ΑΛΕΞΑΝΔΡΕΙΟ Τ.Ε.Ι. ΘΕΣΣΑΛΟΝΙΚΗΣ
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Πτυχιακή εργασία

«Εμπειρικές μελέτες στις ευέλικτες μεθόδους: βιβλιογραφική έρευνα (μέχρι Φεβρουάριο 2010) για τις εμπειρικές μελέτες στις ευέλικτες μεθόδους και στον ακραίο προγραμματισμό.»

Του φοιτητή
Μανίκη Μιχάλη
Αρ. Μητρώου: 0117/28

Επιβλέπων καθηγητής
Σφέτσος Παναγιώτης

Θεσσαλονίκη 2010

Abstract:

Οι ευέλικτες μέθοδοι και οι πρακτικές τους είναι ένα αμφιλεγόμενο θέμα. Κάποιοι ερευνητές υποστηρίζουν ότι βοηθάνε τις εταιρίες λογισμικού και κάποιοι άλλοι ισχυρίζονται ότι δεν έχει κανένα αντίκτυπο ή ακόμα χειρότερα ότι είναι επιζήμιο. Αυτή η βιβλιογραφική έρευνα έχει ως στόχο να ερευνήσει τη τρέχων κατάσταση των ευέλικτων μεθόδων και να ερευνήσει αν μπορούν να βοηθήσουν τα έργα λογισμικού ή όχι. Θα ερευνηθεί η υπάρχων βιβλιογραφία μέχρι το Φεβρουάριο του 2010 και θα προσπαθήσει να απαντήσει μερικές από τις ερωτήσεις σχετικά με τις ευέλικτες μεθόδους, όπως το αν οι ευέλικτες μέθοδοι βοηθάνε εταιρίες λογισμικού καθώς και τα έργα τους καθώς επίσης αν μπορούν να βοηθήσουν τα πανεπιστήμια και τους φοιτητές.

Abstract:
Agile methods and practices are a controversial topic. Some practitioners suggest that it helps software companies and some claim that it has no effect or even worse it is damaging. This review will aim to identify the current state of agile methods and investigate whether they help software projects or not. It will search existing literature until February 2010 to try and answer some of the questions about agile methodologies, such as if agile methods help software companies and their projects and if they can also help universities and students.

Systematic literature review in empirical studies for agile methods and extreme programming

Manikis Michael

Alexander Technological Educational Institute of Thessaloniki, Greece

February, 2010

Contents

1. Introduction

There has been 9 years since 2001, when Agile Manifesto was formed and there is still much discussion about if agile methods can contribute to better and/or faster software development. Some are big supporters of agile methodologies and some think that traditional methods are more suited for software development. Researchers do experiments and studies to find out if agile methodologies can help software companies, by gathering statistics from successful and failed agile projects.

This systematic review aims to search and find empirical data and experiments, synthesize them and find out the effects that agile methods have on software projects, and if they can help to raise the probability of success for a project, as well as to provide a better communication between the developers.

It further aims to provide practitioners an insight view of a number of experiments that have been done for agile methods and give them information about the current state of agile methods. It also aims to help software industry by providing data about the effects that agile methods have on software projects and if they can effectively and successfully integrate agile methods into their software projects.

This review is organized in 7 sections. In section 2 there will be a brief theory of agile methods. In section 3 an analysis of the review method that was followed, section 4 contains the results for every primary study, section 5 will have a discussion of results and limitations of this review, section 6 contains the conclusions and finally section 7 contains the references of this systematic review.

2. Theory

A description of agile methods will be given first and then there will be a description pair programming and test-driven development which are some of the core practices of agile methodologies. Also there will be a summary of the main points of criticism against agile methods. Lastly there will be a description of the objectives and the research questions of this review.

2.1 Agile Methods

In early 2001, seventeen of the agile proponents met, discussed and formed the Agile Manifesto. Many representatives from most agile methods, such as Extreme Programming, SCRUM, DSDM, Crystal, FDD and others attended. They stated that there is a need to escape from the heavyweight, documentation-driven software development. The Agile Manifesto [55] reads as this:

- Individuals and Interactions over process and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan.

“That is while there is a value in the items of the right, we value the items on the left more.”

Agile means [56] “deliver quickly, change quickly, change often”.

There are many agile methods with different characteristics and practices but they have some common ones, like iterative development, communication, as well as the reduction of documentation to the absolutely necessary. Those characteristics help developers adapt to change easier and take decisions faster and respond quickly. Also documentation reduction helps the project to deliver quicker [57].

Extreme programming is probably one of the most popular agile methods. The 12 rules of extreme programming are the following:

1) The planning game.
2) Small releases.
3) Metaphor.
4) Simple design.

5) Tests.

6) Refactoring.

7) Pair programming.

8) Continuous integration.

9) Collective ownership.

10) On-site customer.

11) 40-hour weeks.

12) Open workspace.

In order to see the true benefits of Extreme Programming someone should practice all of its practices [57].

Scrum is the second most popular agile method. Scrum is described [58] as a process that “accepts that the development process is unpredictable”. Iterations of scrum are split as follows:

1) Pre-sprint planning

2) Sprint.

3) Post-sprint meeting.

Alistair Cockburn in 1990 wanted a method that would provide better communication. The result of his work was the Crystal Methods. Cockburn explains [56] that increased communication can reduce unnecessary documentation and the more iterative the development is the less you really need documentation. The most agile crystal method is Crystal Clear, followed by Crystal Yellow, Crystal Orange, etc. As you move away from crystal clear the rules of the Crystal Method increase thus the method becomes less agile.

Feature-Driven Development was developed by Jeff DeLuca and Peter Coad. It all started when DeLuca take on a failing project, the previous contractor had wrote too much documentation but no delivered version. DeLuca hired Coad and they both developed it successfully and the resulting method they used was feature-driven development (FDD) [57].

Lean Development (LD) was developed by Bob Charette, and is based on the rules and practices of Lean Production found in Toyota in the 1980s [9].

Dynamic Systems Development consists of six stages [59]: Pre-project, Feasibility Study, Business Study, Functional Model Iteration, Design and Build Iteration, Implementation, and Post-project.

2.2 Pair Programming

Pair Programming is an agile practice in which two programmers work together at a single computer. One writes the code (driver) and the other reviews it (navigator).

Benefits of pair programming include the following [60].

Design quality: Program has fewer bugs as it is tested by two developers and one is specifically assigned to watch out for possible bugs as the other writes the code.

Reduced cost of development: Fewer bugs mean less development costs especially if they are caught early before they are more difficult and costly to fix.

Learning and training: Because programmers work in pairs, knowledge can easily be shared. One programmer can learn from the other and novice programmers can quickly catch up with more experienced ones thus offering better overall software quality.

Overcoming difficult problems: Problems can be more faster and more easily solved when programmers work and they search for a solution together.

Improved morale: Some programmers report greater satisfaction when they work together.

Decreased management risk: If one programmer leaves the team, the others can quickly catch up because they all share and work on the same code.

Increased discipline: When programmers work in pairs they are less likely to spend time not working and doing something else.

Fewer interruptions: People don't interrupt so much a pair than they interrupt a single programmer working alone.

Decreased risk of RSI: Risk of physical stress to the programmers by working too much is reduced since they can switch positions and rest while the other pair types.

Some drawbacks include [60]:

Work preference: Some people just cant work with others and they prefer to work alone.

Intimidation: A less skilled programmer can be less confident if he works with a more skilled developer and he might not contribute to the code and to the pair.

Tutoring: Skilled programmers may find boring to work with a less skilled programmer because they have to teach him techniques that he doesn't already know, thus spending time from the actual code writing.

Personality conflicts: The programmer pairs may have difficulty working together and it may be even worse and less productive than if they worked alone.

There are also some empirical evidence [63] that pair programming can help with the programming courses in universities in the CS curriculum. Another study [68] shows that students don't really care about the personality of their programmer pair as long as they have similar programming experience. Despite that there are studies [69] that show no significant relation between students working in pairs and students working solo.

2.3 Test-Driven Development

TDD [60 is an agile practice that focuses on the creation of unit tests to test the code. The process goes as follows: First the developer adds a test for a future feature, then produces code to implement the feature and that passes the test and lastly refactors the new code.

The sequence of TDD as described in [61] is the following:

1) Add a test

A test is added before the new feature is implemented. The test will fail because the feature doesn't yet exist but it will help the tester refine the requirements before and not after writing the actual feature.

2) Run all tests and see if the new one fails

All tests are run, including the new one. New test must fails and this guarantees that it is working properly (since the new feature is not added yet).

3) Write the code

The feature is written in a way that will make the test successful. The new code might not be very good and may have bugs but this doesn't matter because it will be improved in later iterations. This new code must be designed to implement that specific feature and only that.

4) Run the automated tests

All tests are now run, and it is being checked if the code passes all tests. If all tests pass developers start refactoring.

5) Refactor code

Now the code is being improved and with the use of previous tests programmers can be assured that their refactoring won't cause more problems and that the feature implemented before are still working as intended.

This circle continuous as new tests and functionality are being added to the code.

2.4 Agile Criticism

Despite the fact that there are people that support agile methods, there are also people that criticize them for being ineffective and damaging for a software project.

The main points of criticism include the following [9]:

• Agile development exists since 1960 and if it was good it would have shown its benefits.

• The use of a very small amount of documentation will create a faulty design.

• No research have proved many of the claims about agile practices.

• XP practices are good in theory but they cant and they never applied successfully in practice.

• Agile methods might be useful for small teams and projects but for large scale projects it just cant help.

2.5 Review Objectives

There seems to be much interested by companies on learning about agile methods and possibly try them. A survey [56] conducted in the USA and Europe on 2000 shows that 14% of companies use agile methods and also 49% of companies that are aware of agile methods are interested in learning and adopting them. A more recent survey on 2007 shows that the percent has raised to 69% for those that use agile methods [70]. A survey conducted on 2008 raised that percent even more. 95% of companies report that they use at least to some extent an agile method with a 51% reporting that more than 50% of companies projects use agile methods [22]. Rajlich [62] describes agile methods as “a new paradigm” and specifically states that developers were used to old practices and they wouldn't care for anything else. Now with those new practices there isn't enough research yet and it is necessary in order to solve some problems.”

Despite the fact that there is much interest in agile methods not many systematic literature reviews are being published, and its difficult for companies that do want to adopt agile methods to search for reviews in order to find out if agile methods will be successful inside their company and for their software project. With this systematic review there is an aim to give a better insight into the available empirical data of experiments, case/field studies, and surveys/reviews on agile methods and to help companies make decisions that will aid them choose and adopt and agile methodology. Also this review will help researchers as well as students that are interested in agile methods to stay up to date to the current state of agile methods.

This systematic review will try to answer the following research questions:

- Can agile methodologies and its practices help software companies develop software projects more effectively than traditional methods?
- Can agile methodologies help universities with teaching computer science courses to students?

This review will also try to qualify the primary studies so it can examine the quality of the answers and the need for possible future reviews on the subject.

3. Review Method

The purpose of a systematic review is to search and find all available research documents that can help with answering the research questions that have been stated before. This review followed the steps below to conduct the research.

1) Identify the review objectives and then establish a review protocol and lay down the research questions that this review will try to answer.
2) Establish the inclusion and exclusion criteria, the quality assessment criteria.
3) Search the literature that will help answer those questions.
4) Assess the primary studies in order to qualify them based on the criteria created.
5) Extract the data from the primary studies along with the quality assessment of each of them.
6) Synthesize the extracted data, compose the results and conclude.

In the paragraphs that follow there is an analysis of those stages in detail to provide a better view of the methods and tools used for each one.

3.1 The Review Protocol

The review protocol is the framework of the review. It contains the research questions that this review will try to answer, the way the search process will be conducted as well as the quality criteria that the primary studies must pass in order to be accepted. For the developing of the protocol the guidelines as described in Kitchenham's study [65] were followed.

3.2 Research Questions

This review will try to explore the success rate of agile methods on companies that have adopted them and which factors influence this success. The review will also investigate existing literature to find out what kind of companies are best suited to adopt agile methodologies successfully, what are the problems that might emerge and if agile methodologies can be effectively adopted on large scale projects and companies. Finally it will be investigated the effect that agile methods have on teaching undergraduate students. What is the response of the students and if agile methods can help teachers teach students more effectively.

The study will include studies from students and professionals and no exclusions will be performed based on population type.

The research questions are:

• Can agile methodologies and its practices help software companies develop software projects more effectively than traditional methods?

• Can agile methodologies help universities with teaching computer science courses to students?

3.3 Identifying literature

For identifying the studies that will help answer the research questions the search strings that will be used as a search term on digital libraries were formed first. The study is focusing on agile methods and also on the two most famous agile practices, pair programming and test driven development (TDD). Only empirical data were included and no theoretical studies or expert opinions were accepted. The search terms that were used were the following:
'test first', 'test driven', 'pair programming', 'agile', 'empirical', 'experiment', 'survey', 'review', 'case study', 'field study'.

Those terms were then combined using AND and OR Boolean operators intot he following search string:
(test first AND empirical) OR (test first AND experiment) OR (test first AND survey) OR (test first AND review) OR (test first AND case study) OR (test first AND field study) OR (test first) OR (agile AND experiment) OR (agile AND survey) OR (agile AND review) OR (agile AND case study) OR (agile AND field study) OR (agile AND empirical) OR (agile) OR (test driven AND empirical) OR (test driven AND experiment) OR (test driven AND survey) OR (test driven AND review) OR (test driven AND case study) OR (test driven AND field study) OR (test driven) OR (pair programming AND empirical) OR (pair programming AND experiment) OR (pair programming AND survey) OR (pair programming AND review) OR (pair

programming AND case study) OR (pair programming AND field study) OR (pair programming). The review didn't include expert opinions, lessons learned, theoretical studies, panels, summaries, interviews and it was based entirely on empirical data, experiments and surveys/reviews.

After the composition of the search strings, the latter were inserted on digital databases and online journal papers to identify the useful studies for this review. The list of databases, journals, conferences and digital libraries that were searched is the following:

Journals and Conferences:

- Information and Software Technology (IST)
- Journal of Systems and Software (JSS)
- IEEE Transactions on Software Engineering (TSE)
- IEEE Software (IEEE SW)
- Communications of the ACM (CACM)
- ACM Computer Surveys (ACM Sur)
- ACM Transactions on Software Engineering Methodologies (TOSEM)
- Software Practice and Experience (SPE)
- Empirical Software Engineering Journal (EMSE)
- IET Software (IET SW)
- Proceedings International Conference on Software Engineering (ICSE)
- Proceedings International Symposium of Software Metrics (Metrics)
- Proceedings International Symposium on Empirical Software Engineering (ISESE)

Digital Libraries:

- ACM Digital Library (http://portal.acm.org)
- IEEEXplore (http://ieeexplore.ieee.org)
- CiteseerX Library (http://citeseerx.ist.psu.edu)
- ScienceDirect (http://www.sciencedirect.com)
- Wiley InterScience (http://www3.interscience.wiley.com)

- SpringerLink (http://www.springerlink.com)

3.4 Inclusion and Exclusion Criteria

After applying the search strings on the databases, the resulting papers were evaluated based on their title and abstracts. Papers published until February 2010 were included. This review only focused on reviews, case/field studies, surveys, and experiments that presented empirical data on the field of agile methods, pair programming and TDD. Papers that didn't present empirical data such as lessons learned, panel sessions, summaries, expert opinions that were based on personal opinion with no empirical data to back it up, as well as papers that discussed and explained agile methods and its practices in theory, were all excluded from the search. A problem also encountered in Dybå (2008) [9], was noted here also, some articles were not clearly stating on their abstracts that they were empirical studies and some others had misleading titles. All those papers were included for further quality assessment after evaluating more data on the paper such as its conclusions, if they presented data that they might be useful. There was no exclusions on papers based on population type, both student and professional authors' papers were included as long as they have been published on a well-known journal or conference. Papers that did present empirical data but were not published anywhere were excluded from the search. The search process only identified articles written on english. After this initial quality assessment of papers, the result was 123 papers. Those papers will then be assessed for their quality again based on more details. All those papers were then inserted into EndNote X3 and they were organized there. While the quality assessment continues the papers in EndNote were grouped and organized depending if they are useful or not.
Figure 1 shows the type of each of the final 55 studies after the final quality assessment.

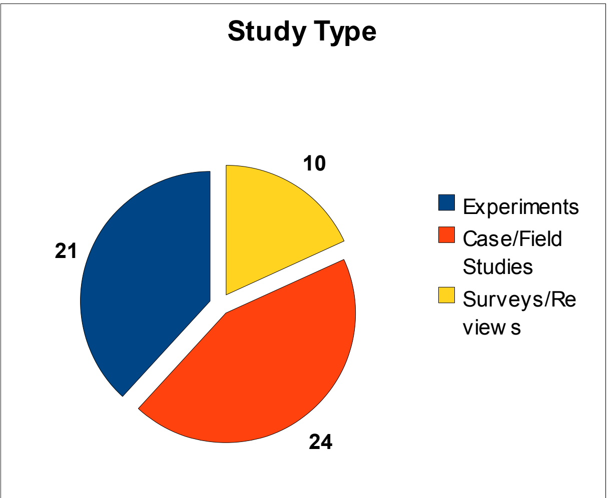

Figure 1

Figure 2 shows the number of primary studies after each stage of screening process.

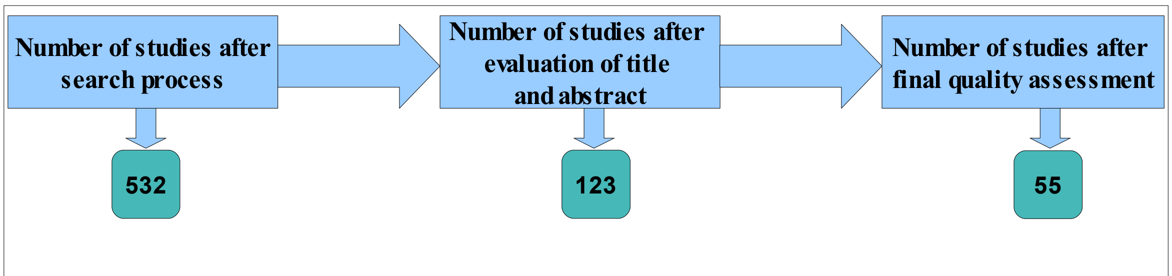
Number of studies after search process
Number of studies after evaluation of title and abstract
Number of studies after final quality assessment
532
123
55

Figure 2

The inclusion and exclusion criteria as mentioned in the review protocol are the following:

Inclusion Criteria

- [Q1] Systematic Literature Reviews, Surveys, Experiments, Case and Field Studies that are clearly identified as empirical researches containing empirical data and/or statistical data on the field of agile software development and its practices.
- Papers of both students and professionals were included as long as they have been published in a well-known and broadly recognized journal and/or conference.
- [Q2] Papers that can help with the research on the research questions of this review.
- Papers that passed the minimum quality criteria (mentioned in Quality Assessment)

Exclusion Criteria

- Papers that haven't been published in a journal.
- Papers that didn't contain empirical data and were mostly theoretical researches and/or expert opinions.
- Papers that discuss the agile software development process and its subcategories in theory.

3.5 Quality Assessment

Those 123 studies that passed the initial screening process, where then reevaluated in more detail using the criteria suggested by Critical Appraisal Skills Progamme (CASP) especially those for qualitative research [66], and by "principles of good practice for conducting an empirical research in the field of software engineering" [67]. Those 8 criteria fall into 3 main categories which are the following:

- *Rigour:* does the study follow a specific explained approach in the implementation of the various methods used in the study?
- *Credibility:* does the author discuss possible bias and findings and are the findings useful for the purposes of the study?
- *Relevance:* Are the findings useful for software companies and/or researchers?

By further analyzing those 3 main categories, the following 5 criteria are identified firstly for the rigour of each study. Specifically:

- [Q3] The research method that was followed was explained as to why the specific one was used.
- [Q4] There is a description as to why the specific sample was selected and with what criteria.
- [Q5] A control group was used to compare the results of the study.
- [Q6] There is a description of the data collection methods, as to how the data were collected and why this specific method was used.
- [Q7] Data analysis methods were described concerning as to why those methods were chosen, how the data were selected and if they are enough to answer the questions of the study.

Following the rigour, the credibility of each research is analyzed and 2 more quality criteria emerge. Those criteria are:

- [Q8] Does the researcher identified his possible bias and the role he might have played to the research?
- [Q9] The results were discussed, they were identified if they answer the

research questions and if the authors discuss the strength of their results.

Lastly the relevance of each research is analyzed by using 1 criteria. This criteria is the following:

- [Q10] Are the findings and results of the study useful and worthy for companies and/or scientific research?

Quality criteria [Q1] and [Q2], as mentioned in the inclusion criteria, were the only ones that were able to exclude a study from the research, the other criteria form the quality of each study and even if some studies don't pass those criteria ([Q3]-[Q10]) they still are included in the study. Those 10 criteria form the quality assessment of each study (Appendix B) and will ensure that each study that is included in this literature review will have a valuable contribution and will help answer the research questions. Depending on the quality of the papers it will also be identified if there is a need for conducting further research to get better results. In each of those criteria there were two answers, (yes) or (no). The studies that answered (yes) get 1 point for that criteria and if they answered (no) they get 0 points. Those points then add up and form the final quality score of each study.

The search process as well as the quality assessment of each paper was conducted by one person (the author of this review).

3.6 Data Extraction

Data from each of the 55 studies that were selected and passed the quality criteria were then extracted by using an extraction form. The data that was extracted were the necessary data to conduct the research and answer the research questions. Some articles didn't specifically state some of the data that were needed to complete the extraction form, so some of them were left blank.

All the data that were extracted were organized in OpenOffice Calc. Data extraction process was also conducted by one person.

3.7 Synthesize the data and conclude

Finally the data that were extracted at the previous stage, were synthesized and the results from all of the studies were documented. Results from each study were compared and organized in Calc tables and at the end the conclusions of the study were drawn. A summary of the extracted data of each study can also be found in tables throughout this review.

4. Results

After the final application of the quality assessment criteria, the final studies were chosen that were useful for the purposes of this review. Those studies differ from university studies to studies performed on companies. The studies cover a range of topics from agile methods to pair programing and TDD. A presentation of those study will follow. The presentation of the studies is organized based on their research type. First there will be an analysis of experiments, next case studies and field studies will be analyzed and finally surveys and reviews will be presented.

On the section that follows there will be a brief description of all the primary studies and their characteristics, next there will be a discussion of all the studies based on their type and finally the conclusions are drawn.

4.1 Analysis of primary studies

4.1.1 Experiments

As mentioned the primary studies will be divided and discussed between 3 groups. Firstly there will be a description of the experiments. Table 1 contains a summary of the experiments.

Table 1

ID	Author	Type of Study	Research Field	Research Environment	Population Size	Results
3	Syed-Abdullah, S. et. al.	Experiment	XP	Professional	75 Students	XP team experienced higher overall w ell being w hen project requirements w ere uncertain.
7	Misra, S. et. al.	Experiment	Agile Methods	Professional	N/A	Customer involvement, quick decision time and corporate culture, among others, are importnant success factors for adapting agile methods.
17	Alshayeb, M. et. al.	Experiment	Agile Methods	Professional	N/A	SDI can be used w ith agile methods. New design effort might cause project instability according to SDI.
19	Ferreira, C. et. al.	Experiment	Agile Methods	Professional	59 Projects	The more agile methods and practices used the more stakeholder satisfaction. Stakeholder satisfaction is important and should be taken into consideration by developers.

20	Alshayeb, M. et. al.	Experiment	Agile Methods	Professional	2 Projects	New design effort doesn't cause more error fix and refactoring and can be safely applied.
23	Huang, L. et. al.	Experiment	Test Driven Development	Student	N/A	70% increased prodictivity for test first team. No other significant differences.
26	Madeyski, L.	Experiment	Test Driven Development	Student	N/A	MSI and BC show ed no difference betw een test first and test last.
27	Canfora, G. et. al.	Experiment	Test Driven Development	Professional	28 Programmers	More time needed for TDD, possible increased quality. Better cost estimation w ith TDD.
28	Janzen, D. et. al.	Experiment	Test Driven Development	Mixed	N/A	Mature developers are more w illing to use TDD. Most developers think TDD is useful but it might be difficult to implement.
31	George, B. et. al.	Experiment	Test Driven Development	Professional	24 Programmers	TDD passed 28% more test cases, w hile requiring 18% more time. 80% find TDD effective and lack up-front design doesn't cause problems.
32	Sfetsos, P. et. al.	Experiment	Pair Programming	Student	70 Students	Heterogeneus personality groups are more effective for pair programming.
34	Braught, G. et. al.	Experiment	Pair Programming	Student	176 Students	Pair programming helps students w ith low er SAT scores perform better. Instructor differences w ere mitigated.
36	Madeyski, L.	Experiment	Pair Programming	Student	N/A	No difference in BC and MSI scores. Further research might be necessary.
37	Müller, M. M.	Experiment	Pair Programming	Student	18 Students	Pair design phase is feasible, no increase in cost. Same number of errors detected.
39	Müller, M. M.	Experiment	Pair Programming	Student	42 Projects	Less faulty expression defects for pairs. Pair programming suitable for complex and challenging problems.
41	Canfora, G. et. al.	Experiment	Pair Programming	Professional	N/A	Pair design causes better quality prediction but low er development time prediction.
42	Choi, K. S. et. al.	Experiment	Pair Programming	Student	128 Students	Diverse MBTI type groups best stuited for pair programming groups.
43	Müller, M. M.	Experiment	Pair Programming	Student	38 Students	For same level of correctness pair and solo programming have same cost. For different level of correctness pair produces higher quality at the exense of increased cost.
44	Thomas, L. et. al.	Experiment	Pair Programming	Student	60 Students	Low confident students enjoy pair programming the most. High confident students dont enjoy pair programming and they prefer to group w ith high confident students.
49	Desai, C. et. al.	Experiment	Teaching w ith Test Driven Development	Student	83 Students	No increase in product quality. Better learning how to test code. No increase in instructor effort.
52	Hanks, B.	Experiment	Teaching w ith Pair Programming	Student	30 Students	Pair programmers do make same mistakes as solo programmers but they are able to solve most of them alone. Increased student confidence.

Syed-Abdullah et. al. (2006) [3] examined 2 groups of student that worked for real industrial projects. The purpose of the study was to examine the effect of agile methods (specifically XP) on the well being of software developers. First team was the control group using a method previously discussed in the computer courses (Discovery method), second group was using XP. The experiment examined the anxiety, contentment, depression and enthusiasm of the software developers through 3 different software project. Two of the projects were more stable and the last one was more unpredictable. Results showed that anxiety levels were higher for the XP team for the stable projects but for the unstable project while XP team started with higher anxiety, the increase through the life-cycle of the project wasn't big, while the Discovery team experienced high increase of anxiety especially towards the end of the project. It should be noted that a possible anxiety factor for XP teams might have been the responsibility to learn a new developing technique while Discovery team has already learned the method from previous CS courses.

Despite these, no significant difference was found between the 2 teams. Contentment levels were higher for XP team at the start of the project, both teams experienced a drop on their contentment levels with XP team experiencing higher decrease. No significant difference was also reported here. Depression levels for XP team were higher in the stable projects, but XP team had much lower depression levels in the unpredictable project. No significant difference was also reported here. Lastly enthusiasm levels were higher for XP team in all projects against the Discovery team. Significant statistical difference was reported here. Syed-Abdullah et. al. also tested if a higher number of XP practices caused a better well being (including anxiety, contentment, depression, enthusiasm). Significant difference was reported here. It seems that the more XP practices used, the better the well being of the developers becomes.
Syed-Abdullah et. al. concludes that the overall well being of the XP team (except for contentment) was better but only when the requirements of the project are uncertain.

Misra, S. et. al. (2009) [7] conducted an experiment by questioning a large number of practitioners using agile practices and belonging in different sectors of industry. The aim of the survey was to identify some success factors in adopting agile methods. Greater customer satisfaction, collaboration and commitment, quicker decision time, better corporate culture relates to agile projects, more qualitative controls on projects, better personal characteristics of team members, the more favorable the societal culture is for agile projects, the more the environment supports continuously learning and informal training, all these factors lead to a better success of the agile project.
The survey identified some factors that have no effect to the success of the agile project, such as close location of project teams, smaller team size, more in-formalized plans, more technically competent team members, more communication and negotiation. They also found some new possible success factors for an agile project. Those factors include, learning from failure, timing issues, other team characteristics and use of tools. Although Misra, S. et. al. suggest additional research to examine those factors.

Alshayeb, M. et. al. (2005) [17] made an empirical study to test if the System Design Instability metric (SDI) can be used in an agile environment to re-plan software projects. SDI is used the determine the instability of the system. It measures the domain abstraction and design, and if the system is stable then classes, hierarchy, class names etc wont change much. Their results show that SDI can be used in an agile environment to estimate and re-plan software projects. They noticed that traditional methodology projects as well as XP projects follow the same pattern. New design-effort positively correlates to the SDI thus showing possible increase in instability of the system. Surprisingly, refactoring and error-fix were negatively correlated to the SDI. They explain this result by stating that refactoring and error-fix indicate a stable design at the domain abstraction level which is the level that SDI measures.

Ferreira, C. et. al. (2008) [19] conducted an empirical study of 59 south African software projects to explore the connection of agile methods and stakeholder satisfaction. All of their hypothesis were confirmed. In detail, iterative development keeps developers on the right direction and satisfies stakeholders. Continuous integration, TDD and collective code ownership are also important and successful execution can lead to a better system. Regular feedback also helps recognize requirements which in the end will satisfy stakeholder. They also note that the more satisfied the stakeholders are during the development process, the more satisfied they will be with the end product when it will be released. Any stakeholder dissatisfaction needs to be taken seriously by the developers because it might lead to problems for system acceptance. In conclusion they propose especially small companies to adopt those agile methodologies and to balance traditional and agile practices, leading to a positive effect on the overall project.

Alshayeb, M. et. al. (2006) [20] in their study investigated a possible connection between the new design effort of XP and error-fix and refactoring. Particularly, the more new-design effort that was performed in the system, the less refactoring and error-fix was needed. They also found that error-fix is related to the number of days spent on a story. The more days spent, the more error-fix will be performed. However new-design effort and refactoring doesn't seem to relate with the number of days spent on a story. The study shows that new-design can be safely applied to the system and it wont cause too much error-fix and refactoring efforts.

Huang, L. et. al. (2009) [23] performed an experiment to test the differences between 2 groups of students composed of different teams. First group was using test first approach and second group was using test last approach (traditional approach). Results show that Test First group spent more time on testing than Test Last group. But the experiment found no significant statistical difference between the 2 groups in terms of software quality. Also Test First group wasn't more productive than Test Last group although on average Test First group productivity was 70% higher. Test First teams had more varied productivity than Test Last teams that were more linear. Experiment concludes that further research, especially in industry area, is needed to verify those results because there were some limitations.

Madeyski, L. et. al. (2010) [26] examines the impact of Test First (TF) on branch coverage and mutation score indicator. Branch coverage tests if the unit tests examine a large portion of the code rather than a small amount. Mutation score indicator injects mutants in the code to examine the effectiveness of discovering them by the unit tests used. The authors didn't find a significant difference between TF teams and Test Last (TL) teams concerning branch coverage and mutation score. They state that they are unable to explain why those tests don't show the positive effects of TF because it is the first time that mutation score indicator is examined using TF approach and further research may be necessary, although they suggest that if more professional developers were used results may have been different.

Canfora, G. et. al. (2006) [27] presents the results of an experiment conducted in a company to compare TDD and TAC (Testing After Coding). The result shows that TDD requires more time spent than TAC although this might be compensate by the possible increased code quality. Unit tests of TDD, agreeing with [26], don't seem to be more accurate and precise than TAC unit tests. Finally TDD provides better predictability which in turn leads to better cost estimation, something that some companies might find useful.

Janzen, D. et. al. (2007) [28] reports on an experiment for the purpose of comparing the acceptance of TDD. The sample included two groups, beginner programmers and mature programmers. Survey results show that mature programmers choose TDD much more than beginner programmers. Among the beginner programmers those that have actually tried TDD are more willing to accept it. Same trend appears on mature programmers. Its interesting that TDD benefits appear to be recognized by more participants than those that actually choose TDD, and comments on the surveys show that many participants do recognize TDD benefits but perceive TDD as more difficult and different than what they are used to.

George, B. et. al. (2004) [31] conducted an experiment between 2 groups of 24 professional pair programmers. First was using TDD and second group was using waterfall-like approach. TDD group yielded better code quality, TDD program passed 18% more test cases. Additionally as previous studies state, TDD programmers spent 18% more time than control group programmers. Although the authors note here that control group programmers skipped some of their tests and this might lead that waterfall-like approaches don't encourage testing. Also survey results showed that 80% of the programmers consider TDD as an effective approach and they don't think that lack of up-front design is a problem because TDD facilitates simpler design.

Sfetsos, P. et. al. (2009) [32] presents an experiment conducted between 70 undergraduate students in order to investigate the effect of pair personalities on pair effectiveness and pair collaboration-viability. Specifically they compared pair programming groups with homogeneous personalities and heterogeneous personalities. Results showed that heterogeneous personality groups had better performance and collaboration validity. Communication, design code correctness and design velocity were much better for heterogeneous personality groups. Pair communication for heterogeneous pairs was also better that homogeneous pairs and that led to higher design and code correctness. They conclude that further investigation might be necessary to support those findings.

Braught, G. et. al. (2008) [34] reports on an experiment to investigate the effect of pair programming on programming skill. The authors used students from Dickinson college to see if pair programming helps students with lower programming skills. Results indicate that pair programming does help students with lower SAT scores to achieve better scores. Pair programming was also helpful for all kinds of students to assist them in order to complete the courses successfully. Pair programming also helped to mitigate the differences between different instructors.

Madeyski, L. et. al. (2008) [36] examined the effect of pair programming in thoroughness and fault detection effectiveness on unit tests. Using branch coverage (BC) and mutation score indicator (MSI) metrics they examined 2 groups of students at the Wroclaw University of Technology. One group was using pair programming and the other solo programming. The results indicate that the BC and MSI scores didn't have significant difference between the 2 groups. After doing a more selective analysis by removing a number of projects the authors again didn't find significant difference on the scores between the different groups. The authors conclude that further research is necessary especially research that will focus on larger more complex projects that have more chance to benefit from pair programming.

Müller, M. et. al. (2006) [37] performs an experiment to examine the cost of pair design phase on pair programming and solo programming. The authors split the pair programming phase into 2 different phases, pair design phase and pair implementation or solo implementation. Pairs are created for pair design phase and they either split for solo implementation or continue for pair implementation phase. Results show that if programs of similar correctness level have to be developed then the costs are the same for solo or pair programming. Also the authors couldn't reject the hypothesis concerning the number of errors of both groups. Both groups had similar number of errors, and only different kind of errors, although the authors state that this might be affected by the small data set. Authors conclude that pair design phase might be an alternative to the pair programming process.

Müller, M. et. al. (2007) [39] in their experiment examine 42 programs developed by students consisted of both pairs and solo programmers to identify if pair programmers make different kind of mistakes than solo programmers. The results indicate that pair and solo programmers make the same algorithmic mistakes but pair programmers make less faulty expression defects. Also it seems that for complex and challenging problems, such as a technological problem where the solution is easy to be identified then a pair of programmers might be a better choice, offering reduced probability of failure and not necessarily doubles the personnel cost.

Canfora, G. et. al. (2007) [41] presents an experiment performed with professional software developers in a Spanish company. The authors investigated the effects of pairs when they are applied not in the coding phase but in the designing phase. Pair designing was found out that it improves quality but it slows down the task. It seems that pair designing is less efficient that pair programming. In conclusion with pair designing it is more easy to predict the quality but it becomes more difficult to predict the development time.

Choi, K. S. et. al. (2008) [42] explores, in an experiment, the impact of personality on pair programming. The authors used a group of 68 undergraduate students and 60 master's degree students and split them into 3 groups based in their Myers–Briggs Type Indicator (MBTI) type. One group had students who were alike in MBTI, in the second group students were opposite, and in the third group students were diverse (partially alike and partially opposite). Results show that the productivity level was much higher for the group of students that had a diverse type of MBTI. Their differences provide a “checks and balances” system that helps them stay on track and find solutions more easily and faster. While their similarities help them hide their differences and gives them a greater sense of compatibility. Completely opposite MBTI type group may yield a better end product but it had lower productivity than diverse group although still much better than completely alike group, which it seems that is not suitable for pair programming as completely alike personalities lose too much productivity.

Müller, M. et. al. (2005) [43] conducted an experiment between 38 students at the University of Karlsruhe. The authors investigated the difference, especially in development cost, between pair programming and solo programming with the added factor of peer review. Results indicate that both techniques end up having the same development cost if the same level of correctness is needed for the project. If different level of correctness is taken into account then pair programming produces programs of better quality at the expense of higher costs than solo programming. Authors conclude that further research is necessary, firstly by using professional programmers and not students and secondly by using a larger test group to obtain statistic significant results.

Thomas, L. et. al. (2003) [44] examined the effect of attitude on pair programming. 60 students were divided into pairs and then further divided into attitude groups. The authors noted here that the ending groups were too small and statistical significance was not easily achieved. The attitude difference was a scale between code-warriors (very confident students about their programming skills) and code-a-phobes (not at all confident students about their programming skills). Results showed that, overall, students enjoyed pair programming and believed that it helped them. Further analyzing the results shows that less confident students enjoy pair programming the most and very confident students like pair programming the least of all. Very confident students like pair programming even less if they are grouped with different attitude students and if they have to pair they prefer pairing with same attitude level (confident) students.

Desai, C. et. al. (2009) [49] reports on an experiment that was taken at a University using students of CS1/CS2 classes along with a control group. The authors concluded that TDD didn't greatly affect student's product quality but it does help them to learn how to test their code. On the other hand instructor's effort didn't increase by the extra introduction of TDD into the curriculum.

Hanks, B. et. al. (2008) [52] examined students at Fort Lewis College to find out if solo programmers make the same mistakes as pair programmers. Results were positive, pair programmers do make the same mistakes as solo programmers, with the difference that pair programmers are able to solve most of those problems, especially low-level problems, alone and thus not requiring assistance. This ends up increasing student confidence.

4.1.2 Case / Field Studies

In the following paragraphs there will be a description of the case and field studies. Table 2 shows a summary of the details of those studies.

Table 2

ID	Author	Type of Study	Research Field	Research Environment	Population Size	Results
1	Tolfo, C. et. al.	Case/Field Study	Agile Methods	Professional	3 Companies	1 of 3 Companies adapted succesfully. Organizational culture problems.
2	Tolfo, C. et. al.	Case/Field Study	XP	Professional	6 Companies	Organizational culture should be taken into account w hen adapting XP.
5	Petersen, K. et. al.	Case/Field Study	Agile Methods	Professional	N/A	Agile methods have positive as w ell as negative effects that need to be considered w hen adapting.
6	Moe, N. et. al.	Case/Field Study	Scrum	Professional	16 Programmers	Scrum adaption had problems w ith shared leadership. Agile methods need to give advices on shared leadership.
8	Lee, S. et. al.	Case/Field Study	Scrum	Professional	N/A	Distributed agile projects are feasible. 30% improvement in productivity after adapting Scrum.
10	Layman, L. et. al.	Case/Field Study	XP	Professional	N/A	Low er defect density pre- and post-release. Productivity w as same.
11	Layman, L. et. al.	Case/Field Study	XP	Professional	N/A	Distributed agile project w as successful. Communication is critical in such a project.
15	Hanssen, G. et. al.	Case/Field Study	Agile Methods	Professional	80 Employees	Customer engagement very important. Visibility of project increased.
16	Germain, É. et. al.	Case/Field Study	XP	Student	N/A	No significant effort difference.
18	Fogelström, N. et. al.	Case/Field Study	Agile Methods	Professional	N/A	Overall agile methods cant be used w ith MDPD projects, but certain practices can be used.
21	Lan, C.	Case/Field Study	XP	Professional	22 Programmers	Agile practices can benefit large scale projects w ith a few changes. Up-front should be used despite being discouraged in agile.
24	Laurie, W.	Case/Field Study	Test Driven Development	Professional	N/A	40% low er defect density. Less time debugging. Risk minimized. No producitivity decrase due to testing.
25	Maximilien, E. et. al.	Case/Field Study	Test Driven Development	Professional	N/A	50% low er defect rate. No productivity decrease.

29	Bhat, T. et. al.	Case/Field Study	Test Driven Development	Professional	N/A	TDD requires more time but produces code w ith better quality.
30	Damm, L.-O. et. al.	Case/Field Study	Test Driven Development	Professional	2 Projects	Project cost measured by ROI w as 5-6% less. Fault cost measured by AFC w as 30% less. Lead time decreased. Possibility for more positive effects.
35	Bipp, T. et. al.	Case/Field Study	Pair Programming	Student	100 Students	Pairs used time more efficiantly requiring less time than solo. More code know ledge, higher quality code. Minor loss in efficiency.
38	Hulkko, H. et. al.	Case/Field Study	Pair Programming	Professional	4 Projects	Same defect number for solo and pair programming. Pair programming more suitable for finding mistakes on small or complex code.
45	Laurie, W.	Case/Field Study	Pair Programming	Student	41 Students	Higher quality code in less time for pairs. Pair pressure has a positive effect.
46	Melis, M. et. al.	Case/Field Study	Test Driven Development / Pair Programming	Simulation	4 Projects	Pair programming and test driven development overall decrease defects, KLOCs and increase w orkload. Same quality w ithout PP and TDD increases w orkload very much.
47	Chong, J. et. al.	Case/Field Study	Pair Programming	Professional	2 Companies	Pairs more effective w hen they sw itched 'driver' and 'navigator' positions. Same skilled programmers more effective for pairing. Early pair sw itching is recommended but not later.
48	Proulx, V. K.	Case/Field Study	Teaching w ith Test Driven Development	Student	N/A	Better performance and low er chance for failure. More appreciation from companies.
50	Simon, B. et. al.	Case/Field Study	Teaching w ith Pair Programming	Student	13 Students	Solo programming w as found to be more lonely and stressful w hen encountered difficulties. Faster problem solution for pairs. Pair scheduling w as a problem.
51	Janzen, D. et. al.	Case/Field Study	Teaching w ith Test Driven Development	Student	140 Students	Test first students scored higher grades and learned better how to test.
54	McDow ell, C. et. al.	Case/Field Study	Teaching w ith Pair Programming	Student	554 Students	More confident and proficient students. Higher grades for all pairs suggest no 'easy ride' for low skilled students. Female programmers benefit from pair programming.

C. Tolfo et al. (2009) [1] investigates 3 companies to identify the effect that organizational culture has on the implementation of agile methods. Only 1 of the 3 companies reported successful adaptation of the agile method. The other 2 companies weren't fully adopted agile methods and their organizational culture was conflicting. He found out that by representing and visualizing the different cultural levels of the organization it makes it easier to adopt an agile method successfully. Those levels have to be understood because sometimes companies misunderstood them and don't fully adopt the agile method. So if a company faces problems adopting an agile method, it must be looked up if the culture of the company hinders the adoption. C. Tolfo *et al*. also states that organizational culture doesn't involve only the operational and tactical context but also the strategical. This strategical context includes investors, managers and even customers. All those entities can affect the adoption of the agile method, so before attempting to adopt one, it must be discussed extensively because the strategical context can act as a barrier to the agile method adoption. The first step for adopting and agile method should be the identification of the company's cultural levels. These cultural level exist for large as well as small and new companies.

C. Tolfo et al. (2009) [2] on another study about the influence of organizational culture on the adoption of Extreme Programming (XP), investigated 6 companies. The results were that there is a relation between successful adaptation of XP and company's culture, as mentioned in [1]. It is important for companies to evaluate their culture before moving to the adoption of an agile method. In this process questionnaires can help but they can't be used alone because they don't always yield valid results. Interviews can also be used, as well as an observer. All those cultural conflicts can be the cause of companies unable to adopt XP throughout, and they remain only to the adoption of a specific XP practice and while some practices encounter positive culture, some other encounter negative.

Petersen, K. et. al. (2009) [5] conducted a study to compare issues and advantages of existing empirical results on agile methods with an industrial study in a large-scale project. Results show that when adopting agile methods in large-scale projects many advantages occur on one side of the project but on the same time new issues arise on some other side. For example Petersen, K. et. al. state that using small teams increases control over the project but raises issues on the management level where the coordination of the projects has to take place. The study identified almost no new advantages of agile methods but new issues that haven't been mentioned in previous literature were found. The study states that software companies need to choose agile practices carefully because often only the advantages are taken into consideration and not the possible drawbacks that those practices might have.

Moe et. al. (2010) [6] conducted a nine month field study in a professional software development team by introducing them to the Scrum model. Results show that the team had difficulties adopting Scrum. One of the reasons was the difficulty to implement self-managing teams, as agile software development proposes. There was a lack of trust between the Scrum master and the members of the team as well as lack of a shared mental model. Scrum and agile methods offer no advices on how shared leadership should be implemented. Also highly specialized skills and a corresponding division of work were factors that hindered effective teamwork.

Lee, S. et. al. (2009) [8] presents a study of distributed agile project management. Specifically it examines the My Yahoo! 'Zorro' and My Yahoo! 'Chameleon' projects. 'Zorro' was a project using non-agile methods that was launched internationally and had many problems including localization problems and much time consumption. With 'Chameleon' the My Yahoo! team implemented Scrum. The launch was more successful with up to 30% improvement in the product quality and more customer satisfaction. Lee, S. et. al. suggest that international Scrum masters should be replaced by regional Scrum masters, and become regional representatives. Furthermore they state that mutual respect is the key to build trust in a distributed environment and teams need to build trust and respect of cultural differences. They conclude and suggest to other companies to implement distributed agile as they will find that their project quality and productivity will increase.

Layman, L. et. al. (2006) [10] explains a study that was undertaken to examine the adoption of Extreme Programming (XP) to Sabre Airline Solutions software company. The authors noticed that among all the XP practices, stand-up meetings and continuous integration were the most popular and the ones that had positive opinions. Testing techniques, such as unit tests and TDD, weren't used too much. Developers said that due to the large amount of legacy code that existed before adopting XP, unit tests were difficult to perform as they would require considerable effort. Developers also didn't write tests in order to meet the deadlines and because they thought that testing the whole legacy code wasn't cost effective. Pair programming was only used in complex situations and was abandoned again in order to meet deadlines. Collective code ownership had a positive opinion but because the project was large some specific developers had specialized knowledge of some parts of the code. A drawback on collective code ownership was observed where developers had decreased amount of responsibility for poorly written code because they weren't the only ones responsible for that part of code. They conclude that Sabre team showed similar pre-release defect number and lower defect removal efficiency. Also Sabre team released the project with lower number of defects, and in total (post-release and pre-release) defects were lower than industry averages, although they state that the results might need further investigation.

Layman, L. et. al. (2006) [11] conducted an industrial case study to examine a distributed team in USA and Czech Republic using Extreme Programming (XP), and how they managed to create a successful project. Their findings agree with Lee, S. et. al. (2009) [8] that distributed software teams can successfully adopt agile methods. Specifically he points out some recommendations that make a distributed agile team to create successful software products. Distributed agile teams should define a person to play the role of the customer immediately after the project is started. The customer must make decisions upon the project and guide the developers, as well as having a big interest in the project. Furthermore it is suggested that when the project management teams and the development teams are separated someone should take up the role of communicating daily with both of those teams and can speak all the languages involved. Also when face-to-face communication is not possible the teams can use e-mail listserv to increase communication and provide quick responses. Finally they suggest the use of a globally available project management tool (such as XPlanner) to monitor the project at all times. According to the authors all those recommendations are essential for a distributed team to implement agile methodologies and create a communication-rich environment.

Hanssen, G. et. al. (2006) [15] in a case study examined the effect of customer engagement in an agile project. The study involved a company that changed their development process to an agile one. Agreeing with other studies he mentions that customer cooperation with the project is very important and motivated the developers. Also developer's confidence increases as a result of the continuous settlement of objectives. Visibility of the project was also increased after the adoption of agile methods. Furthermore they state that there was an increase in cost for running the agile customer engagement practices. They also state that short iterations might expose the company to a risk, and that its better if a large number of customers is used so developers can capture effectively the all possible needs.

In a case study performed among six groups of students, Germain, É. et. al. (2005) [16] investigated the effect of effort between 2 software development methods. From the traditional methods area they used UPEDU and from the agile methods area they used XP. Results didn't show a significant difference between the effort required for the stages of each process. One process required more effort on one area while the other on another area. But the overall effort was independent from the process that was used. They also noticed no significant productivity reduction to the UPEDU team for constructing explicit artifacts although the students didn't really realize the need for those artifacts. They state that the project was small and that might affected that result.

Another article of Fogelström, N. et. al. (2009) [18] investigates if agile principles can be adopted in a market driven software product development (MDPD). MDPD doesn't focus on one customer (as agile methodologies propose), instead it focuses on a mass market. The study was conducted at Ericsson. Ericsson was using traditional methodologies and they moved to an agile environment. Results were negative. A misalignment of the properties of agile development and MDPD was observed. 'Evolving release scope', 'feature orientation' and 'reactive development', all contradict with the principles of MDPD. Agile methodologies having build with a focus to the project, lack support for a long-term product development focus. They conclude that agile methodologies can hinder product development if they are applied to a MDPD focused project, but certain practices might be able to be used by MDPD projects for example practices that minimize time spent on analyzing pre-project decisions.

Lan, C. (2004) [21] examines an organization that adopted agile practices in a large scale project. Results show that with a few changes to the agile principles, agile methodologies can be adopted to large scale projects. Specifically up-front design despite being discouraged in agile principles, is necessary for large scale projects. Also it is better for companies with large scale projects to implement short releases with a layered approach. Short release cycle will not have fixed duration but it will depend on the layers and tasks. Pair programing can also be used but only in some specific situations like unit testing and test-case development. In conclusion agile principles can benefit large scale projects and offer them faster software development and they can be effectively adopted to large scale projects but some changes need to happen to the agile principles in order for the project to be successful.

Laurie, W. et. al. (2003) [24] explains a case study that was conducted at IBM that compared 2 teams developing a legacy product using traditional approaches and a new product using TDD approach. They noticed a significant reduction in defect rate for the TDD team. Specifically the TDD team had almost 40% lower defect rate than the legacy team. In the meantime TDD team developers despite spending more time testing, they spent less time debugging. Productivity was also same between the 2 teams. They conclude their study stating that TDD had helped them minimize risk because problems appeared much earlier.

In another study at IBM, Maximilien, E. et. al. (2003) [25] reports same results. Product defect rate by using TDD was reduced by 50%. Meanwhile productivity didn't decrease by the time spent for testing.

Bhat, T. et. al. (2006) [29] presents a case study that was taken place in Microsoft in two divisions, Windows and MSN. The case study aimed to investigate the effect of TDD on program code and quality. Results show that projects using TDD required more time. This is to be expected because extra tests are being performed. Also TDD projects resulted in a better code quality by at least 2 times. The authors conclude that further investigation is needed especially to investigate the cost between requiring more time and producing better code.

Damm, L. et. al. (2006) [30] conducted a case study that aimed to test TDD on component level instead of class/method level. They used 2 projects at Ericsson AB. to compare them to their similar predecessors that weren't using TDD approach. The authors measured Affordable Fault Cost (AFC) and Return on Investment (ROI). They state that especially with ROI despite being positive with the TDD approach that “the real benefits come in subsequent releases”. Results of the case study showed that the TDD investment had a significant ROI from the decreased fault costs. And since the decreased fault costs were noticed due to decreased fault slipping through the stages of the project, it is likely that the overall end project quality was increased due to lower number of errors. Lead time of each process was also decreased since test leaders stated that bug fix deliveries were the most important factor for lead time. Decreased error rate and the overall TDD approach also increased the delivery precision, but the authors state that concerning delivery precision results may not be 100% correct because there are many factors that affect delivery precision. The total fault cost, measured using AFC, of the project was 30% less that with the traditional approach and the total project cost, measured using ROI, became 5-6% less. The authors conclude that by using TDD there might be more beneficial factors that are hard to measure, like increased company maturity and increased developer motivation.

Bipp, T. et. al. (2008) [35] presents an extensive case study that was taken place at the University of Dortmund, Germany. 13 software development teams with approximately 100 students were examined. Teams were divided into two groups, one that used pair programming teams, and the other that used solo programmers. Conclusions of the study show that pairs gained more knowledge about the project and used their time more efficiently and that led them to not require more time than solo programmers. Furthermore testing and bug fixing is much easier for pair programmers, they produce higher quality code and less experienced programmers can be more easily integrated in pairs. The only disadvantage that was noticed was a minor loss in efficiency.

Hulkko, H. et. al. (2005) [38] conducted a case study to investigate the effect of pair programming on the overall product quality of a project. They examined 4 industry software projects and the results were the following. Pair programming was found especially useful during the first phases of the project and during the last phase. Overall productivity wasn't found to be significantly different than solo programming. Furthermore pair programming seems to be more suitable for implementing complex tasks, for learning and for finding mistakes on simple code than solo. Coding standard adherence had higher deviation for pair programming and comment ratio was higher for pair programming. In conclusion, pair programming, except on one case study, wasn't found to produce lower defects, and that results contradicts to many literature data. The authors believe that pair programming overall doesn't provide extensive quality benefits than solo programming. Although more studies may be needed especially to distinguish defect's severity between the 2 methods.

Laurie, W. et. al. (2000) [45] reports on a case study conducted at the University of Utah between 2 groups of students, using pair programming and solo programming. Results show that pairs produced higher quality code with fewer errors and faster than solo programmers. This is important for industry because delivering a product faster and a product that will need fewer maintenance outweighs the minimal cost of using 2 programmers for the same task. Furthermore another positive impact that was seen was the pair-pressure. Pairs put pressure on each other and are less likely to spend their time doing something else other than coding. Lastly most programmers that worked with pair programming report that they enjoyed it and that they enjoyed their work more. The only problems with pair programming exist when someone pairs with another one of excess ego (he wants to do everything his way), or too less ego (he doesn't contribute to the group at all).

Melis, M. et. al. (2006) [46] presents a case study performed by using a simulator process based on 4 projects. They measured the KLOCs, the working days, the defects and the user stories on different situations based on the use of pair programming (PP) and test-driven development (TDD). Results indicate that by using PP the resulting defect density is reduced by 28%. and LOCs are reduced by 35%. Overall, the use of PP increased working hours but decreased defect density and produced a better design by reducing the LOCs. On the other hand by using TDD defect density reduces by 65% and there was a 9% increase in LOCs. TDD increases working hours (less than PP) and the needed test cases increase the LOCs, but the resulting defect density is decreased (more than PP). By using both PP and TDD, the working time increased by 32%, but the resulting project source size decreased by 19% and the resulting defect density was cut to half. If we try to achieve the same quality that is achieved with PP and TDD but without using those techniques then the needed working hours increase by 145%.

Chong, J. et. al. (2007) [47] reports on a case study that involved 2 software development teams in San Francisco Bear Area. Through their observations on the 2 teams they conclude on the factors that companies should consider when adopting pair programming. First they suggest that the usual recommendation of 'driver' and 'navigator' in the literature should not be so rigid. The pairs appeared to be more effective when they switched between those 2 positions. They also felt more engaged in their tasks when they used a keyboard or when they had the opportunity to use it soon. This approach should be considered by the developers of tools for pair programming. Those tools should help the transition between the 'driver' and 'navigator' role. Furthermore skill differences between pair programmers should be taken into account. Most pair programmers don't like to pair with different skilled programmers. There seems to be an exception when the lower skilled programmer is new to the company. Lastly pairs can change at the first stages of the project and when the tasks are small because it helps to spread code knowledge between the team, but at later stages it should be avoided because it can break up an effective pair and introduce a programmer new in the task.

Proulx, V. et. al. (2009) [48] presents a case study that explores the effects of TDD (test driven development) when it is introduced in a university curriculum.

Students after taking the curriculum had better performance and had lower chance for failure. Also students who had taken TDD curriculum had more chances to co-op with professional programmers and companies found them to be more skilled than master's degree students. Overall the students understood better the whole programming language and its details.

Simon, B. et. al. (2008) [50] performed a case study that consisted of students of 2 institutions that pair programmed in CS1 class and then continued to work alone in CS2. Students stated that it was more easier to understand some aspects of programming when working in pairs. They find solo programming as 'lonely' and 'stressful' when they encounter difficulties and they can more easily solve those difficulties when they work in pairs. Furthermore they can find more solutions to their problems when working in pairs. Depending on the time necessary to complete the task, student's responses varied. It seems that the good students can finish their tasks faster when they program solo. Here emerged a problem with pair programming that was the scheduling. Students had problems to schedule hours that they should meet and program together. Also students that program solo find it more exciting when they complete the tasks alone. On the other hand pair program is noted as more social and an opportunity to meet other students. Lastly many solo programming students confessed that when they had problems they contacted a fellow student to help them.

Janzen, D. et. al. (2008) [51] conducted a case study on students to test the effects of TDL (test driven learning). They used 2 groups of students, one that was using test-first and one that was using test-last. The authors concluded that test-first helped students write more unit tests and it also helped them write more tests even when they changed back to test-last. Test-first students also scored higher grades and required less time than test-last students.

McDowell, C. et. al. (2006) [54] conducted a case study to investigate the effect of pair programming on 554 students at the University of California-Santa Cruz. The authors concluded that pair programming produces more confident and proficient programmers. Paired students also had higher grades and more success course completion rate and that result suggests that students don't “use” their pair just to pass the courses. Pairs produce more quality code and are more confident, agreeing with previous studies. Lastly women seem to benefit from pair programming especially in a field that has low women representation, specifically who work in pairs have higher retention rates that solo programming women.

4.1.3 Surveys / Reviews

Following the research of case / field studies, this review will now investigate surveys and reviews and provide an analysis of the papers below. Table 3 has the summary of those papers.

Table 3

4	Salo, O. et. al.	Survey/Review	Agile Methods	Professional	13 Companies	54% applied XP, 27% applied Scrum. Most companies that applied Agile methods w ere satisfied.
9	Dybå, T. et. al.	Survey/Review	Agile Methods	Literature	33 Articles	XP most used. Companies and customers satisfied. On-site customer and pair programing reported as exhaustive.
12	Hansson, C. et. al.	Survey/Review	Agile Methods	Professional	900+ Programmers	Most companies had agile characteristics despite not using agile methods. Examples include changing requirements and customer relationship.
13	Bow ers, A. et. al.	Survey/Review	XP	Professional	27 Programmers	Metaphor, acceptance tests, on-site customer least used. All practices should be used for successful adaption.
14	Chow , T. et. al.	Survey/Review	Agile Methods	Professional	109 Agile Projects	High caliber team, proper agile techniques practice, agile delivery strategy the success factors for adaption.
22	VersionOne	Survey/Review	Agile Methods	Professional	2391 Programmers	Fast delivery and changing requirements most praised agile benefits. Up front planning is bigest concern. Scrum most used. Organizational culture is biggest barrier. Increase in agile use from 2007.
33	Begel, A. et. al.	Survey/Review	Pair Programming	Professional	487 Respondents	More than 60% of developers believe that pair programing w orks w ell and that it produces high quality code.
40	Hannay, J. E. et. al.	Survey/Review	Pair Programming	Literature	18 Articles	Increased quality for complex tasks at the expense of increased effort. Novice programmers achieve higher correctness w ith pair programming.
53	Desai, C. et. al.	Survey/Review	Literature	Student	18 Articles	Student code had few er defects. Productivity increased.
70	Scott W. Ambler	Survey/Review	Agile Methods	Professional	781 Respondents	69% use agile methods. 77% of projects w ere 75-100% successful. Co-located teams are the most successful. Small team and 2 w eek iterations are most used.

Salo et al. (2008) [4] made a survey concerning 13 industrial organizations in 35 different European countries. He examined both companies that have adopted agile methods as well as companies that haven't. 54% of the companies that have adopted agile methods are applying XP practices. The most used practices were found to be: open office space (66%), coding standards (61%), 40h week (59%), continuous integration (44%) and collective ownership (42%). The least used practices were: TDD (41%), pair-programming (33%), collective code ownership and on-site customer (30%), simple design and planning game (28%). From the applied practices nearly 90% were rating them as useful and a 5,8% were rating them as harmful. Concerning Scrum 27% stated that they apply Scrum practices often. Product Backlog is the most favored practice with a 24%. The rest Scrum practices weren't used that much. 77% were satisfied from the Scrum practices and 11% were negative towards them. Salo et. al. also noted that the experienced usefulness of XP and Scrum was much higher than the expected usefulness. In companies that haven't applied XP 57% were positive (against 90%) and 28% were negative (against 5,8%). For Scrum the positive side was 28% (against 77%). Also there seems to be a fairly low knowledge of Scrum practices (20% of respondents answered 'I don't know' on the questionnaires). Salo et. al. concludes that despite the various limitations of the survey, a positive result was observed after the adoption of XP and Scrum practices in the software companies.

Dybå, T. et. al. (2008) [9] presents the results of a systematic review on empirical studies of agile development. XP was the method used by almost all the companies in the primary studies of this review. They mention that XP is more suitable for small companies rather than large-scale projects. They report that most agile practices are easy to adopt and work well. A benefit of XP was that it flourished on many diverse company environments. Customers are satisfied with agile methods although on-site customer can sometimes be stressful. Companies and developers are also satisfied with agile methods although some find pair programming to be an exhaustive practice. Lastly agile methods seem to yield better code quality although the authors state that this result might be biased. They conclude that their strength of evidence wasn't not enough and that further research might be necessary.

Hansson, C. et. al. (2006) [12] describes a survey made between various professional software developers with the purpose to examine if existing industrial development processes have agile characteristics. They found out that industries have more agile practices than they are aware of. Despite traditional practices stating that requirements should not change in the middle of the development process, all companies tested, accept new requirements or change already accepted ones. Iterative development was also observed. Although only one company implements this approach, the other companies release more or less frequent updates. Most companies also had long-term relationship with the customer, a characteristic that is not part of the traditional software engineering. They also found that innovativeness on projects has more agile characteristics than traditional methods, which relate more to project size and criticality. Hansson, C. et. al. finalizes their results by stating that it all depends on the characteristics of not only the company but of the individual projects also.

Bowers, A. et. al. (2007) [13] presents a survey that was taken. The survey was conducted between 27 developers on different software companies. Results show that among the least used agile practices was the metaphor. They attempt to explain that by saying that metaphor is probably abstract and that different skilled developers cant easily recognize a single metaphor. They continue by stating that despite unit testing and TDD was popular, acceptance tests weren't that much. But lack of acceptance tests can reduce developer confidence for refactoring. Another practice that wasn't used much was, by surprise, on-site customer. On-site customer is one of the most important practices of agile and the absence of it can greatly hinder agility of the project. And worst of all developers may end up taking the role of customers. Bowers, A. et. al. conclude that in agile methodologies all their practices should be used because they complement each other and their weaknesses. If that is not possible then the developers need to understand the intent of each practice very well so they can find alternative ways to overcome the weakness that will arise from the absence of that practice.

On another survey study Chow, T. et. al. (2008) [14] presents a survey study that gathered data from 109 agile projects across 25 countries. They found out that many success factors that exists in the existing literature are not correct. Only 10 out of their 48 hypothesis were confirmed. Specifically they found that the success factors for an agile project are, a) having a high caliber team, b) proper practice of agile development techniques, c) and a correct agile-style delivery strategy. 3 more factors were identified that may contribute to success in some cases. Those include, a) agile-friendly environment, b) good agile project management process, c) strong involvement of the customer. They finalize their report by stating that as long as those 3 factors are followed then the agile project will probably be a success and no more unnecessary factors need to be considered.

A survey conducted by VersionOne (2008) [22] shows some interesting results. The survey conducted between 2391 participants of 80 countries. The first interesting result shows that 57% of the participants use a distributed agile approach. This agrees with some articles that we mentioned before that distributed agile approach can and is used by companies. The most important factors that companies considered for adapting agile methodologies was the fast software delivery and the increased ability to respond to changing requirements, followed by increased productivity and software quality. Up front planning seems to be the biggest concern of software companies before adapting agile principles. Scrum and Scrum/XP hybrid are the most used agile methodologies. Especially Scrum was chosen by almost half of the companies tested. Organizational culture and resistance to change are the biggest barriers for companies to further increase their agile adoption. Iteration planning, unit testing, daily standup meetings and release planning are the most used agile practices. Despite that all the agile practices had an increase in usage compared to a similar 2007 survey. 76% of companies report that 75%-100% of their agile projects were successful.

Begel, A. et. al. (2008) [33] explains a survey that was taken place in Microsoft between a random 10% of the developers. Survey findings indicate that 64.4% of the respondents believe that pair programming works well with them. That result lowers if the question moves up to the team and organization level. 65.4% of the respondents believe that pair programming produces higher quality code. Time taken for pair programming is divided between respondents believing that it takes less time than solo programming and respondents believe that it takes more. Overall the first benefit that most respondents find in pair programming is better code quality, followed by fewer bugs. The problems of pair programming include cost efficiency, scheduling issues and personality conflicts. Lastly survey shows that most programmers would like to pair with people that have complementary skills.

Hannay, J. et. al. (2009) [40] presents a meta-analysis of studies on pair programming to investigate the effect it has on software projects. Pair programming is suggested that it has positive effect and it helps achieving correctness on highly complex programming tasks, also mentioned in Hulkko, H. et. al. (2005) [38]. But this increased quality seems that it has the cost of higher effort. Also pair programming is more suitable for simple tasks as it provides a time gain. Programmers are able to achieve and complete tasks in pairs that would be more difficult if they worked alone. Furthermore novice programmers when they work in pairs are better able to achieve the same correctness level as senior programmers. The authors conclude that if a company doesn't know the skill level of its programmers but it does know the complexity level of the projects and its tasks then they may implement pair programming when tasks are simple and time is important, or when tasks are complex and quality is important.

Desai, C. et. al. (2008) [53] reports on a survey among studies that explored the effect of TDD on students. Most studies report that students using TDD were more confident, their code had fewer defects and generally their productivity increased. There were some studies were TDD had no significant impact but the majority of studies suggests that TDD does help students greatly, and introduces them to program testing.

Ambler, W. S. (2007) [55] presents a survey that involved 781 participants in software industry. From the results, 69% of participants use agile methods. Of those that don't use it only a 12% says that they will never try it. 77% of projects were between the range of 75%-100% successful, while projects that are co-located seem to be the more promising for success. On the other hand off-shoring projects have the least chance for success. 2 weeks, followed by 4 weeks are the most used iteration lengths. 1-10 people are the most used team sizes in agile projects. Most companies have more than one agile project so it wasn't the first time they used agile.

4.2 Limitations of the Review

The first limitation of this review lies in the selection of the studies and in the process of data extraction. There is a chance that the search strings were not sufficient enough to find all the available and important studies for the purpose of this review. Also during the data extraction process, it was noted that some studies lack sufficient description of some of their characteristics. For some articles it was difficult to determine, for example, the clear description of the methods used. Lastly this review included only studies with empirical and/or statistical data, so if more studies were included instead of rejected, due to quality criteria, then results might have been different.

5. Discussion

The discussion will be split into 3 categories. First the experiments and their results will be discussed. Secondly the case and field studies and finally the surveys and the reviews.

5.1 Experiments

A total of 21 experiments were included. Overall most reports from experiments indicate a positive impact from agile methods. Teams that use agile methods experience an overall better well being [3]. When adapting agile methods customer and companies culture are very important [7]. Stakeholders enjoy agile methods and the more agile methods that are used the more the stakeholders are satisfied so this proves that even 'outside' people other than programmers can see the benefit from agile methods [19]. A experiment [17] showed that System Design Instability metric (SDI) can be effectively used with agile methods although it warns that new design-effort can increase instability of project, but it should be noted that it doesn't increase error fix and refactoring as some might think [20] and if the instability is taken into account then it might be safely applied [20]. Considering pair programming (PP), one of the popular and maybe controversia20l agile practices, the pairs should be different personality groups. Same personality groups doesn't seem to work very well with PP. Specifically heterogeneous [32] and diverse Myers–Briggs Type Indicator (MBTI) type groups [42] are the most suitable. An experiment to students [44] showed that low level confident people enjoy more PP because they have a partner to talk and solve the problems with so this is a sign of suitable people to work in pairs. If it is difficult to implement PP as a whole, it is also possible to split pair programming and only adapt a pair design phase. Experiments show that pair design phase results in no increase in cost [37], it also benefits companies as it causes better quality prediction although it lowers the development time prediction [41]. Furthermore PP results in better quality [39] [43], and it can greatly help with complex problems [39]. It is worth mentioning that to achieve the quality of PP with solo programming the cost must increase very much [43]. A recent study though found no significant difference in error fixing capability (branch coverage (BC) and mutation score indicator (MSI) scores) for solo and pair programming [36]. PP is also very useful for teaching CS courses as it helps students with low grades [34] and it provides them with a framework to solve their problems alone without needing teaching assistance [52]. Another practice of agile methods, test driven development (TDD), also showed positive results. Experiments show that TDD increases productivity [23] and quality [27] [31], but it needs more time in order to run the tests [27] [31]. Most developers find TDD useful but some find it difficult to implement [28]. Same study that was run for PP also shows no significant difference in BC and MSI scores for TDD [26] and that is surprising as someone can expect better error fix coverage when adapting TDD.

5.2 Case/Field Studies

24 Case and/or Field studies were included in the review. One important factor when adapting agile methods is organizational culture [1] [2] and it should be examined before adapting agile methods to see if it is compatible. Another consideration for companies that want to adapt agile methods is that while some

agile practices have benefits there is a chance that they will also have negative effects [5] and this should be considered to not overlook them. Customer is also very important for agile methods [15] and he should participate in the development process for the project to be successful. While agile methods propose close communication and face to face interaction with customers, distributed agile projects have been found to be successful [8] [11], but communication is a very important factor for those projects and measures should be taken to implement it even from remote locations, and with some changes to the agile practices even large scale projects can adapt agile methods [21]. Defect rate when using agile methods decreases [10], especially if test driven development is also implemented [24] [25] [29], although it raises time requirements as mentioned also in experiments [29], at the same time effort doesn't increase for agile methods [16]. On the other hand market driven software product development (MDPD) projects should not adapt agile methods but maybe some specific practices only can help them [18]. Pair programming implemented through the use of agile methods also results in better quality [35] [45] [46], although some found no difference in quality [38]. Agile practices also benefit students when they implemented in universities. They increase student's grades [51] [54] and they help students achieve better performance [48]. One problem here is pair scheduling for student pairs [50] and it should be taken into consideration or the pair wont experience the benefits.

5.3 Surveys/Reviews

Lastly 10 surveys and reviews were chosen. Results show that agile use have increased through the years [4] [22] [70]. More and more companies find agile methods and practices useful, particularly they praise fast delivery and changing requirements [22]. On site customer seems to be the least used [13] and is found difficult to implement [9] followed by pair programming [9], metaphor and acceptance tests [13], but on the other hand some companies had successfully adapted pair programming without problems [33] and here emerges again the problem of organizational culture [22] that might affect proper adaption. It should be noted here that the proper use of agile practices [14] and the correct implementation of all [13] is an important factor for success. Despite some practices being hard to use, and this greatly depends on the organizational culture also [22], companies should try to implement them when they try to go agile. Agile methods and specifically pair programming if they are adapted successfully they improve overall quality, as mentioned in previous studies. Lastly it is worth mentioning that companies that haven't adapted agile methods, already have agile characteristics and use agile practices without noticing it and even skip some of the rules of their traditional methods to implement agile characteristics that can benefit them more [12].

6. Conclusions

Results from experiments case studies and surveys seem to complement each other. Agile practices have a benefit to the projects and they help improve overall quality and cost of software projects [10] [23] [24] [25] [29] [27] [31] [35] [39] [41] [43] [45] [46]. Success rate has been increased through the years and a recent study [22] shows a 76% of companies reporting that 75%-100% of their agile projects were successful. Both surveys, experiments and case studies agree in one factor for success, and that is organizational culture [1] [2] [7] [22]. If culture is not in line with agile characteristics it can greatly hinder adaption and even lead the project to failure. Experiments and case studies also report customer engagement [15] to be very important and surveys suggest that generally the more agile practices a company adapts successfully the more chance the project has for success [13]. The experiments that were included, report that agile methods are very useful for small companies. Meanwhile case studies and surveys conclude that even large scale projects can use agile practices (with some changes) and benefit from them [21]. Distributed projects can also effectively use agile methods but good communication should be achieved [8] [11]. Agile methods generally help projects and very few studies reported no significant benefit [16] [26] [36]. Productivity is also improved in some cases [8] [23] [53], but time effort will most likely increase [27] [29] [31] [40] [41] [46]. Most experiments report less defects by using agile methods and practices such as pair programming (PP) and test driven development (TDD) [20] [27] [31] [39] [43]. There were however some experiments

[26] [36] [49] that report no difference. Case/Field studies follow the same line with experiments, stating an overall 30-40% lower defect rate and a 30-70% increased productivity [8] [10] [24] [25] [29] [30] [35] [45] [46] [48]. From surveys, results show that developers agree with those findings reporting that they find agile methods and practices to produce better quality code [4] [9] [22] [33] [70], although some report difficulty adapting them [9]. Results from experiments show that PP seems to be most effective when pairs have a diverse personality type [32] [42]. Experiments show that in order to reach same quality with traditional methods the cost and time raise too much [43]. Agile methods and practices can also greatly benefit students and all the studies agree in that. Both experiments and case studies conclude that student grades are improved after the introduction of agile practices, such as PP, in the classroom [34] [48] [49] [50] [51] [52] [54]. Students more easily pass their classes, and require less assistance from teachers as they can better solve problems on their own. A small percent of students especially high skilled and high confident students didn't like pair programming [44].

To conclude, agile methods since their first inception have increased in use and most projects report success, especially with small teams, but even different type of teams can benefit from them with some changes. This review hopes to provide some guidelines for companies about some factors that should be taken into consideration when adapting agile and also report on the current state of agile practices. Further reviews are necessary especially by including even more studies especially professional studies to further solidify the results of this review.

7. References

[1] Tolfo, C., R. S. Wazlawick, et al. (2009). "Agile methods and organizational culture: reflections about cultural levels." Software Process: Improvement and Practice 9999(9999): n/a.

[2] Tolfo, C. and R. S. Wazlawick (2008). "The influence of organizational culture

on the adoption of extreme programming." Journal of Systems and Software 81(11): 1955-1967.

[3] Syed-Abdullah, S., M. Holcombe, et al. (2006). "The Impact of an Agile Methodology on the Well Being of Development Teams." Empirical Software Engineering 11(1): 143-167.

[4] Salo, O. and P. Abrahamsson (2008). "Agile methods in European embedded software development organisations: a survey on the actual use and usefulness of Extreme Programming and Scrum." IET Software 2(1): 58-64.

[5] Petersen, K. and C. Wohlin (2009). "A comparison of issues and advantages in agile and incremental development between state of the art and an industrial case." Journal of Systems and Software 82(9): 1479-1490.

[6] Moe, N. B., T. Dingsøyr, et al. "A teamwork model for understanding an agile team: A case study of a Scrum project." Information and Software Technology In Press, Corrected Proof.

[7] Misra, S. C., V. Kumar, et al. (2009). "Identifying some important success factors in adopting agile software development practices." Journal of Systems and Software 82(11): 1869-1890.

[8] Lee, S. and H.-S. Yong (2009). "Distributed agile: project management in a global environment." Empirical Software Engineering.

[9] Dybå, T. and T. Dingsøyr (2008). "Empirical studies of agile software development: A systematic review." Information and Software Technology 50(9-10): 833-859.

[10] Layman, L., L. Williams, et al. (2006). "Motivations and measurements in an agile case study." Journal of Systems Architecture 52(11): 654-667.

[11] Layman, L., L. Williams, et al. (2006). "Essential communication practices for Extreme Programming in a global software development team." Information and Software Technology 48(9): 781-794.

[12] Hansson, C., Y. Dittrich, et al. (2006). "How agile are industrial software development practices?" Journal of Systems and Software 79(9): 1295-1311.

[13] Bowers, A. N., R. S. Sangwan, et al. (2007). "Adoption of XP practices in the industry - A survey." Software Process: Improvement and Practice 12(3): 283-294.

[14] Chow, T. and D.-B. Cao (2008). "A survey study of critical success factors in agile software projects." Journal of Systems and Software 81(6): 961-971.

[15] Hanssen, G. K., T. E. F\, et al. (2006). Agile customer engagement: a longitudinal qualitative case study. Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering. Rio de Janeiro, Brazil, ACM: 164-173.

[16] Germain, É. and P. N. Robillard (2005). "Engineering-based processes and agile methodologies for software development: a comparative case study." Journal of Systems and Software 75(1-2): 17-27.

[17] Alshayeb, M. and W. Li (2005). "An empirical study of system design instability metric and design evolution in an agile software process." Journal of Systems and Software 74(3): 269-274.

[18] Fogelström, N. D., T. Gorschek, et al. (2009). "The impact of agile principles on market-driven software product development." Software Process: Improvement and Practice 9999(9999): n/a.

[19] Ferreira, C. and J. Cohen (2008). Agile systems development and

stakeholder satisfaction: a South African empirical study. Proceedings of the 2008 annual research conference of the South African Institute of Computer Scientists and Information Technologists on IT research in developing countries: riding the wave of technology. Wilderness, South Africa, ACM: 48-55.

[20] Alshayeb, M. and W. Li (2006). "An empirical study of relationships among extreme programming engineering activities." Information and Software Technology 48(11): 1068-1072.

[21] Lan, C. (2004). How Extreme Does Extreme Programming Have to Be? Adapting XP Practices to Large-Scale Projects. Hawaii International Conference on System Sciences.

[22] VersionOne, (2008), 3rd Annual Survey 2008 “The State of Agile Development”, Available at: “http://www.versionone.com/pdf/3rdAnnualStateOfAgile_FullDataReport.pdf”.

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

[24] Laurie, W. (2003). “Test-Driven Development as a Defect-Reduction Practice.”

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

[26] Madeyski, L. (2010). "The impact of Test-First programming on branch coverage and mutation score indicator of unit tests: An experiment." Information and Software Technology 52(2): 169-184.

[27] Canfora, G., A. Cimitile, et al. (2006). Evaluating advantages of test driven development: a controlled experiment with professionals. Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering. Rio de Janeiro, Brazil, ACM: 364-371.

[28] Janzen, D. S. and H. Saiedian (2007). A Leveled Examination of Test-Driven Development Acceptance. Proceedings of the 29th international conference on Software Engineering, IEEE Computer Society: 719-722.

[29] Bhat, T. and N. Nagappan (2006). Evaluating the efficacy of test-driven development: industrial case studies. Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering. Rio de Janeiro, Brazil, ACM: 356-363.

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

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

[32] Sfetsos, P., I. Stamelos, et al. (2009). "An experimental investigation of personality types impact on pair effectiveness in pair programming." Empirical Software Engineering 14(2): 187-226.

[33] Begel, A. and N. Nagappan (2008). Pair programming: what's in it for me? Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement. Kaiserslautern, Germany, ACM: 120-128.

[34] Braught, G., L. M. Eby, et al. (2008). The effects of pair-programming on individual programming skill. Proceedings of the 39th SIGCSE technical symposium on Computer science education. Portland, OR, USA, ACM: 200-204.

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

[36] Madeyski, L. (2008). "Impact of pair programming on thoroughness and fault detection effectiveness of unit test suites." Software Process: Improvement and Practice 13(3): 281-295.

[37] Müller, M. M. (2006). "A preliminary study on the impact of a pair design phase on pair programming and solo programming." Information and Software Technology 48(5): 335-344.

[38] Hulkko, H. and P. Abrahamsson (2005). A multiple case study on the impact of pair programming on product quality. Proceedings of the 27th international conference on Software engineering. St. Louis, MO, USA, ACM: 495-504.

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

[40] Hannay, J. E., T. Dybå, et al. (2009). "The effectiveness of pair programming: A meta-analysis." Information and Software Technology 51(7): 1110-1122.

[41] Canfora, G., A. Cimitile, et al. (2007). "Evaluating performances of pair designing in industry." Journal of Systems and Software 80(8): 1317-1327.

[42] Choi, K. S., F. P. Deek, et al. (2008). "Exploring the underlying aspects of pair programming: The impact of personality." Information and Software Technology 50(11): 1114-1126.

[43] Müller, M. M. (2005). "Two controlled experiments concerning the comparison of pair programming to peer review." Journal of Systems and Software

78(2): 166-179.

[44] Thomas, L., M. Ratcliffe, et al. (2003). Code warriors and code-a-phobes: a study in attitude and pair programming. Proceedings of the 34th SIGCSE technical symposium on Computer science education. Reno, Navada, USA, ACM: 363-367.

[45] Laurie, W. (2000). Strengthening the Case for Pair Programming. IEEE Software. R. K. Robert, C. Ward and J. Ron. 17: 19-25.

[46] Melis, M., I. Turnu, et al. (2006). "Evaluating the impact of test-first programming and pair programming through software process simulation." Software Process: Improvement and Practice 11(4): 345-360.

[47] Chong, J. and T. Hurlbutt (2007). The Social Dynamics of Pair Programming. Proceedings of the 29th international conference on Software Engineering, IEEE Computer Society: 354-363.

[48] Proulx, V. K. (2009). Test-driven design for introductory OO programming. Proceedings of the 40th ACM technical symposium on Computer science education. Chattanooga, TN, USA, ACM: 138-142.

[49] Desai, C., D. S. Janzen, et al. (2009). 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.

[50] Simon, B. and B. Hanks (2008). "First-year students' impressions of pair programming in CS1." J. Educ. Resour. Comput. 7(4): 1-28.

[51] Janzen, D. and H. Saiedian (2008). Test-driven learning in early programming courses. Proceedings of the 39th SIGCSE technical symposium on Computer science education. Portland, OR, USA, ACM: 532-536.

[52] Hanks, B. (2008). "Problems encountered by novice pair programmers." J. Educ. Resour. Comput. 7(4): 1-13.

[53] Desai, C., D. Janzen, et al. (2008). "A survey of evidence for test-driven development in academia." SIGCSE Bull. 40(2): 97-101.

[54] McDowell, C., L. Werner, et al. (2006). "Pair programming improves student retention, confidence, and program quality." Commun. ACM 49(8): 90-95.

[55] Beck K., Cockburn A., Jeffries R., Highsmith J., (2001) “Agile manifesto”, http://www.agilemanifesto.org, 2-2010.

[56] Highsmith J., Orr K., Cockburn A., (2000) “Extreme programming in E-Business Application Delivery”, http://www.cutter.com/freestuff/ead0002.pdf, 2-2010.

[57] Cohen D., Lindvall M., Costa P., “An Introduction to Agile Methods”.

[58] Schwaber K., (2002) “Controlled chaos: living on the edge”, http://www.agilealliance.org/
articles/articles/ap.pdf, 2-2010.

[59] DSDM Consortium, http://www.dsdm.org, 2-2010.

[60] Wikipedia, (2010), “Pair Programming”
http://en.wikipedia.org/wiki/Pair_programming, 2-2010.

[61] Beck K., (2003), “Test-Driven Development by Example”, Addison Wesley.

[62] Rajlich V., (2006) “Changing the paradigm of software engineering”, Communications of the ACM 49 (8) 67–70.

[63] D.C. Cliburn, (2003), “Experiences with pair programming at a small college”, Journal of Computing Sciences in Colleges, 19(1).

[64] Salleh N., ”A Systematic Review of Pair Programming Research – Initial Results”.

[65] B.A. Kitchenham, (2007) “Guidelines for performing Systematic Literature Reviews in Software Engineering”, EBSE Technical Report, Software Engineering Group.

[66] Greenhalgh T., (2001), “How to Read a Paper”, second ed., BMJ Publishing Group.

[67] B.A. Kitchenham, S.L. Pfleeger, L.M. Pickard, P.W. Jones, D.C. Hoaglin, K. El Emam, J. Rosenberg, (2002), “Preliminary guidelines for empirical research in software engineering”, IEEE Transactions on Software Engineering 28 (8) 721–734.

[68] N. Katira, L. Williams, E. Wiebe, C. Miller, S. Balik, and E. Gehringer, (2004), “On understanding Compatibility of Student Pair Programmers”, Proceedings of the 35th SIGCSE technical symposium on Computer Science Education.

[69] S. F. Freeman, Jaeger B. K., J.C. Brougham, (2003), “Pair programming: More learning and less anxiety in a first programming course”, ASEE Annual Conference Proceedings.
[70] Ambler, W. S. (2007), “Agile Adoption Survey 2007”, Ambysoft, Available at: “http://www.ambysoft.com/surveys/agileMarch2007.html”.

Appendix A. Review Protocol

Research Questions

The research questions that this review will address are the following:

- Can agile methodologies and its practices help software companies develop software projects more effectively than traditional methods?
- Can agile methodologies help universities with teaching computer science courses to students?

Search Process

The search process was a manual search of documents through a list of digital libraries, conference proceedings and journals. The following list identifies those:

Journals and Conferences:

- Information and Software Technology (IST)
- Journal of Systems and Software (JSS)
- IEEE Transactions on Software Engineering (TSE)
- IEEE Software (IEEE SW)
- Communications of the ACM (CACM)
- ACM Computer Surveys (ACM Sur)
- ACM Transactions on Software Engineering Methodologies (TOSEM)
- Software Practice and Experience (SPE)
- Empirical Software Engineering Journal (EMSE)
- IET Software (IET SW)
- Proceedings International Conference on Software Engineering (ICSE)
- Proceedings International Symposium of Software Metrics (Metrics)
- Proceedings International Symposium on Empirical Software Engineering (ISESE)

Digital Libraries:

- ACM Digital Library (http://portal.acm.org)
- IEEEXplore (http://ieeexplore.ieee.org**)**
- CiteseerX Library (http://citeseerx.ist.psu.edu)
- ScienceDirect (http://www.sciencedirect.com)
- Wiley InterScience (http://www3.interscience.wiley.com)
- SpringerLink (http://www.springerlink.com)

Inclusion Criteria

Articles published until February 2010 were searched. Those articles should pass the following inclusion criteria to be included.

- [Q1] Systematic Literature Reviews, Surveys, Experiments, Case and Field Studies that are clearly identified as empirical researches containing empirical data and/or experiments on the field of agile software development and its practices.
- Papers of both students and professionals were included as long as they have been published in a well-known and broadly recognized journal and/or conference.
- [Q2] Papers that can help with the research on the research questions of this review.
- Papers that passed the minimum quality criteria (mentioned in Quality Assessment)

Exclusion Criteria

The following list contains the criteria with which some articles were excluded.

- Papers that haven't been published in a journal.
- Papers that didn't contain empirical data and were mostly theoretical

researches and/or expert opinions.
- Papers that discuss the agile software development process and its subcategories.

Primary Study Selection

Primaries studies will be selected by a single researcher and a list of the papers that have been accepted as well as a list of the papers that have been rejected will be kept.

Quality Assessment

Each paper will be assessed using the Critical Appraisal Skills Programme [66] and by the principles of good practices for conducting an empirical research in software engineering [67].
- *Rigour:* does the study follow a specific explained approach in the implementation of the various methods used in the study?
- *Credibility:* does the author discuss possible bias and findings and are the findings useful for the purposes of the study?
- *Relevance:* Are the findings useful for software companies and/or researchers?

Rigour:
- [Q3] The research method that was followed was explained as to why the specific one was used.
- [Q4] There is a description as to why the specific sample was selected and with what criteria.
- [Q5] A control group was used to compare the results of the study.
- [Q6] There is a description of the data collection methods, as to how the data were collected and why this specific method was used.
- [Q7] Data analysis methods were described concerning as to why those methods were chosen, how the data were selected and if they are enough

to answer the questions of the study.

Credibility:

- [Q8] Does the researcher identified his possible bias and the role he might have played to the research?
- [Q9] The results were discussed, they were identified if they answer the research questions and if the authors discuss the strength of their results.

Relevance:

- [Q10] Are the findings and results of the study useful and worthy for companies and/or scientific research?

Data Collection

<u>Study Descritpion</u>

- Study identifier: unique id for the study
- Date of data extraction
- Bibliographic reference: author, year, title, source
- Type of article: journal article, conference paper, workshop paper, book section
- Study aims: what were the aims of the study?
- Objectives: what were the objectives?
- Design of study: qualitative, quantitative (experiment, survey, case study, action research)
- Definition of agile software development given in study
- Sample description: size, students, professionals (age, education, experience)
- Setting of study: studies environment
- Data collection: how was the data obtained? (questionnaires, interviews, forms)
- Data analysis: how was the data analyzed? (qualitative, quantitative)

Study findings

- Findings and conclusions: what were the findings and conclusions? (verbatim from the study)
- Validity Limitations: threats to validity
- Relevance: research, practice

The data will be extracted by one researcher.

Data Analysis

The data will be concentrated and categorized alphabetically in tables by study type and author name, containing their basic information.
All the studies will be reviewed in order to answer the research questions mentioned above.

- *Can agile methodologies and its practices help software companies develop software projects more effectively than traditional methods?*

  The papers that contain empirical data of companies that have adapted agile methodologies will be reviewed. The success rate will be reviewed (if any) and if it depends on any factors. Also projects (if any) that haven't adapted agile methodologies successfully will be identified and the possible reasons behind that.

- *Can agile methodologies help universities with teaching computer science courses to students?*

  This review will try to identify studies that contain data that show the possible usefulness of agile methodologies to teaching computer courses to students. The opinion of those students will be checked and if agile methodologies can help for a better teaching experience.

Appendix B. Quality Assessment Table

ID	Research Design	Sampling	Control Group	Data Collection Methods	Data Analysis methods	Relationship	Findinds Discussion	Valuable Research	Total
1	1	0	0	0	1	0	1	1	**4**
2	1	0	0	0	1	0	1	1	**4**
3	1	1	1	1	1	0	1	1	**7**
4	1	1	0	1	1	0	1	1	**6**
5	1	0	0	1	1	0	1	1	**5**
6	1	0	0	1	1	1	1	1	**6**
7	1	1	0	1	1	1	1	1	**7**
8	1	1	1	1	1	0	1	1	**7**
9	1	1	0	1	1	0	1	1	**6**
10	1	1	1	1	1	0	0	0	**5**
11	1	1	0	1	1	1	1	1	**7**
12	1	0	0	1	1	0	1	1	**5**
13	1	1	0	1	1	0	1	0	**5**
14	1	0	0	1	1	0	1	1	**5**
15	1	0	0	1	1	0	1	1	**5**
16	1	0	1	1	1	1	1	0	**6**
17	1	0	1	0	1	0	1	0	**4**
18	1	1	0	1	1	0	1	1	**6**
19	1	0	0	1	1	0	1	1	**5**
20	1	0	0	1	1	0	1	1	**5**
21	1	1	0	0	0	0	1	1	**4**
22	1	0	0	0	0	0	0	0	**1**
23	1	0	1	1	1	1	1	0	**6**
24	1	1	0	1	0	0	1	1	**5**
25	1	1	0	1	0	0	1	1	**5**
26	1	1	1	1	1	1	1	1	**8**
27	1	1	1	1	1	0	1	1	**7**
28	1	0	1	1	1	0	1	1	**6**
29	1	1	1	0	1	1	1	1	**7**
30	1	0	1	1	1	1	1	0	**6**
31	1	0	1	0	1	0	1	1	**5**
32	1	1	1	1	1	1	1	1	**8**
33	1	0	0	1	1	1	1	1	**6**
34	1	1	1	1	1	0	0	1	**6**
35	1	0	1	1	1	0	1	1	**6**
36	1	1	1	1	1	1	1	1	**8**
37	1	0	1	1	1	0	1	1	**6**
38	1	0	0	1	1	0	1	1	**5**
39	1	0	1	1	1	1	1	1	**7**
40	1	1	0	1	1	0	1	1	**6**
41	1	1	1	0	1	0	1	1	**6**
42	1	1	1	1	1	0	1	1	**7**

43	1	1	1	1	1	0	1	1	**7**
44	1	0	0	1	1	0	1	1	**5**
45	1	0	1	0	0	0	1	1	**4**
46	1	1	1	1	1	0	1	1	**7**
47	1	0	0	1	1	0	1	1	**5**
48	1	0	0	0	0	0	1	0	**2**
49	1	0	1	1	1	0	1	0	**5**
50	1	0	0	1	1	1	1	1	**6**
51	1	0	1	1	1	0	1	1	**6**
52	1	0	1	0	1	0	1	0	**4**
53	1	0	0	0	0	0	1	1	**3**
54	1	0	1	1	1	0	1	1	**6**
70	0	0	0	0	0	0	0	0	0
Sum	**54**	**23**	**27**	**42**	**47**	**13**	**51**	**44**	