

# Test-Driven Development – Claimed Benefits and Results

Harri Korpi 49498D

**Abstract** — Test-Driven Development(TDD) is an important part of XP-Programming technique, it means roughly a way to write the test cases before writing the actual code. TDD has been claimed to have multiple benefits to the quality of the code through forcing programmers to get deeper into the problem-field by making them to plan and implement the unit test cases before implementing the actual code. These unit tests can be run automatically while building the program and all the test code should be run frequently. Old test cases should tell immediately if some old feature is broken while implementing a new one. The claimed benefits of the TDD-method concerns about the quality of the code, understandability, better design and reducing the development time. In this paper first all the researches and their approaches are shortly presented and then these claimed benefits are compared to the results obtained in the researches. Results showed that the other claims hold quite well but the development time didn't seem to have any significant differences between the control groups and the TDD-groups and also that there isn't still enough research done in the field.

**Index terms** — Test-Driven Development, TDD, Test-First Programming.

## 1. INTRODUCTION

The aim of this paper is to introduce all the different experiments done about the plain Test-Driven Development (TDD) method, present the used approaches and compare the results to the claimed benefits (detailed in the section 1.2) of the TDD. TDD method itself is only shortly presented, but it's mainly omitted.

While collecting the different researches I was surprised that there aren't more studies about TDD-method, I was able to find only three different experiments(one with only preliminary results) and one case study from the industry and one pilot study. Even when counted all together the researches are not very big so it's hard to do any bigger conclusions about the obtained results which is also the weakest point of this paper. Nevertheless the whole software engineering field is suffering the same problem due the fact that most of the studies are performed with relatively small (student-)groups to keep the budgets in a reasonable level. This unfortunate fact is also clearly visible in these researches about Test-Driven Development.

### 1.1 Test-Driven Development

TDD is not a new technique, but it has got more publicity lately due the reason that its one of the main elements in the XP-programming(XP comes from eXtreme Programming). TDD is basically a coding discipline where before writing the actual code the unit tests to the class/module/etc are written. The development process itself goes in small increments

including implementing the test cases, writing the code, passing the test cases and refactoring. What TDD basically does is that it moves the testing from the end of the project to be partly done at the same time with the coding. This moves the decision making and feedback of the tests also from the end of the project to the implementation phase. The claimed benefits are based on the idea that when programmers are forced to plan and implement the unit testing before the actually implementation of the module/class/etc they need to get better understandability of the problem field of the module to be implemented or actually they need to know exactly what they are going to implement. These test cases can be used later when the program is changed to tell if the old features are being broken by the new changes, the test cases can also be used to document the code because the test cases communicates the functionality of certain class or module. On the other hand the TDD has been claimed to promote for the better design through forcing the developers to write testable code, which should be better modulated and more independent of the other classes. And as it is always nothing good comes for free, in this case one can end up with huge amount of test code which needs to be maintained in a same way as the code itself and not to even mentioned the time spent to develop the test code. The main question in the TDD, as well as this paper, is if the benefits obtained are more valuable than the costs and what kind of proves there are behind these claims.

### 1.2 Research Approach

The approach of this relatively small research is a literature study about the Test-Driven Development method and the different researches done in the field about it. The results obtained in the researches are mirrored against the claimed benefits to answer to the basic question: "How well the results of the experiments map against the claimed benefits?". The question is divided below in four parts to make it easier to answer.

What is considered as a "claimed benefits" consists of Kent Beck's article Aim Fire (Beck, 2001) and some random articles found from the Internet (Dave Chaplin, 2003; Don Wells, 2000).

### The research questions are:

The experiments show that using TDD method leads to..

1. Higher code quality / fewer defects.
2. Simpler design / better modulation.
3. Better understandability of the problem field.
4. Reduce of the development time (in a long run).

In this paper first the researches are shortly presented and

these four questions are tried to answer based on the results obtained in the experiments and also the approaches used in the different experiments are shortly analyzed.

## 2. RESEARCHES AND APPROACHES

There are only three different researches done just about plain TDD(or which I was able to find) and one case-study done in the IBM company and one small pilot study. The researches are quite new which is surprising because TDD method itself is not new. TDD has been reported to be used in NASA already in the 60's in a project Mercury (Williams et al., 2002).

The first research was done by M.Muller and O.Hagner (Muller & Hagner, 2002) in the university of Karlsruhe, the second one was done by Body George and Laurie Williams (George & Williams, 2002) and the latest one was done in spring semester of 2003 by M.Panzcur's and M.Ciglaric's group in the university of Ljubljana(Panzcur et al., 2003). The IBM case study was done by E.M.Maximilien and Laurie Williams(Maximilien & Williams, 2003). The pilot study was done by R. Kaufmann and D. Jansen (Kaufmann & Jansen, 2003), but due to its "experimental" nature its approach is shortly presented but its results are not used in this paper.

Each experiment are presented below in the time order of their occurrence and all the text is based on the articles and the pictures are taken straight from the original articles.

### 2.1 M. Muller's & O. Hagner's Research

As mentioned earlier the first research about the benefits of TDD was done by M. Muller and O. Hagner. It was run with 19 graduated students in the University of Karlsruhe between July 2001 and August 2001(Muller& Hagner, 2002) as a part of an XP course.

#### 2.1.1. Approach

The research group of 19 students was divided into TDD-group of 10 people and control group of 9 people. All the students had previously taken an XP course with introduction to the XP practices including TDD and pair-programming. In the experiment TDD was separated from the XP so that both groups used more traditional software development methods except the TDD method used by the TDD-group. The work was to develop a main class for a graph library which included only the method declarations but no implementation. The basic approach was to measure the benefits of TDD used in a waterfall-like process with individual programming.

The experiment was divided in two phases: In the first phase (implementation phase) the individuals developed the work according to given requirements until they thought that it was ready. After the first phase an acceptance test was offered. In the second phase(acceptance phase) the work was continued until all the test cases of the acceptance test was passed.

The measured variables were:

1. The programming efficiency (development time)
2. Reliability of the solution
3. Program understanding.

Also because the test cases are in so central role in the TDD the coverage of the test cases were analyzed. Automated monitoring infrastructure was used to track the login-times and the collecting the different compiled versions and output of the runs. Quality was measured against JUnit's passed test-cases (which was naturally different than the acceptance test). Understandability was measured in means of number of reused methods, number of failed methods and number of failed methods used more than once. Development and problem solving time was measured by clock.

The amount of time used by the research group to accomplish the experiment was approximately between 500 and 800 min (~ 8-13 h) .

#### 2.1.2. Results

The final conclusions of the research were that by using the TDD method the programmers don't necessarily produce the code faster, but it pays back with the slightly increased reliability and the faster correct reuse of the existing methods. Based on the last one it seems that the overall understanding of the program was higher among the TDD-group.

The experiment was performed in two phases and the results were quite different between the phases: In the first phase the results of the TDD-group seemed to be less reliable and it took more time to develop but in the second phase the TDD-groups' program's quality grew significantly and they were considered more reliable than the control group's ones. Also there seemed to be less method calls that failed more than twice in the TDD-group's programs.

#### 2.1.3. Conclusions

As the authors concluded themselves the results are not reliable because of the size of the group was too small so that one could make larger conclusions about the results. Despite it the experiment's results showed quite clearly that there was hardly any connection between the decreased development time and used TDD-method, more likely vice versa, it showed that TDD-method takes even slightly more time than the traditional method used by the control group.

Also the improvement of the code quality seemed to be higher than in the control group's programs after the second phase but lower in the first phase. It was concluded that the TDD-method doesn't necessarily produce better code quality but the program's understandability seemed to get higher (measured with reuse of the code). Testers quality was measured with reference implementation of the same graph library and the results showed that quality of the tests of the TDD group were slightly lower than the control group's ones but it was concluded that the difference is so small that it was most likely obtained by change. Remarkable is that even thought the test quality was more or less the same the TDD groups' overall code quality was slightly higher after the second phase.

#### The results in a nutshell:

- TDD method takes little bit more time
- TDD leads to slightly more reliable code(After the 2<sup>nd</sup>

phase, quality was lower in the 1<sup>st</sup> phase)

- TDD leads to better understanding and correct reuse of the existing code

### 2.2 Body George and Laurie Williams

Body George's and L.Williams experiment was conducted with 24 professional pair-programmers in three different companies (George & Williams, 2002; George & Williams, 2004).

#### 2.2.1. Approach

The approach was to use pair-programming in a waterfall-like development process with professional programmers in their real working environments. The aim of the work was to develop a bowling game application based on the given specifications and they were asked to work until they thought that the work was ready. The approach used was different in the first company than in the rest two because after analyzing the results of the first experiment it was decided to change the approach. The reasons to change the approach were that in the first company the developers had a ready acceptance test and they seemed to stop working after the program passed it and also the researchers weren't happy about the way the errors were handled. Three things were done after the first experiment: the groups were emphasized to (1) handle the error conditions, (2) the control group was advised to write the unit-tests after the code and (3) acceptance test was not provided. Nevertheless only one pair from the control group implemented the asked test cases.

The research group of 24 (8 from each company) persons were divided into a 12 pairs by random selection inside the company. The groups used similar waterfall-like development process with the exception of the TDD-method used by half of the pairs. The difference between these was basically that the TDD group wrote test cases before the actual code where the control group used design-develop-test order (~Waterfall). Among the group the TDD and pair-programming experience varied from being daily used in the company to a novice; two of the companies used TDD and pair-programming in their day-to-day development and the in the third people had only 3 weeks experience before the experiment.

The hypothesis of the research were:

1. TDD yields to a superior external code quality compared to the water-fall like design.
2. Programmers will code faster with using TDD-method.

The quality of the code was measured with passed black-box test-cases written for the experiment and the time as an overall development time needed to accomplish the task. The size of the program was around 200 lines of code and the time spent for the experiment was between round 3 to 6 hours(see table 1).

#### 2.2.2. Results

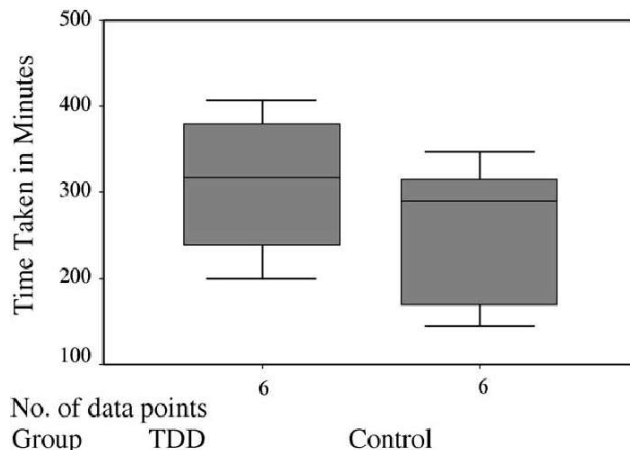
The results(see table 2) showed that TDD groups tend to pass more test cases (average 18% more, median was higher),

but it took 16 % more time to develop(with the limitation that the control group didn't write the instructed test cases). From this it was concluded that because every group thought that their application was ready when returned it the TDD programming leads to better understandability of the problem field.

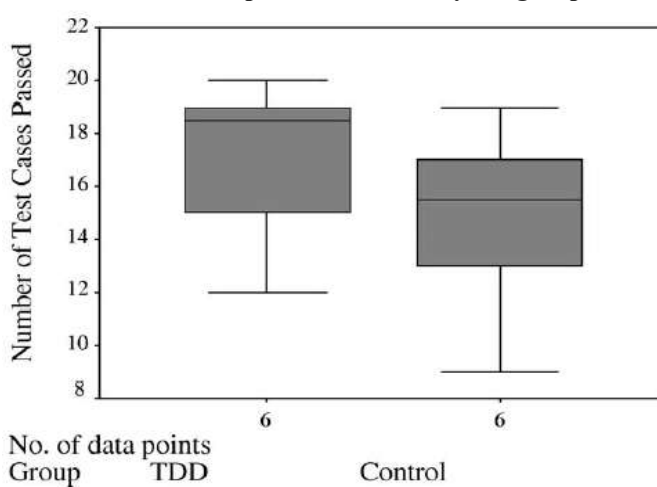
Also the test code coverage was measured and it was (for TDD group) mean of 98 % method, 92 % statement and 97 % branch coverage, which are slightly higher than the industrial standard(80-90 %). After the experiment feedback was received from the developers. It was analyzed and it showed that: 87,5 % believed that the TDD leads to the better requirements understanding, 95,8 % believed that it reduces the debugging effort. Only half believed that it leads to decrease of the development time and 92 % believed that it yields to higher code quality. Remarkable was that 56 % thought that adaption the TDD mindset was difficult but still 71 % kept the method noticeably effective. These results of the feedbacks were compared in Pancur's study (Pancur et al. , 2003) to the results he obtained in his study with students and it is shown in table 3 below.

The results showing the distributions of the development time and the passed test cases in the survey are presented in the Tables 1 and 2 (George & Williams, 2004).

**Table 1 – Development time spent by the groups**



**Table 2 - Number of passed test cases by the groups**



### The results in a nutshell

- TDD takes 16 % more time
- TDD passes 18 % more test cases
- Leads to better understandability

#### 2.2.3. Conclusions

As the authors mentioned the sizes of both the experiment group and the program were too small to give a reliable results or to make any bigger conclusions but compared to the claimed benefits the results didn't seem to show that TDD decreases the development time, but it seemed to support the claim that the TDD leads to the better understandability of the problem field and also it seemed yield relatively high code coverage of the test-cases. The feedback from the developers showed that they thought that the TDD is efficient method in reducing the defects from the program. Also the fact that the experiment was changed after the first company is also a weakness of this experiment.

#### 2.3 M. Panzcur, M. Ciglaric, M.Trampus and T.Vidmar

The experiment was run between February to June 2003 in the University of Ljubljana and it is part of M.Panzcur's PhD studies(Panzcur et al., 2003). I was able to find only the preliminary results so unfortunately the experiment cannot be completely presented here.

##### 2.3.1. Approach

The experiment was run with 38 senior undergraduated CS students, who were divided into a TDD group of 20 and the control group of 18 students. Both of the groups used agile methods: control group's method was called ITL(Iterative Test Last). The difference between the groups was only that the TDD group wrote the test code before the parts to be implemented. Both of the methods could be considered as agile methods with the short cycles: code-test-refactor for the ITL-group and test-code-refactor for the TDD-group. The research questions were that how the writing test cases before the code influences to the (1)development process and to the (2)resulting code. Special attention was put in dividing the groups equally using past academic performance in balancing the members of the groups.

The experiment was run in two parts, the first part consisted of three smaller and one bigger(~3 weeks) exercises which were done in pairs. The second part consisted of a half day's exercise which was done individually. The tools used in the experiment were Java, Eclipse 2.1 and JUnit also the development time tracking infrastructure(Eclipse plug-in) was used in the second phase where in the first phase the development time was reported by hand. The research group didn't have previous experience about TDD and they hadn't written hardly any tests to their code before.

##### 2.3.2. Results

Because the experiment was run quite recently the only results that have been published concerns about the coverage of the test cases after the two first assignments in the first

phase, the feedback from the students and the external code quality of the first assignment.

The coverage of the test-cases after the first two assignments showed that both of the groups were able to yield higher coverage than the industrial standard(80-90%), but TDD groups results were slightly below the control groups' ones. The only exception of this was the method coverage which was 100 % for the TDD group. The mean values of all the measured code coverages(method-,statement-,branch coverages) were 92,6 % for the TDD group and 95,1 % for the ITL group. Also the external code quality was measured in the first assignment and it showed that the TDD groups' programs passed fewer test cases than the control groups' (mean 92,6 % TDD and 95,1 % ITL). But as the authors concluded the differences were so small that it is most probably by change.

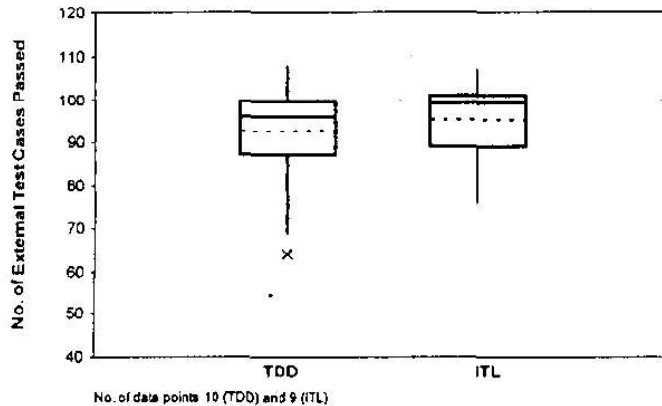
Also the feedback from the developers were analyzed and it showed different results from those in Body George's and Laurie Williams'(George,B;Williams,L, 2003) results, the differences are presented in table 3 below (originally table 1 in Panzcur's paper). The main difference was that the developers didn't find TDD as powerful method as the professionals in George's and Williams' research and the main common factor was that 63,2 % of the TDD group thought that it was hard to adopt the TDD's mindset. Also as mentioned before the test group hadn't tested their code before so it could be that they thought that writing the tests are not efficient in general and this could explain the difference in the opinion about the efficiency.

**Table 3 - feedback from the developers compared to the Body George's and Laurie Williams' results:**

	TDD GROUP	ITL GROUP	B&W TDD
TDD/ITL facilitates better requirements understanding.	63,2	60	87,5
TDD/ITL reduces debugging time.	57,9	53,3	95,8
TDD/ITL reduces code development time.	21,1	33,3	50
TDD/ITL yields higher code quality.	77,8	73,3	92
TDD/ITL promotes simpler design.	47,4	40	79
TDD/ITL is noticeably effective.	26,3	40	71
Getting into the TDD/ITL mindset is difficult.	63,2	20	56
Lack of upfront design phase is a hindrance.	73,7	60	23

From the Table 3 it can also be seen that even though Panzur's test group didn't believe that TDD is an effective method of coding they still seemed to believe that it leads to higher code quality(77,8 %).

Table 4 shows the amount of passed test cases after the first exercise and it seems that the ITL group has slightly more passed test cases but the difference don't seem to be significant.

**Table 4 – Test cases passed in the first assignment**

### 2.3.3. Conclusions

The preliminary results don't show any significant differences between the two groups but only a small portion of the collected data has been analyzed and published. The internal quality (measured with passed test cases) are presented in the table 4 above and as one can see the TDD groups seem to pass slightly fewer test cases, but the difference was concluded not to be significant and "is most likely due to change". On the other hand as in M. Muller's and O. Hagner's research (Muller & Hagner, 2002) it happened that TDD group had lower quality of the code in the beginning but it seemed that they got better results later, but these results are not available yet so it might be too early to speculate it.

Also the testers quality was measured and again as in Muller's study (Muller & Hagner, 2002) the control group yielded slightly higher code coverage, but this difference was concerned not to be significant. Both of the groups yielded relatively high code coverage (~above 90%).

It is also worth to notice that the results are only from the first phase of the experiment when the students work individually and they use manual time tracking, in the second phase they used pair programming and automated time tracking tool. These might bring some extra variation too to the final results of the experiment.

### 2.4 Other Studies about TDD

In this section I will present studies that are not considered as an experiment in that way as the previous ones. This section starts with short presentation about the IBM business case done by E. Maximilien and L. Williams (Maximilien & Williams, 2003; Williams et al., 2003) which offers some contrast between the sizes of the experiments presented above and the sizes of the normal projects in the industry. I was able to find also a small pilot study about TDD which is shortly presented in the section 2.4.2, but as mentioned before its results are not used in this paper. The pilot study was performed by Kaufman and Janzen in 2003 using 8 college students (Kaufmann & Janzen, 2003) divided into two four people groups. These groups measured against similar team using more traditional development process but this study is

cut outside of this paper's scope due to its "preliminary" approach, this study is however shortly presented but its results are mainly omitted in this paper.

#### 2.4.1. Case study at IBM

The study was performed by E. Maximilien and L. Williams (Maximilien & Williams, 2003; Williams et al., 2003) and it was performed in a real project at IBM. This presentation is mainly based on the article from Williams, Maximilien and Vouk (Williams et al., 2003).

##### 2.4.1.1 Approach

The case at IBM was that they had been developing a similar JavaPOS (Java Point of Sale) access points for different devices such as printers, cash drawers and bar code readers on various operating systems for over three years. Despite the growth of the experience of the area the defect rate in the Functional Verification Test (FVT ~System Test Phase) was not reduced as expected, because of this the development and the management teams were open to new approaches and the TDD-method was suggested.

The team was size of 9 engineers which worked in two different locations (5 in US and 4 in Mexico) and none of them had previous experience about the TDD-method. The results obtained with TDD were compared to a legacy project of the similar product: differences in the teams are presented in the table 5 below. The main difference with their processes were the unit testing approaches which was changed from ad hoc-based to TDD approach in the new project. It is noticeably that the legacy-group had already more complex base to start from than the new TDD-group's project so it shouldn't be seen as a pure control group.

**Table 5 – Team and product comparison**

	Legacy 7 <sup>th</sup> Iteration	New 1 <sup>st</sup> Release
Team Size (Developers)	5	9
Team Experience (Language and Domain)	Experienced	Some Inexperienced
Collocation	Collocated	Distributed
Code Size (KLOC) New; Base; Total	6.6; 49.9; 56.5	64.6; 9.0; 73.6
Language	Java/C++	Java
Unit Testing	Ad hoc	TDD
Technical Leadership	Shared resource	Dedicated coach

In the TDD approach a strategy for writing the test cases was created so that each design document defined a unit testing section about the public classes and methods that would be tested (JUnit's test cases). The goal was to yield over 80 percent test code coverage for those important methods and to make sure that the tests are run frequently the tests were automated so that they were run every day and the results of the runs were sent by email to all the developers. The total size of the project was 73,6 KLOC (64,6 KLOC of new code) for the program and 34 KLOC of test code. Compared to the

previously presented experiments the size different is significant.

The research questions were compared to the legacy project to see how the TDD approach affects to the:

1. Defect Rate
2. Productivity
3. Test Frequency, measured with the ratio of the interactive vs. automated test cases(Omitted in this).
4. Design
5. Integration(Omitted in this), means basically the approach that the product was build daily and the test cases were run.

#### 2.4.1.2 Results

The impact to the defect rate(1) was significant, it was estimated based on the historical data that the defect ratio would be 8 defects per 1 KLOC but after the FVT it was shown that the defect rate was 4 defects per KLOC (Maximilien & Williams, 2003). Defect rate was also compared to the legacy project it seemed that the TDD group had 40 % less defects in the FVT. Development time(2) showed slightly decrease in the development time but it was considered as “roughly” the same because the difference was not considered significant. Design(4) was also believed to be simpler and easier to change; couple of devices were added to the system and their projects succeeded in about two-thirds of the time scheduled. The comparison of the results in the FVT is shown in the table 6. As it can be seen from the table the Legacy-project had to execute the FVT phase twice. In general the developers seemed to like the TDD method and they continued to use it in their next projects(Maximilien & Williams, 2003).

**Table 6 – Results Comparison**

	Legacy 7 <sup>th</sup> Iteration	New 1 <sup>st</sup> Iteration
FVT Effort	E	0.49E
Test Cases Run	TC	0.48TC
Test Cases/Total LOC	TCL	0.36TCL
Defects/Test Case	DTC	1.8DTC
Defects/LOC	DFL	0.61DFL

Table 6 basically shows that the legacy project had to put twice more effort on testing and the testing in the TDD group was more efficient but was this the obtained because of the TDD method or the fact that the legacy-group had to build their changes with huge amount of the previous code was omitted in the results.

#### 2.4.2 Reif Kaufmann and David Janzen

R. Kaufmann and D. Jansen (Kaufmann & Jansen, 2003) did their preliminary study about using TDD in the Bethel College in the spring semester 2003. This study is only shortly presented here but its results are not taken account in the final

results of this paper due its “preliminary” or “pilot” approach presented below.

#### 2.4.2.1 Approach

This pilot experiment was run with two groups each of four members and it was run under a course called “software studio”. Both of the teams developed some game using Java as a programming language and none of the people didn't have previous experience about writing games with Java and about TDD. The teams and their works were self-selected, so the results are not straightly comparable(but this was a pilot study as mentioned before). The aim of the study was to get preliminary results of effects of the TDD method on (1) the quality of the code, (2) programmer productivity and (3) programmer confidence. Quality and productivity were measured with using tools such as: JavaNCSS, JDepend, JMetric and CCCC. They were used to count the variables like size of the programs, non-commented lines of code, functions per class and complexity numbers counted by the tools (McCabe's Cyclomatic Complexity, etc). Confidence was measured by a survey arranged in the end of the course.

#### 2.4.2.2 Results

The results showed that the TDD group developed 50 %(2) of more code but still the complexity numbers were the same. Based on this the size of the application was considered as a reliable variable for measuring the productivity. On the other hand the TDD group had more programming experience so it couldn't be said to depend on straight from the use of TDD. The tools showed that both of the groups had problems with their design(2) but it was concluded that the project was unrealistic in a way that the students knew that they didn't need to maintain the code after the course. The survey showed in the end that the TDD group's members were more confident to their solution (from 1-5, average 4,75) than the control group (av. 2,5). TDD group also believed using the same scale that TDD helped their design and debugging with average 4,25.

#### 2.4.2.3 Conclusions

The main problem of this study is that is was meant to be preliminary: Both teams developed a different application and no attention was put on balancing the groups according to the experience. Based on that the results are not reliable and that's why this study is limited out of the scope in this paper. Also the size of the study would have been too small to give reliable results(tot. 8 people and 2 teams). Also in this study even the TDD group failed to follow the TDD's guidelines and they ended up with only 16 JUnit's assertions. Basically the only result of this would be that the developers believed that the TDD method helps them in the design and debugging phases.

### 3. ANALYSIS OF THE RESULTS

In this chapter the different approaches are analyzed and then the results are mirrored against the research questions set in the first chapter.

### 3.1 Different approaches

Table 7 presents the different approaches adopted in the different experiments. As one can see all the studies had different approaches in the researches: Muller and Hagner (Muller,Hagner, 2002) did their experiment with undergraduated students with a small program which was developed in two phases. They used Waterfall-like development and the work was done individually. B. George and L. Williams(George & Williams, 2003) performed their experiment with professional programmers in the industry using pair programming but also with quite small program. Pancur's (Pancur et al., 2003) group did their experiment with undergraduated students, they used both pair programming and individual works in two phases, but only the preliminary results are available from the experiment, and maybe in their approach they tried to measure multiple different variables which might cause some extra variation to their results (Individual/Pair programming, Manual/Automatic time tracking). Different in the Pancur's approach was that the control group were using a Agile method as well(method was called ITL=Iterative Test Last).

Maximilien and Williams(Maximilien & Williams, 2003) did their IBM case study in a real project in the industry from a real customer need with a distributed team, but quite a few words were mentioned about the other processes they used. Nevertheless it is quite obvious that the projects in the real industry are huge compared to the projects done in the experiments. The pilot experiment by Kaufmann's and Jansen's study (Kaufmann & Jansen, 2003) had different approach by experimenting the use of TDD in a group work which no one hadn't done before but unfortunately their research was meant to be a pilot version so it's results are not included to this paper.

**Table 7 – Different Approaches**

	Muller& Hagner	George& Williams	Pancur,Ciglaric, Trampus,Vidmar	Maximilien & Williams: IBM
Group size	19	24	38	9
Group type	Students	Professionals	Students	Professionals
Process type	Individuals	Pair	Individual / Pair	?
Process	Waterfall	Waterfall	Agile(ITL)	TDD
Size of the experiment	500-800 min	200 LOC	?	73 600 LOC

Especially one should notice the difference of the sizes between the experiments performed in the academic world compared to the case study in the IBM and also it's good to notice that the IBM study didn't have a pure control group so it's hard to make any valid comparison about it.

### 3.2 Results and conclusions

The aim of this literature study was to examine how the different researches support the claimed benefits of the TDD. The summary of the results obtained in the different studies is presented in the table 8 after presenting the results. The research questions set in the chapter 1.2 were to examine how

the results of experiments support the claims that:

The experiments show that using TDD method leads to..

1. Higher code quality / fewer defects.
2. Simpler design / better modulation.
3. Better understandability of the problem field.
4. Reduce of the development time (in a long run).

- *1. Higher code quality / fewer defects:* Table 8 shows roughly the results from different studies from where it can be seen that the first claim(1) that the TDD method leads to higher code quality is supported in all the cases except the Pancur's study and Muller's first phase. When considered that the Pancur's results are also from the first assignment it can be concluded that the question is supported with the exception that it takes time to adapt the TDD method and in the beginning the results might be false, but when the experience grows this seems to get support. Among unexperienced programmers it might be misleading method with false sense of reliability. Also from table 3 it can be seen that the about half of the developers both in Pancur's and Williams' studies thought that it was hard to get in the TDD's mindset which tells so that the developers might have difficulties in the beginning.

Also Matthias Muller and Frank Padberg (Muller & Padberg, 2002) have proposed a formula to be used when counting a return of investment while using the Test-Driven Development which is based on Muller's and Hagner's experiment(Muller & Hagner, 2002) described earlier. It suggests that the TDD method takes more time to develop the program, but it would pay back with increased quality and life-cycle benefits with reducing the bug-rate while changing the code(Notice that this is clearly visible in the IBM-case). Overall the researches seemed to support the idea that the TDD method leads to higher quality of code with the exception that in the beginning the results might be even worse than normally. This thing should be realized while starting a project with unexperienced people and solution could be training, courses, etc.

- *2. Simpler Design / better modulation:* Not all of the researches were examining this claim. It was maybe due the fact that it might be hard to measure especially in small experiments. However in Muller's study it was shown that the TDD group yield better("correctly faster") reuse of the existing methods which could be a hint of a better design. The IBM case seemed to support the idea and in B. George's and L. Williams' study(George & Williams, 2003) they concluded that "Qualitatively, this research also found that TDD approach facilitates simpler design..". The original claim is based on the idea that when programmers are forced to plan the testing first they need to write testable code, which needs to be independent and simple. In the researches this point was mainly omitted or then just concluded that the design got better. The feedback from the developers in George's (George & Williams, 2003) showed that the professional developers believed that the claim is true(79 %). Based on these it can be concluded that TDD

leads to simpler design through better reuse of the existing methods, though it must be mentioned that the “better design” is a quite abstract concept and it cannot be perfectly measured.

- *3. Better understandability:* This was mainly measured with the reuse of the existing code and it was included to George's & Williams'(George & Williams, 2003) and Muller's & Hagner's (Muller & Hagner, 2002) studies. Both of them seemed to support this claim. In George's and Williams' study this was concluded based on the idea that the TDD group tended to work longer with the given task so they had to think that the work wasn't still finished and based on that it was concluded that they had better understanding of the problem field. In Muller's and Hagner's study this was counted that the TDD group used “correctly faster” existing code than the control group. This claim is then also considered as satisfied.
- *4. Reduce of the development time(in a long run):* Development time was measured in all of the studies and the results varied little bit but in general this claim didn't seem to get hardly any support. George's & Williams' study (George & Williams, 2003) showed 16 % increase of the development time while other studies didn't seem to find any significant differences between the time spent by the control- and TDD groups. On the other hand in George's and Williams' study the control group didn't write the asked unit tests so this could explain the differences between the development time. Also it could be claimed that all the experiments were too small so that the all the benefits of the TDD (Test code as documentation, old test cases to verify that the old code is not broken etc) cannot be accomplished in such a small projects. The case study in the IBM however gave information about the TDD method used in a big project but it couldn't find any significant differences between the development time neither. In the IBM case it was mentioned that the code was modified later to include two drivers more and those projects were accomplished in two thirds of the time scheduled for them, if this could be considered being the result of using TDD that would support the claim but nevertheless it wasn't mentioned. Anyways this claim was not supported and even vice versa the results seemed to give hint more that the TDD method doesn't necessarily produce significant overlay to the project, at it's best. On the other hand the long run benefits were not included to these experiments.

**Table 8 – The summary of the experiments:**

Research	Quality	Understanding	Development Time	Tests Coverage
<b>Muller&amp;Hagner</b>				
- First Phase	Lower	-	~same	-
- Second Phase	Higher	Higher	~same	Lower
<b>George&amp;Williams</b>	Higher(18 %)	Higher	Higher(16 %)	** High
* <b>Pancur, Ciglaric, Trampus, Vidmar</b>	~Same	-	Not published	~Same
<b>Maximilien, Williams, Vouk</b>	Higher(40 %)	-	~same	?

\* Preliminary results. \*\* Control group not measured.

There were also other variables measured in the results of the experiments so these are also shortly analyzed here even though they are outside of the scope of this research. These variables were:

- *Code Coverages:* First of all the quality of the unit testing could be considered as a strongly dependent variable of the code coverage obtained in the unit testing. Significant was that in all the studies where the testers quality was measured the TDD group yielded slightly less code coverage than the control groups(Higher than the industrial standards anyways). This can tell that the TDD groups couldn't see all the exceptional cases before writing the code, but on the other hand TDD groups still tended to have higher internal code quality in most of the cases. It could be also claimed that in the experiments the people tend to follow the instructions more precisely than in the real life, which could be continued after B.George's and L. Williams' IBM case when the professionals didn't even follow instructions given in the experiment. So at least from my personal experience the unit tests are the first ones one seem to skip if it's not absolutely necessarily and in these cases TDD offers a discipline, just as Kevin Beck (Beck,K; 2001) has stated “Test-Driven Development is not about testing”, it more a discipline which makes sure that the test cases are written and that the basic meaning of the item under work is tried to be understood before implementing if you can do it without the TDD good, if not then TDD will force you to do it.
- *Feedback from the developers:* Table 3 shows the differences between the feedback from the developers in the Pancur's and George's studies. The differences can be mostly explained by the fact that the Pancur's students hadn't tested their code with unit tests before and they could have thought that the unit testing is not efficient in general. The thing which is more important is to see the similarities of the question about hardness of adapting the TDD mindset which is about half of the programmers in both cases and also quite strong believe that TDD method leads to better quality of code.

#### 4. DISCUSSION

In this paper the main focus was in the studies performed in the academic world but it would have been nice to found more case studies from industry because they could offer more information about the long-term benefits of the TDD method. The results showed with the exception with development time the claimed benefits hold quite well, and it could be still claimed that the development time will decrease in a long run when the bug-fixing and the lifetime benefits save resources. Also none of these experiments could offer reliable data so these conclusions made are also far of being reliable but on the other hand they were based on multiple studies and all of them seemed to give support to some of the claims.

Also it can be concluded that the adaption of the TDD method might be difficult and it can take some time before it is clear and good enough understood. It can be also claimed that TDD's benefits would be also obtained by using some other methods like ITL used in Pancur's(Pancur et al., 2003) experiment, but the thing that TDD offers is a disciplined way of making sure that everybody really does the unit test cases and that they are run frequently(not to even mentioned the need for good planning and understanding the problems before coding).

From the researches it can be seen that the TDD-groups tend to have worse results in the beginning of the projects but then their quality will improve significantly later in the projects and the internal quality tends to overcome the quality of the control team, this also gives hints that it takes time to adopt the TDD-method but the cost can be paid back with improved quality later in the project. This hardness of getting in the TDD's mindset should be taken into account before starting the projects for example through offering education or courses about the method before starting to use it in the companies.

## 5. SUMMARY

In this paper three different experiments and one case study (plus one pilot study presented) of Test-Driven Development were presented and their results were mirrored against the claimed benefits of the method. These questions were:

The experiments show that using TDD method leads to..

1. Higher code quality / fewer defects.
2. Simpler design / better modulation.
3. Better understandability of the problem field.
4. Reduce of the development time (in a long run).

Based on the results of the different studies it was concluded that support was found for the first three but the fourth claim didn't seem to get any support even vice versa but because most of the studies didn't seem to find any significant differences between the development times between the TDD and control groups it was concluded that "TDD method doesn't necessarily produce significant overlay to the project at it's best". These results however cannot be taken as a reliable fact due the reasons that all the studies presented are not reliable and on the other hand they all have little bit different approaches for the experiments. Even though this paper cannot be seen as a reliable truth of the TDD method I personally hope that it gives a good understanding of the research situation in the field of TDD so that it easy to see which different approaches has been taken and which results have obtained and in which way these results could be combined together. Also it is worth to notice that there aren't big enough studies about the method so that one could keep its results reliable and also the IBM study is the only where the method has been reported to been used in a real big project in the industry, especially these experiences would have been nice to read. Also the TDD's problem areas such as User Interfaces and Databases were nicely left outside of these studies.

## 6. REFERENCES

- Dave Chaplin, 2003,"Test-Driven Development",  
<http://www.bytevision.com/TestDrivenDevelopmentArticle.aspx>
- Don Wells, 2000,  
<http://www.extremeprogramming.org/rules/testfirst.html>
- Beck, K., Aim Fire. IEEE 2001, vol. 18, no 5. pp. 87-89.
- Boby George, Laurie Williams, March 2003, "Software engineering: An initial investigation of test driven development in industry", Proceedings of the 2003 ACM symposium on Applied computing
- Boby George, Laurie Williams, April 2004, "A structured experiment of test-driven development", Information and Software Technology, Volume 46, Issue 5, 15 April 2004, Pages 337-342
- Muller, M.M and Hagner, O, 2002, " Experiment about Test-first Programming", proc. of confrence on empirical assessment in software engineering EASE 2002
- Pancur, M.; Ciglaric, M.; Trampus, M.; Vidmar, T., Sep 2003, "Towards empirical evaluation of test-driven development in a university environment", EUROCON 2003. Computer as a Tool. The IEEE Region 8 , Volume: 2 , 22-24 Sept. 2003 Page(s): 83 -86
- E.M.Maximilien, L. Williams, 2003: "Assessing Test-Driven Development at IBM", IBM Corp.; Proceedings. 25th International Conference, page(s): 564- 569, ISSN: 0270-5257
- L.Williams,E.M.Maximilien,M.Vouk,2003: "Test-Driven Development as a Defect-Reduction Practise" , IEEE 2003, Proceedings of the 14th International Symposium on Software Reliability Engineering (ISSRE'03)
- Matthias Muller, Frank Padberg, 2003: "About return on Investment of Test-Driven Development", In International Workshop on Economics-Driven Software Engineering Research (EDSER), Portland, Oregon, USA, May 2003
- Reid Kaufmann, David Janzen, 2003: "Implications of test-driven development: a pilot study" , Pages: 298 – 299, ISBN:1-58113-751-6, Publisher ACM Press New York, NY, USA