th ACM technical symposium on Computer science education. Chattanooga, TN, USA, ACM: 148-152, 2009.

[21] L.O. Damm and L. Lundberg. Results from introducing componentlevel test automation and Test-Driven Development. Journal of Systems and Software, 79(7):1001-1014, 2006.

[22] L.O. Damm and L. Lundberg. Quality impact of introducing componentlevel test automation and test-driven development, in: P. Abrahamsson, N. Baddoo, T. Margaria, R. Messnarz (Eds.), Software Process Improvement, Vol. 4764 of Lecture Notes in Computer Science, Springer, 2007, pp. 187-199.

[23] G. Melnik and F. Maurer. A cross-program investigation of students' perceptions of agile methods. In ICSE '05: Proceedings of the 27th International Conference on Software Engineering, pages 481-488, 2005.

[24] R. A. Ynchausti, Integrating Unit Testing Into A Software Development Team's Process, in: M. Marchesi, G. Succi (Eds.), XP 2001: Proceedings of the 2nd International Conference on Extreme Programming and Flexible Processes in Software Engineering, Sardinia, Italy, 2001, pp. 84-87.

[25] M. Pancur, M. Ciglaric, M. Trampus, and T. Vidmar. Towards empirical evaluation of test-driven development in a university environment. In

EUROCON '03: Proceedings of the International Conference on Computer as a Tool, pages 83-86, 2003.

[26] L. Williams, E. M. Maximilien, and M. Vouk. Test-Driven Development as a Defect-Reduction Practice. In ISSRE '03: Proceedings of the 14th International Symposium on Software Reliability Engineering, pages 34-48, Washington, DC, USA, 2003.

[27] E.M. Maximilien, and L. Williams. Assessing test-driven development at IBM. Proceedings of the 25th International Conference on Software Engineering. Portland, Oregon, IEEE Computer Society: 564-569, 2003.

[28] B. George and L. A. Williams. A structured experiment of test-driven development. Information and Software Technology, 46(5):337-342, 2004.

[29] M. Müller, O. Hanger, Experiment about Test-First programming, IEEE Proceedings on Software 149 (5) pp. 537–544, 2002.

[30] S. H. Edwards. Using TestDriven Development in the Classroom: Providing Students with Automatic, Concrete Feedback on Performance. International Conference on Education and Information Systems: Technologies and Applications, USA, 2003.

[31] H. Erdogmus, M. Morisio and M. Torchiano. "On the effectiveness of the testfirst approach to programming." IEEE Transactions on Software Engineering 31(3): 226 - 237, 2005.

[32] E. Arisholm, Gallis, H., Dyba, T., Sjoberg, D., "Evaluating Pair Programming with Respect to System Complexity and Programmer Expertise", IEEE Transactions in Software Engineering, 33(2), pp. 65 - 86, 2007.

[33] A. Begel and N. Nagappan. "Pair programming: what's in it for me?", ESEM '08: Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measuremen, pp. 120–128, 2008.

[34] T. Bipp, A. Lepper, and D. Schmedding. "Pair programming in software development teams - An empirical study of its benefits." Information and Software Technology 50(3): 231-240, 2008.

[35] G. Canfora, A. Cimitile, F. Garcia, M. Piattini, C.A. Visaggio. "Evaluating performances of pair designing in industry", Journal of Systems and Software, Volume 80 , Issue 8 pp: 1317-1327, 2007.

[36] C. McDowell, L. Werner, H. Bullock, J. Fernald. "Pair programming improves student retention, confidence, and program quality." Commun. ACM 49(8): 90-95, 2006.

[37] J. Chao, and G. Atli, Critical Personality Traits in Successful Pair Programming. AGILE'06, IEEE Computer Society, 2006.

[38] J.T. Nosek, "The Case for Collaborative Programming," Comm. ACM, Vol. 41, No. 3, pp. 105–108, 1998.

[39] J. Nawrocki and A. Wojciechowski, "Experimental Evaluation of Pair Programming", proc. of European Software Control and Metrics (Escom), 2001.

[40] L. Williams, R. R. Kessler, W. Cunningham, and R. Jeffries, "Strengthening the Case for Pair Programming," IEEE Software, vol. 17, no. 4, pp. 19–25, 2000.

[41] C. McDowell, L. Werner, H. Bullock, and J. Fernald, "The Effects of Pair-Programming on Performance in an Introductory Programming Course," proc. Proceedings of the 33rd SIGCSE technical symposium on Computer science education, pp. 38–42, 2002.

[42] L. Madeyski. "Is External Code Quality Correlated with Programming Experience or Feelgood Factor?", In proc. XP 2006, LNCS 4044, pp 65-74, 2006.

[43] H. Hulkko, and P. Abrahamsson. "A Multiple Case Study on the Impact of Pair Programming on Product Quality". In: ICSE '05: Proceedings of the 27th International Conference on Software Engineering, New York, NY, USA, ACM Press 495–504, 2005.

[44] K.M.Lui and K.C.C Chan. "When Does a Pair Outperform Two Individuals?", LNCS, volume 2675/2003, XP 2003, pp. 225-233, 2003.

[45] P. Sfetsos, I. Stamelos, L. Angelis, I. Deligiannis. "An Experimental Investigation of Personality Types Impact on Pair Effectiveness in Pair Programming". Empirical Software Engineering, Volume 14, Number 21/April, p.p. 187-226, 2009.

[46] M.M. Müller. "Two controlled experiments concerning the comparison of pair programming to peer review." Journal of Systems and Software, 78(2): pp. 166-179, 2005.

[47] M.M. Müller. "Do programmer pairs make different mistakes than solo programmers?" Journal of Systems and Software 80(9): pp. 1460-1471, 2007.

[48] K.M. Lui and K.C.C. Chan. Pair programming productivity: novice–novice vs. expert–expert. Int. J. Human-Comput Studies 64(9):pp. 915–925, 2006.

[49] T.H. DeClue, Pair programming and pair trading: effects on learning and motivation in a CS2 course, Journal of Computing Sciences in Colleges, May 2003, 18(5), 2003.

[50] B. Hanks, C. McDowell, D. Draper, M. Krnjajic. Program quality with pair programming in CS1, Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education, pp. 176-180, 2004.

[51] S. Xu and V. Rajlich. "Empirical Validation of Test-Driven Pair Programming in Game Development". Proceedings of the 5th IEEE/ACIS International Conference on Computer and Information Science, 2006.

[52] L. Layman, L. Williams, L. Cunningham, Exploring extreme programming in context: an industrial case study, Agile Development Conference, 2004.

[53] S. Ilieva, P. Ivanov, E. Stefanova, Analyses of an agile methodology implementation, in: Proceedings 30th Euromicro Conference, IEEE Computer Society Press, 2004, pp. 326–333.

[54] C.A. Wellington, T. Briggs, C.D. Girard, Comparison of student experiences with plan-driven and agile methodologies, in: Proceeedings of the 35th ASEE/ IEEE Frontiers in Education Conference, 2005.

[55] B. Tessem, Experiences in learning xp practices: a qualitative study, in: XP 2003, vol. 2675, Springer Verlag, Berlin, pp. 131–137, 2003.

[56] G. Melnik, and F. Maurer, A cross-program investigation of student's perceptions of agile methods, in: International Conference on Software Engineering (ICSE), St. Louis, MI, USA, 2005.

[57] F. Macias, M. Holcombe, M. Gheorghe, A formal experiment comparing extreme programming with traditional software construction, in: Proceedings of the Fourth Mexican International Conference on Computer Science (ENC 2003), 2003.

[58] S. Lee and H.S. Yong, Distributed agile: project management in a global environment, Empirical Software Engineering, Volume 15, Nubr. 2, pp. 204-217, 2010.

[59] R. Moser, P. Abrahamsson, W. Pedrycz, A.Sillitti and G. Succi, A Case Study on the Impact of Refactoring on Quality and Productivity in an Agile Team, LNCS, Volume 5082/2008, pp. 252-266, 2008.

[60] Y.S.Lincoln, E.G.Guba . Naturalistic inquiry. Sage, Thousand Oaks, 1985.

[61] B. Kitchenham and J. Walker, A quantitative approach to monitoring software development, Software Engineering Journal, pp. 1-13, 1989.

[62] ISO, "ISO 8402 Quality Vocabulary," in International Organization for Standardization. Geneva, 1986.

[63] IEEE, "IEEE Std 1074 -1997 - Standard for Software Life Cycle Processes," 1998.

[64] ISO/IEC, "ISO/IEC 9126-1 Software engineering- Product quality- Part 1: Quality model," 2001.

[65] ISO/IEC, "ISO/IEC 9126-2 Software engineering -Product quality- part2: External metrics," 2002.

[66] ISO/IEC, "ISO/IEC 9126-3 Software engineering -Product quality- part3: Internal metrics," 2002.

[67] ISO/IEC, "ISO/IEC 9126-4 Software engineering -Product quality- part4: Quality In Use metrics," 2002.

[68] ISO/IEC, "ISO/IEC ISO/IEC 12207: Information Technology - Software Life Cycle Processes, 1995.

APPENDIX A

TABLE II. QUALITY ASSESSMENT OF STUDIES.

Study	1	2	3	4	5	6	7	8	9	10	11	Total
	Research	Aim	Context	Research design	Sampling	Control Group	Data collection	Data analysis	Relation ship	Findings	Value	
S1	1	1	1	1	1	0	1	1	0	1	1	9
S2	1	1	1	1	0	0	1	1	0	1	1	8
S3	1	1	1	1	0	0	1	1	0	1	1	8
S4	1	1	1	1	1	1	1	1	0	1	1	10
S5	1	1	1	1	1	1	1	1	0	0	1	9
S6	1	1	1	1	0	1	1	1	0	1	1	9
S7	1	1	1	0	0	1	0	1	0	1	1	7
S8	1	1	1	1	0	0	1	1	0	1	1	8
S9	1	1	1	1	0	0	1	1	0	1	1	8
S10	1	1	1	1	0	0	1	1	1	1	1	8
S11	1	1	1	0	0	0	0	1	0	1	1	7
S12	1	1	1	1	0	1	1	1	0	1	1	9
S13	1	1	1	1	0	0	1	1	0	1	1	8
S14	1	1	1	1	0	0	1	1	0	1	1	8
S15	1	1	1	1	1	1	1	1	0	1	1	10
S16	1	1	1	0	1	1	1	1	0	1	1	9
S17	1	1	1	0	0	0	1	1	0	1	1	7
S18	1	1	1	0	1	1	1	1	0	1	1	9
S19	1	1	1	1	1	1	1	1	0	1	1	10
S20	1	1	1	1	0	0	1	1	1	1	1	9
S21	1	1	1	1	0	0	1	1	0	1	1	8
S22	1	1	1	0	0	1	1	1	0	1	1	8
S23	1	1	1	1	0	0	1	1	0	1	1	8
S24	1	1	1	0	0	0	1	1	0	1	1	7
S25	1	1	1	0	0	1	1	1	0	1	1	8
S26	1	1	1	0	0	0	1	1	0	1	1	7
S27	1	1	1	1	1	1	1	1	0	1	1	10
S28	1	1	1	1	1	1	1	1	0	1	1	10
S29	1	1	1	0	0	0	1	1	0	1	1	7
S30	1	1	1	0	0	1	1	1	0	1	1	8
S31	1	1	1	0	0	0	1	1	0	1	1	7
S32	1	1	1	1	1	1	1	1	0	1	1	10
S33	1	1	1	1	1	1	1	1	0	1	1	10
S34	1	1	1	1	1	1	1	1	0	1	1	10
S35	1	1	1	1	1	1	1	1	0	1	1	10
S36	1	1	1	0	0	1	1	1	0	1	1	8
S37	1	1	1	0	0	1	1	1	0	1	1	8
S38	1	1	1	0	0	0	1	1	0	1	1	7
S39	1	1	1	1	0	1	1	1	0	1	1	9
S40	1	0	1	1	0	1	1	1	0	1	1	8
S41	1	1	1	1	0	1	1	1	0	1	1	9
S42	1	1	1	1	0	0	0	0	1	1	1	7
S43	1	1	1	1	1	0	1	1	0	1	1	9
S44	1	1	1	1	0	1	1	1	0	1	1	9
S45	1	1	1	1	0	0	1	1	0	1	1	8
S46	1	1	1	1	0	0	1	1	1	1	1	9

APPENDIX B

TABLE III. OVERVIEW OF PRIMARY STUDIES

ID	Authors	Type of study	Agile Practice	Research Environment	Popula-tion	Results
S1	N. Nagappan et al. (2008)	Case Study	TDD	Professional	3 teams	– The defect density of the four products decreased between 40% and 90% compared to similar projects that did not use the TDD practice. – 15–35% increase in initial development time after adopting TDD.
S2	C. Sanchez et al. (2007)	Case Study	TDD	Professional	1 team	– Improved external and internal quality for the same products. – The use of TDD decreases the degree to which code complexity increases.
S3	T. Bhat, and N. Nagappan (2006)	Case Study	TDD	Professional	6 (A) 5-8 (B)	– 15%(project B) - 35%(project A) longer development time – decreased defects/KLOC by 62%(project A) -76% (project B)
S4	L.Huang and M. Holcombe (2009)	Experiment	TFD	Academic	39	– external quality of delivered software applications increased with the percentage of time spent on testing regardless of the testing strategy – More effort on testing – 70% higher productivity but the improvement is not statistically significant.
S5	D. Janzen and H. Saiedian (2008)	-Experiment -Case Study	TDD	Mixed Professional Academic	19 - N/A	– test-first programmers are more likely to write software in more and smaller units that are less complex and more highly tested. – coupling analysis does not provide clear answers – cohesion is not improved
S6	A. Gupta and P. Jalote (2007).	Experiment	TDD	Academic	22	– improves external code quality (affected by the actual testing efforts) – reduces overall development efforts – improves developers' productivity
S7	C. Desai, and D. S. Janzen (2009).	Experiment	TDD	Academic	14	– Test-Last group had 39% more defects than their Test-First counterpart (external quality). – No increase of the internal quality
S8	L.O. Damm and L. Lundberg (2006)	Case Study	TDD	Professional	50	– 5-30% decrease in fault-slip-through rate – 60% decrease in avoidable fault cost – total project cost reduced by 5-6%
S9	L.O. Damm and L. Lundberg (2007)	Case Study	TDD	Professional	50	– ratio of faults decreased from 60-70% to 0-20%
S10	G. Melnik and F. Maurer (2005)	-Qualitative - Survey	-Agile practices -TFD	Academic	240	– 73% of students perceived that TF improves quality
S11	R. Ynchausti (2001)	Case Study	TDD-TFD	Professional	5	– improvements in quality ranged from 38% to 267% fewer defects.
S12	M. Pancur et al. (2003)	Experiment	TDD	Academic	34	– small difference in external quality (external tests passed) – lower code coverage
S13	L. Williams et al. (2003)	Case Study	TDD	Professional	9	– reduced defect rate by 40%
S14	E.M. Maximilien, and L. Williams (2003).	Case Study	TDD	Professional	1 team	– 50% lower defect rate. No productivity decrease.
S15	B. George and L. Williams (2004)	Experiment	TDD	Professional	24	– TDD programmers produce higher quality code because they passed 18% more functional black-box test cases.
S16	M..M. Müller, and O. Hagner (2002).	Experiment	TFD	Academic	19	– no differences in quality – better reuse with TDD
S17	S. H. Edwards (2003)	Experiment	TDD	Academic	59	– 45 % fewer defects with TDD
S18	H. Erdogmus et al. (2005)	Experiment	TFD	Academic	24	– the minimum external quality increased with the number of tests – TF-students wrote more tests – More consistent quality results with TDD

S19	E. Arisholm et al. (2007)	Experiment	PP	Professional	295	– on the more complex system, the pair programmers had a 48% increase in the proportion of correct solutions – no significant differences in the time taken to solve the tasks correctly
S20	A. Begel and N. Nagappan (2008)	- Survey - Qualitative	PP	Professional	487	– (65.4%): pair programming produces higher quality code, – fewer bugs, spreading code understanding, – increased working time, scheduling problems, and personality conflicts.
S21	T. Bippet al. (2008)	Case Study	PP	Academic	100	– less time than solo programming. – more code knowledge, – higher quality code, – minor loss in efficiency.
S22	G. Canfora et al. (2007)	Experiment	PP	Professional	N/A	– quality improvement is higher than 15%, – increment of effort to complete the task.
S23	C. McDowell, et al. (2006)	Case Study	PP	Academic	554	– pairing students produce higher quality programs, are more confident in their work, and enjoy it more, – female programmers benefit from pair programming.
S24	J. Chao and G. Atli (2006)	- Survey - Experiment	PP	Professional Academic	60 58	– certain personality traits ensure higher quality code(i.e. Openminded and Responsible), – differences were statistically significant.
S25	J. T. Nosek (1998)	Experiment	PP	Professional	10	– the pairs produced higher quality solutions (readability and functionality), – increased morale (qualitative assessment).
S26	J. Nawrocki and A. Wojciechowski (2001)	Experiment	PP	Academic	21	– no difference in quality (lines of code and number of resubmissions due to defects in code), – no difference in the development time.
S27	L. Williams et al. (2000)	Experiment	PP	Academic	41	– increased quality (number of passed test cases) – (40 – 50%) less time than the individuals, but at increased cost
S28	C. McDowell et al. (2002)	Experiment	PP	Academic	313	– increased quality (functionality and readability) – better information and knowledge transfer
S29	L. Madeyski (2006)	Experiment	PP	Academic	188(132pp)	– external code quality (here the number of acceptance tests passed) is correlated with the feelgood factor, and in using a classic testing approach.
S30	H. Hulkko, and P. Abrahamsson (2005)	Case Study	PP	Professional	4-6	– contrasting results in the defect density (quality) – contrasting results in productivity
S31	K.M. Lui and K.C.C. Chan (2003)	Experiment	PP	Professional	15	– increased quality – better and faster designing of algorithms – increased productivity
S32	P. Sfetsos et al.	Experiment	PP	Academic	70	– heterogeneous personality groups are more effective (more acceptance tests passed, better design/code, increased velocity and better communication)
S33	M.M. Müller (2005)	Experiment	PP	Academic	38	– for same level of correctness, pair and solo programming have same cost. – for different level of correctness pair produces higher quality at the expense of increased cost.
S34	M.M. Müller (2007)	Experiment	PP	Academic	38 (42-projects)	– pair programming suitable for complex and challenging problems – less faulty expression defects for pairs
S35	K.M. Lui and K.C.C. Chan	Experiment	PP	Academic	40	– pair programming effectively helps developers solve unfamiliar programming problems – novice–novice pairs against novice solos are much more productive than expert–expert pairs against expert solos.

S36	T.H. DeClue (2003)	Experiment	PP	Academic	24	– increased quality (design/code) – improved quality of teamwork, communication skills, comprehension and learning
S37	B. Hanks et al. (2004)	Experiment	PP	Academic	N/A	– pairing students produce programs that are shorter and less complex – more confident in their work – more likely to complete CS1
S38	S. Xu and V. Rajlich (2006)	Case Study	PP+TDD+Refactoring	Academic	12	– higher quality (more test cases, cleaner code with higher cohesion, more reasonable number of methods)
S39	L. Layman et al. (2004)	Case Study	XP – practices	Professional	10	– 65% improvement in prerelease quality – 35% improvement in post-release quality – 46% increase in productivity
S40	S. Ilieva et al. (2004)	Case Study	XP – practices	Professional	4	– 13% fewer defects reported – 42% increase in productivity
S41	C.A.Wellington et al. (2005)	Experiment	XP – practices	Academic	16	– improved quality measurements – significantly greater code quality
S42	B. Tessem (2003)	Case Study	XP – practices PP+TDD	Academic	6	– pp leads to higher quality – better work estimation of the work size with planning game. – TFD leads to higher code/design quality
S43	G. Melnik, and F. Maurer (2005)	Mixed	XP – practices	Academic	240	– 76% suggested that XP improves the quality of code – 78% believe or strongly believe that using XP improves the productivity of small teams
S44	F. Macias et al. (2003)	Experiment	XP – practices	Academic	4-5	– no difference in either internal or external quality between the XP teams and the traditional teams – no difference in product size between the XP teams and the traditional teams
S45	S. Lee and H.S Yong (2010)	Case Study	Scrum	Professional		– distributed agile projects and Scrum: 30% improvement in the product quality – more customer satisfaction
S46	R. Moser et al. (2008)	Case Study	XP – practices Refactoring	Professional	4	– refactoring leads to higher quality and productivity – better results for small teams