An Experimental Evaluation of Test Driven Development vs. Test-Last Development with Industry Professionals

Hussan Munir, Krzysztof Wnuk
Department of Computer Science
Lund University, Sweden
hussan.munir@cs.lth.se
krzysztof.wnuk@cs.lth.se

Kai Petersen, Misagh Moayyed
School of Computing
Blekinge Institute of Technology, Sweden
Kai.petersen@bth.se
mm1884@gmail.com

ABSTRACT

Test-Driven Development (TDD) is a software development approach where test cases are written before actual development of the code in iterative cycles. **Context:** TDD has gained attention of many software practitioners during the last decade since it has contributed several benefits to the software development process. However, empirical evidence of its dominance in terms of internal code quality, external code quality and productivity is fairly limited. **Objective:** The aim behind conducting this controlled experiment with professional Java developers is to see the impact of Test-Driven Development (TDD) on internal code quality, external code quality and productivity compared to Test-Last Development (TLD). **Results:** Experiment results indicate that values found related to number of acceptance test cases passed, McCabe's Cyclomatic complexity, branch coverage, number of lines of code per person hours, number of user stories implemented per person hours are statistically insignificant. However, static code analysis results were found statistically significant in the favor of TDD. Moreover, the results of the survey revealed that the majority of developers in the experiment prefer TLD over TDD, given the lesser required level of learning curve as well as the minimum effort needed to understand and employ TLD compared to TDD.

Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Software/Program Verification—*Reliability, Statistical methods*; K.6.4 [**Management of Computing and Information Systems**]: System Management—*Quality Assurance*

General Terms

Reliability, Verification

Keywords

Test-Driven Development, TDD, Test-Last Development, TLD, Productivity, internal or external code quality.

1. INTRODUCTION

The most significant aspect of software development is software quality [1]. The majority of the agilists have considered test-first approach as the preferred development methodology where they write the test case before they actually write the domain code to pass unit tests [1]. Test Driven Development (TDD) has been practiced in the past but it has become a well known activity from the last decade in the form of Test-Driven Development (TDD) [1]. Initially it was considered as a key practice of Extreme programming(XP) [2] but later on it was considered as a standalone process [3] [4]. The single most important rule in TDD is: "If you can not write a test for what you are about to code, then you should not even be thinking about coding" [5]. As a consequence, in the TDD process the tests are written before the production code is written, see Figure 1. After that, both TDD and TLD contain the refactoring activity to change the production and test code and make it as simple as possible, ensuring that all tests pass. Both methods repeat steps mentioned in figure 1 until the feature or user requirement is implemented. TDD allows the developers to think ahead of the functionality before writing the code, to provide the detailed design just in time, to have the tests to validate their work before implementation, to enable early code refactoring and maintenance [5, 6].

However, from the TDD experiments done mainly on students, the benefits of TDD in terms of productivity, code quality and the number of tests passed seem to be small and associated with the increased effort cost in most cases [5] [8] [9] [10] [11] [12] [13]. The studies involving practitioners report no significant differences in terms of productivity and branch coverage in one case [14] and significant differences in terms of productivity and external code quality in the other case [15]. The industrial case studies about TDD suggest reduced defect rate [16] and increased code quality [8] for the price of increased effort [8, 16]. Moreover, a recent literature review on TDD identified only 9 studies on this topic conducted with high rigor (complete description of methodology, rigorous conduct of studies) and high relevance (application in realistic context) [17].

In this paper, we report the results from an experiment

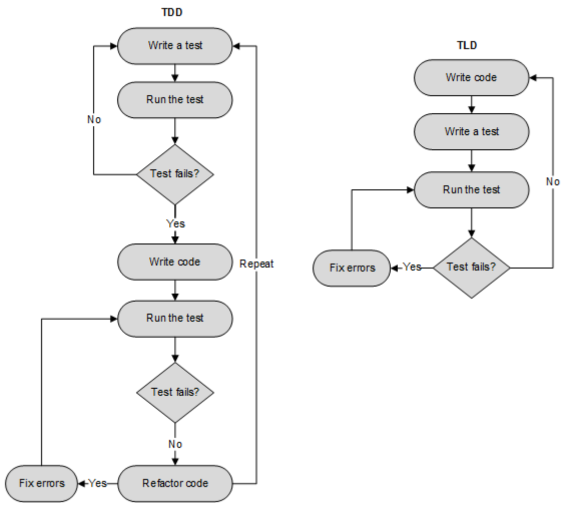

Figure 1: TDD Vs. TLD (based on [7])

with practitioners designed to address the issues of low rigor and relevance and to further investigate the potential benefits of TDD. The following research questions are investigated in this study:

- RQ1: Does TDD improve external code quality compared to TLD?
- RQ2: Does TDD improve internal code quality compared to TLD?
- RQ3: Does TDD improve productivity compared to TLD?
- RQ4: Does developers prefer TDD over TLD (developers opinion)?

External code quality refers to the delivered quality [18], indicated primarily by passed test cases and number of defects (See table 1). On the other hand, *Internal code quality* deals with the code quality in-terms of code complexity, test coverage, branch coverage, coupling and cohesion between objects [17]. Finally, *Productivity* deals with relationship between output and input [18]. This paper is structured as follows: Section 2 provides related work while Section 3 provides the research methodology. The study results and their analysis are presented in Section 4 and discussed and concluded in Section 5.

2. BACKGROUND AND RELATED WORK

TDD seems to be a suitable methodology for rapidly changing contexts and thus good support for the agile software development methodology [6]. There are many motivations behind adopting TDD in agile development. First, it allows the developers to think ahead of the functionality before writing the code and provides detailed design just in time. Second, it ensures that developers already have the tests to validate their work. Third, it provides with the chance to refactor their code early and maintain the highest quality of the code [6]. Furthermore it also prevents the introduction of new bugs in debugging and helps in easy maintenance of software by continually running automated test cases [5].

Most of the reported evidence on Test-Driven Development belongs to the following aspects of the software engineering: 1) Internal code quality, 2) External code quality, and 3) Productivity [12]. In several student experiments with same research settings, (with particular focus on internal and external code quality as well as productivity measures), TDD makes no significant difference in external code quality [9] [11] [13]. Muller et al. [9] conducted an experiment with the computer science students and compared Test-Driven Development (TDD) with Test-Last Development (TLD). The results revealed that TDD or TLD does not accelerate the implementation process and does not make resulting programs reliable either, though TDD gives evidence of more readable and maintainable program code compared to the TLD approach. Pancur et al. [19] also conducted a controlled experiment with undergraduate students to compare TLD with TDD and they found that TDD does not carry any substantial difference from TLD approach in terms of productivity and code quality. Erdogmus et al. [13] also conducted an experiment with undergraduate students to analyze the effectiveness of Test-First approach and minimum productivity and quality is observed in case of Test-First development.

George et al. [20] conducted a controlled experiment with professional programmers. They compared TDD with waterfall development approach and found that TDD developers passed 18 % more functional black box test cases compared to waterfall developers. However, TDD developer took 16 % more time for development. Moreover, Gupta et al. [10] conducted an experiment with undergraduate and graduate students of computer science and found that TDD improved both code quality and productivity and also reduced the overall development effort. In addition to that, Huang and Holcombe [5] found TDD group passed more acceptance tests than the Test-Last group but the trend related to time used for unit test rather than development method. In another study conducted by Canfora et al. [21], they performed a controlled experiment with professionals with a focus to identify whether or not TDD improves the unit testing. In their results, they revealed that it improves the overall unit testing but slows down the development process.

In a recent study conducted by Pancur et al. [11] also concluded that benefits of test driven development are small compared to TLD although the effects are positive in terms of code quality and productivity and there is need for more empirical evidence. Moreover, the study results have not shown statistically significant difference between TDD and TLD regarding productivity and acceptance tests passed. On the other hand, there are number of case studies performed for the evaluation of TDD and their results are quite positive compared to controlled experiments.

In a case study conducted at IBM, Maximilien et al. [16] built a software system using TDD. The results showed that they have managed to reduce the defect rate by 50 % compared to a similar system that was developed using ad-hoc unit testing, a technique used to test program methods interactively. Additionally, the test case suite developed during TDD is reusable and an extendable asset that helps in improving the code quality over the lifespan of software. However, the project was completed with minimum productivity impact [16]. In another study at Microsoft, Bhat et al. [8] applied TDD in two different environments, Windows and MSN division. They observed the significant increase in the code quality for both projects developed using TDD approach compared to similar projects developed in non-TDD fashion. However, the project took at least 15 % extra upfront time for writing the test cases.

The motivation behind conducting an experiment is that the majority of previous experiments [8] [9] [10] [5] [11] [12] [13] is conducted with students who are often not familiar with the use of TDD. Consequently, it becomes difficult to generalize the results gathered through student experiments. Case studies [16] [8] [22] [23] [24] showed the significant improvement in terms internal code quality, external code quality and productivity in the favor of TDD. However, only one experiment [10] showed the statistically significant results in the favor of TDD. Therefore, the results are found to be inconsistent and one of the reasons could be these most experiment were performed with students including the study that got significant result. For that reason, we designed an experiment with professionals to evaluate if experiment results reported in previous studies conforms to case studies results reported in related work but also to lay the foundation for future studies in the area [17].

3. RESEARCH METHODOLOGY

In this study, a controlled experiment is performed according to the guidelines provided by Wholin [25]. The random design with one factor and two treatments was adopted [25]. The experiment was designed in a way so that each subject uses only one treatment on one object [25]. The experiment investigates four research questions, see Section 1.

Research questions RQ1, RQ2 and RQ3 are answered with the help of the experiment and RQ4 is answered with the help of a survey among the experiment participants. For RQ1, external code quality is measured with the help of the number of acceptance test cases passed and static code analysis. For RQ2, internal code quality is measured using McCabe's cyclomatic complexity and branch coverage. For RQ3, the developers' productivity is measured by the number of user stories implemented per person hours and lines of code per person hours(LOC), see Table 1.

3.1 Hypothesis Formulation

The basis for the statistical analysis of an experiment is hypothesis testing [25]. The following hypotheses are stated for investigating research questions RQ1, RQ2 and RQ3:

External Code Quality (Hyp ExtQlty for RQ1). The *null* hypothesis is that the two studied methods give the same external code quality (H_o^1), with the alternative hypotheses that TDD (H_1^1) or TLD (H_2^1) gives better code quality

$$H_o^1 : ExtQlty_{TDD} = ExtQlty_{TLD}$$

$$H_1^1 : ExtQlty_{TDD} > ExtQlty_{TLD}$$

$$H_2^1 : ExtQlty_{TDD} < ExtQlty_{TLD}$$

Internal Code Quality (Hyp IntQlty for RQ2). The *null* hypothesis is that the two studied methods give the same internal code quality (H_o^1), with the alternative hypotheses that TDD (H_1^1) or TLD (H_2^1) gives better internal code quality

$$H_o^2 : IntQlty_{TDD} = IntQlty_{TLD}$$

$$H_1^2 : IntQlty_{TDD} > IntQlty_{TLD}$$

$$H_2^2 : IntQlty_{TDD} < IntQlty_{TLD}$$

Productivity(Hyp Prod for RQ3). The *null* hypothesis is that the two studied methods give the same productivity (H_o^1), with the alternative hypotheses that TDD (H_1^1) or TLD (H_2^1) gives better productivity

$$H_o^3 : Prod_{TDD} = Prod_{TLD}$$

$$H_1^3 : Prod_{TDD} > Prod_{TLD}$$

$$H_2^3 : Prod_{TDD} < Prod_{TLD}$$

Table 1: Measurement criteria

Variable	Chosen measurement
External code quality	No. of accept. test cases passed
	Static code analysis
Internal code quality	McCabe's cyclomatic complexity
	Branch coverage
Productivity	LOC per person hrs.
	No. of user stories implemented per person hrs.
Developers opinion	Survey questionnaires

3.2 Experiment description

The experiment was designed based on the convenience of test subjects from industry since they were geographically and temporally spread apart. Therefore, the subjects were allowed to conduct the experiment based on their convenient time but we strongly recommended in the experiment guidelines to perform it under quiet environment [1]. The experiment design include the following:

- **Subjects :** Professional software developers with atleast 1 year of development experience with TDD and TLD using JUnit
- **Programming Tools :** Used for conducting the experiment is Java, eclipse [26] and JUnit [27].
- **Dependent variables :** External code quality, internal code quality and productivity
- **Independent variables :** TDD and TLD (With separate guidelines for both groups)
- **Objects :** Bowling Game [28]
- **Design :** Random design [25]

At first, a detailed description of TDD and TLD was given to the participants to execute the experiment. The level of details used to explain the experiment was adjusted carefully to avoid developer bias towards a particular testing technique. This prevents the subjects from guessing the hypotheses. The programming tools and the objects for this

[1]The experimental guidelines are available at http://serg.cs.lth.se/index.php?id=88463

experiment were used in the previous studies [17] and therefore their suitability was already proven. Participating subjects were professional software developers with a minimum of one year of enterprise software development experience with TDD and TLD, specifically in the Java programming paradigm and also familiar with bowling game. Before asking the subjects for participation, the test experiment was conducted with master students with the same programming tools and objects to identify the issues (i.e time, appropriate size of user stories, guidelines etc.) and limitations in the experiment design. In addition, pre-experiment questionnaire form was send to all participant in order to know their roles, development experience (TLD and TDD), bowling game familiarity, and writing tests in JUnit.

In order to collect the data regarding RQ4, a survey was conducted on the subjects. Questions included ease of learning curve, effort, first development choice, requirements thoroughness, defects finding and level of maintenance when using both development approaches. The survey questionnaire was designed based on a Likert scale and open-ended format.

3.3 Context and Subject selection

As far as the context characterization of this experiment is concerned, the multi-test within object study technique [25] was considered. In this experiment, random [25] sampling technique was adopted to assign treatments (TDD and TLD) to test subjects. Initially, 31 test subjects were asked to participate in experiment, among them 15 were assigned to the TLD group and remaining 16 were assigned to the TDD group.

The user stories were adopted from Robert Martin's Bowling Score Keeper problem [28]. The same data was used in previous experiments with TDD by George et al. [15] and Erdogmus et al. [13]. However, in this study we organized the problems into 7 small user stories, each describing a unique deliverable and testable feature. Each story was tested by black box acceptance tests that were not disclosed to test subjects.

3.4 Instrumentation and Measurement

Experiment instruments used in this experiment are the user stories, the guidelines for TDD/TLD subjects and the survey questionnaire. The subjects preferred submissions of their experimental results through email. The measurements criteria used for each of the outcome variable can be seen in Table 1.

As for external code quality **(RQ1)**, a test case class was created to identify the defects in the source code produced. This enabled researchers to identify numbers of defects produced by the TDD subjects compared to the TLD subjects. Moreover, static code analysis tool PMD[2] was used to calculate the number of defects in the code.

For internal code quality **(RQ2)**, McCabe's Cyclomatic[3] complexity was used as an objective measure that indicates how difficult a program or module cold be to test and maintain. Moreover, it helps to identify the overly complex parts of the system that are likely to have more defects than non complex parts [29]. Branch coverage[4] was also used to ensure whether or not all the code is executed and no branch leads to the failure of application [30]. Due to a small size of the assignment, the weighted method, lack of cohesion method, nested block depth, information blocks and efferent coupling were discarded.

For productivity **(RQ3)**, LOC per person hours and number of user stories implemented per person hours were considered as measures for productivity of the developers. Given the number of user stories in the experiment, it was decided not to use the Function Points method for size measurement. Finally, the questionnaire was used to answer research question **(RQ4)** with Likert scale and open ended questions.

3.5 Experiment operation

Before running the experiment, an email was send to all selected test subjects to confirm their availability. Next, the experiment guidelines, bowling game user stories and questionnaires were sent to the confirmed subjects. For each method, we prepared separated guidelines for TDD and TLD group to not to throw away our hypothesis. At the same time, the environment used for the development of user stories was the same and included the Eclipse IDE and the JUnit framework for automated testing. Moreover, an excel sheet was created with names and email addresses of all test subjects to keep track of experiment execution and, in case of any queries, to respond back with clarifications.

Due to graphical and temporal separation of our subjects, the bowling game user stories, guidelines and questionnaires were designed in a way to enable the subjects perform the experiment on their personal workstations. In the experiment guidelines, test subjects were instructed to record the time required to complete each user story. After receiving the experimental data from all the test subjects, we verified if the data is reasonable and the assignment is performed according to the guidelines. 13 subjects out of 31 did not respond back with their code submission, 5 of them were rejected because they did not fill out the experiment questionnaire. These 5 subjects were approached again with a request to send back the filled experiment questionnaire, but the answers were not sent back. 13 subjects managed to complete the bowling game assignment and questionnaire. Among them, seven belonged to the TDD group and the remaining six belonged to the TLD group.

3.6 Validity Threats

3.6.1 Internal Validity

Social threats related to internal validity [25] are considered low since neither the subjects nor researchers had any interest in favoring TLD over TDD or vice versa. Control group bias could be another threat to internal validity of the experiment that deals with the bias when the subjects belongs to the group that feels less important and they would have acted differently had they been assigned to a different group. This threat has been dismissed by creating two different experiment guidelines with different pre- and post-experiment questionnaires.

3.6.2 External Validity

In order to eradicate the selection threat to external validity, only professional Java software developers were included in the experiment. The second threat to external validity is

[2]Java Eclipse Plugin for static code analysis, http://pmd.sourceforge.net/eclipse/

[3]Complexity was calculated using Java Eclipse plugin Metrics 1.3.6,http://metrics.sourceforge.net/

[4]Branch coverage was calculated using Java Eclipse plugin EclEmma, http://www.eclemma.org/

related to the selection of right assignment for the experiment. The bowling game used in the experiment may not be the most representative example of an industry project. However, due to the limited availability and time constraints it was not possible to conduct an experiment with an enterprise level application that requires significantly more effort from our subjects. Moreover, the bowling game has been used in the previous experiments related to TDD which makes our results comparable with them [13, 15].

Due to graphical and temporal diversity, it was not possible to conduct the experiment at one place and at one time. Therefore, we designed the experiment in a way that test subjects were able to complete the experiment on their workstations when they have time and sufficient conditions thereby, degree of control can be seen as a validity threat to the study. Apart from experiment guidelines, we also conducted individual Skype sessions with test subjects to answers questions and provide clarifications. The subjects were given 5 weeks to complete the assignment.

The bowling game user stories used in this experiment involve on average only 300 LOC of code, which is far off from the size of typical industrial software applications. Moreover, we allocated only 90 minutes for the experiment to reduce the fatigue of our subjects. In contrast, industrial software projects take much longer time and require more resources. Therefore, identifying the long term practical challenges through this study can be seen as a limitation of this study.

3.6.3 Conclusion Validity

In order to overcome the potential threat to the reliability of the measurement criteria for all outcome variables, objective measures were selected, e.g. the number of acceptance test cases passed or the McCabeś Cyclomatic complexity. Moreover, alternate choices of measurement were also taken into account before making decisions on final measures for outcome variables and all these measures were independent of researcherś subjective estimations, see Table 1.

The low statistical power of test in this study is found to be a validity threat to finding a statistical significant results, see Table 2. For instance, productivity in terms of number of lines of code and number of user stories has probability of (6%) and (5%) respectively to find a statistically significant result, if an effect is there to be detected.

It should be noticed that TLD developers normally do not write automated test cases in JUnit as it is the case in TDD. In our experiment, the subjects wrote the automated tests in JUnit after using TLD approach as well. Consequently, the criticism on writing of test cases after TLD approach can be seen as threat to the validity of this experiment since JUnit is often only associated with TDD. However, given that study had practitioners from industry they are experienced enough to distinguish between TDD and TLD. Thus, test process is more likely to have an outcome in this case than having students as a test subjects who does not fully understand the difference between TDD and TLD.

3.6.4 Construct Validity

In order to mitigate potential construct validity threats [25], the study constructs are defined at an adequate level for each dependent variable. For instance, better internal code quality means one group passed more number of acceptance test cases than the other and also bears less number of bugs in static code analysis. Similarly, better internal code quality means one group attained better branch coverage and low McCabeś Cyclomatic complexity. Furthermore, two different measures for productivity were used in the study. This shows that, for each dependent variable, two different constructs are used to generalize the results of the experiment to the concept or theory behind it.

The second important threat to mention here is the interaction of testing and treatment which means test subjects are already aware of the metrics used for the measurement of their code submission. To nullify this risk it was made sure in the study that experiment guidelines do not ignore any hypothesis or any information that might cause bias behavior from test subjects while implementing the user stories. The homogeneity of subject groups was ensured by including only those professionals who have at least 1 year of development experience. However, each participant has performed the experiment on his/her work station this might have caused the introduction of confounding factors that are effecting the outcome of study without the knowledge of the researchers.

4. RESULTS AND ANALYSIS

4.1 Descriptive statistics

Descriptive statistics were collected to observe data distribution and to identify the possible outliers in the data [25]. We used mean, standard deviation, skewness and Shapiro-Wilk tests to check whether or not the data is normally distributed, see Table 2. Moreover, we used box plots, bar charts for dispersion and skewness of the data set [25], see Table 2 and Figures 2, 3 and 4.

As for **RQ1**, skewness value for number of acceptance test cases passed for TLD (.215) and TDD (2.18) indicated that data is right skewed, and static code analysis in the TLD (-.117) group is showing that data is moderately left skewed compared to TDD (0.69) which is right skewed, see Figure 2.

As for **RQ2**, skewness values of McCabe's Cyclomatic complexity for TLD (-.383) and TDD (-1.23) revealed that data is left skewed. Likewise, branch coverage values of TLD (-.086) and TDD (-.29) group also depicting the same trend, see Figure 3.

As for **RQ3**, skewness values of productivity in terms of LOC for TLD (-.48) indicates data is left skewed compared to TDD (.43) which is right skewed. On the other hand productivity skewness in terms of number of user stories implement per person hour for TLD (1.15) and TDD (.92) group shows that data is right skewed. , see Figure 4.

Dataset Reduction. We used box-plots to search for outliers. The box-plot analysis suggested several outliers that were checked with the labeling rule, see Figure 2. When checked against the labeling rule, by calculating lower quartile [lower quartile - 1.5 * (upper quartile - lower quartile)] and upper quartile [upper quartile + 1.5 * (upper quartile - lower quartile)] [31] [25], the data point for number of acceptance test cases passed is found to be an outlier thus, removed.

4.2 Hypothesis Testing

Hypothesis H_o^1 regarding RQ1. In case the number acceptance test cases passed for TDD and TLD group, significance values of the Shapiro-Wilk test for normality

Table 2: Statistical values

Measurement criteria	Treatment	Mean	Std. Dev	Skewness	Shapiro-Wilk Test (Sig.)	T-test (Sig.)	Mann-Whitney (Sig.)	Effect Size	Sample size required	Achieved Power (1- err prob.)
Number of accept. test cases passed	TLD	5.33	1.50	0.215	.029	N/A	.297	0.80	76	0.27
	TDD	7.28	3.09	2.18	.004					
Static Code analysis	TLD	7.67	4.27	-.117	.55	.036	N/A	0.97	58	0.33
	TDD	3.85	3.48	.69	.475					
McCabe's Cycl. complexity	TLD	8.66	3.26	-0.383	.505	N/A	.097	1.34	30	0.60
	TDD	5.42	.97	-1.23	.000					
Branch coverage	TLD	67.33	29.23	-0.086	.428	.165	N/A	0.89	62	0.32
	TDD	87.45	12.85	-0.29	.172					
Productivity LOC	TLD	1.42	0.76	-0.48	.660	.861	N/A	0.23	960	0.06
	TDD	1.63	1.02	0.43	.140					
Productivity user stories	TLD	0.06	0.03	1.15	.451	.875	N/A	0.11	4298	0.05
	TDD	0.06	0.02	0.92	.612					

[32] (Sig=.029 and Sig=.004 respectively) indicated that the data is not normally distributed. Therefore, Mann-Whitney U test [25] revealed that the difference in the number of acceptance test cases passed is **not** significant (Sig=.297), see also Table 2. However, the T-test is applied for static code analysis (since data was found to be normally distributed), indicated a statistically significant difference (Sig=.036) in the favor of TDD (see Table 2). Looking further at the box plots, see Figure 2, both development groups TLD and TDD have passed relatively similar number of test cases. One important reason could be that in experiment guidelines it was required to write at least one test case for each user story. To summarize we can not reject hypothesis H_o^1 regarding external code quality, but we can confirm with the statistical significance that TDD performs better in case of the static code analysis values.

Hypothesis H_o^2 regarding RQ2. The results for the McCabe's Cyclomatic complexity were not found to be normally distributed thus, Mann-Whitney U test was used [25]. The test revealed lack of statistical significance (Sig=.097). In case of branch coverage, the data was found to be normally distributed for TLD (Sig=.428) and TDD (Sig=.172), but the T-test suggested no statistically significant difference (Sig=.165) between TLD and TDD, see Table 2. The box plots depicted in Figure 3 show some differences but further visual inspection suggests that they may not be significant. Thus, we can not reject hypothesis H_o^2 regarding internal code quality differences.

Hypothesis H_o^3 regarding RQ3. The data regarding the number of LOC and user stories per person hour turned out to be normally distributed (Shapiro-Wilk test). Therefore, T-test was applied and result was not significant for LOC (Sig=.861) and for the number of user stories implemented per person hour (Sig=.875). The box plots depicted in Figure 4 show some differences but visual inspection suggests that they may not be significant. It is to be noticed that the box-plot in Figure 4.2 suggests an outlier but the labeling rule results [31] disproves this. Thus, we can not reject hypothesis H_o^3 regarding productivity.

Research question RQ4. The results regarding the subjectś preferences (see Table 3) clearly indicate that TLD is easy to use (100% answers) and easier than TDD (86% answers). Subjects involved with the TDD approach presented a rather mixture where 86 % of results indicated a positive level of ease while the remaining 14 % demonstrated difficulty in understanding and using the TDD approach. Furthermore, as many as 86% of our subjects indicated neutral impact of TDD on effort while 66% indicated negative impact of TLD on the effort required. The survey results suggest that TDD is easier to learn (70% suggested positive learning curve).

Table 3: Survey findings.

Aspect	TLD			TDD		
	Positive	Neutral	Negative	Positive	Neutral	Negative
Ease of use	100%	0%	0%	86%	0%	14%
Effort	17%	17%	66%	14%	86%	0%
Learning curve	33%	33%	33%	72%	14%	14%
First choice of the development method	66%	17%	17%	28%	72%	0%
Requirements thoroughness	83%	17%	0%	72%	14%	14%
Defects	85%	0%	15%	44%	28%	28%
Level of maintenance	66%	17%	17%	72%	28%	0%

More subjects (66%) would rather select TLD than TDD (38%) as the first choice development method but the results need to be interpreted with discretion since 72% of the TDD subjects selected the neutral answer for this question. The descriptive statistics of collected data indicates that the TLD subjects unanimously agree that their development style allowed them to thoroughly assess all program requirements and cover all use cases while the TDD subjects came short in comparison.

Both TLD and TDD seems to relive requirements thoroughness since 83% of TLD and 72% of TDD respondents selected the positive option for this question. The results presented in Table 3 also indicate that TLD is more effi-

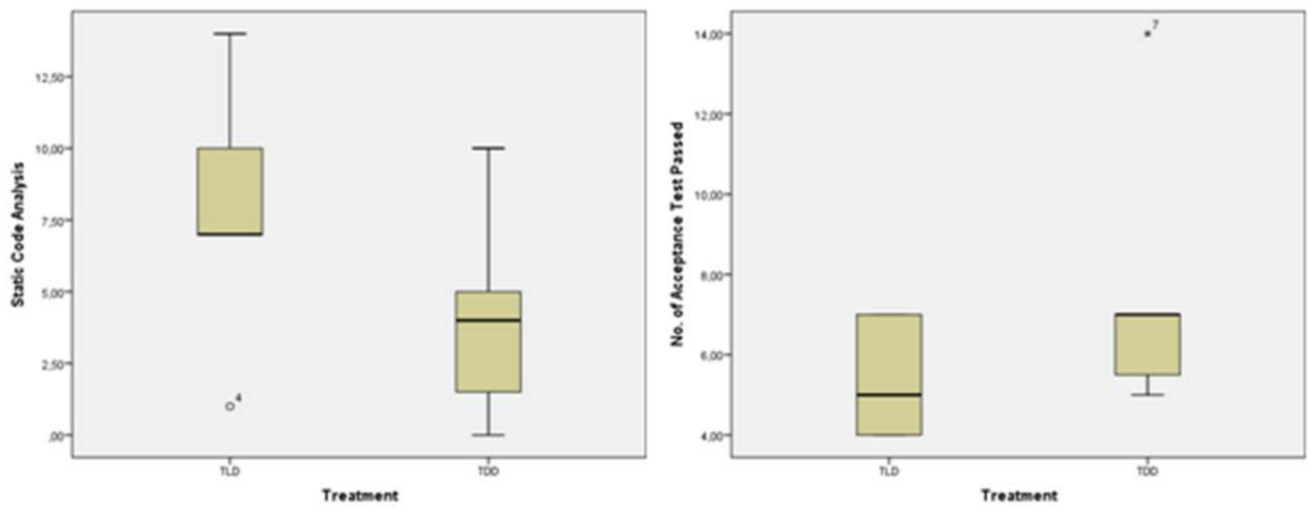

Figure 2: Box plot for external code quality.

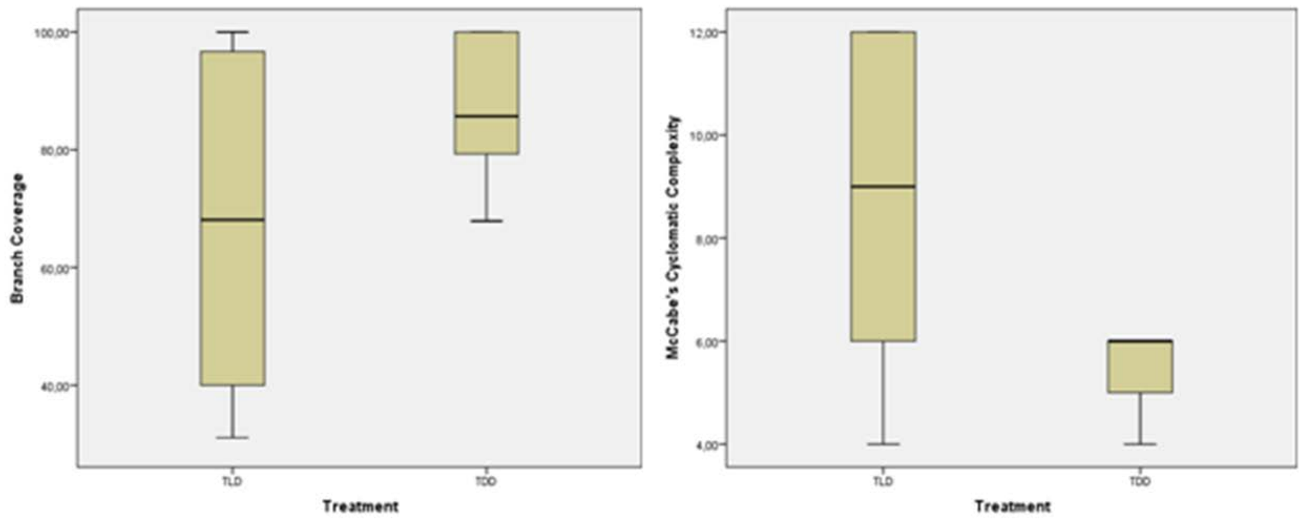

Figure 3: Box plot for internal code quality

cient (85%) and thorough when it comes to bug discovery and resolution of program defects while the TDD results are mixed across categories with a 44 % positive rating answers. Finally over 65% of our respondents in both methods indicated a positive impact of the methods on the level of maintenance.

4.3 Power Estimation

Statistical power estimation of an experiment is the probability that a test will correctly reject the null hypothesis [33]. Statistical power is associated with: 1) **Statistical significance** - α (in this study set to 0.05) is probability of rejecting null hypothesis incorrectly, Type I error. 2) **Sample size N** - large sample size increases the precision and statistical power of the test. 3) **Effect size** - the real difference between the null and alternative hypotheses.

As can be seen in Table 2, the effect size calculated in this study from mean and standard deviations of two independent samples is found to be large for number of acceptance test cases (0.80), static code analysis (0.97), McCabe's Cyclomatic complexity (1.34) and branch coverage (0.89). Conversely, the effect size of productivity in terms of number of lines of code (0.23) and productivity in terms of number of user stories (0.11) is found to be small.

The statistical power achieved in this experiment (1- err prob.) for the number of acceptance test cases passed and static code analysis is 0.27 and 0.33 respectively. It indicates that that the study has a 27 % and 33 % probability for number of acceptance test cases and static code analysis respectively for finding a statistically significant result (if an effect is there to be detected) and to reject the null hypothesis associated with external code quality. Similarly, the achieved power for number of McCabe's Cyclomatic complexity and branch coverage is 0.60 (60%) and 0.32 (32%) respectively. On the other hand, the achieved power for productivity in terms of number of lines of code and number of

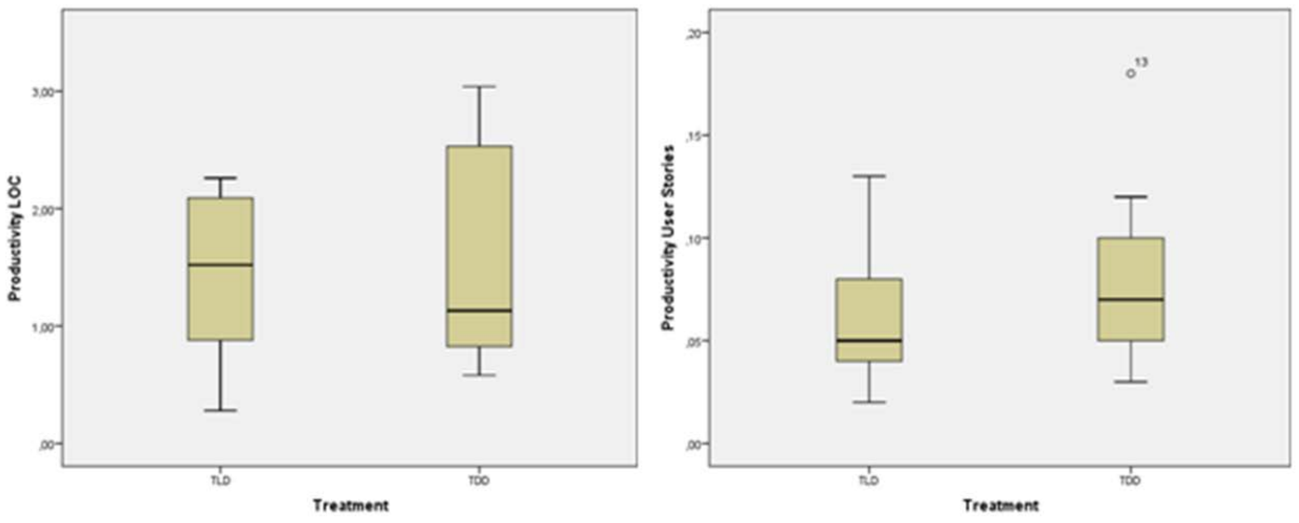

Figure 4: Box plot for productivity

user stories is 0.06 (6%) and 0.05 (5%) respectively.

5. DISCUSSION AND CONCLUSIONS

The statistical analysis does not allow us to reject any of the hypotheses stated for research questions RQ1, RQ2 and RQ3. The results of the experiment showed no statistically significant difference in favor of TDD for dependent variables namely the number of acceptance test cases passed (associated with RQ1), McCabe's Cyclomatic complexity and branch coverage (associated with RQ2) and the number of user stories implemented per person hours and line of code per person hours (associated with RQ3).

However, the difference in the number of defects found using static code analysis result was found to be statistically significant in the favor of TDD (see Table 2), but due to low statistical power and lack of significance for testing the number of acceptance tests passed we were unable to reject the null hypothesis H_o^1 associated with the external code quality (Hyp ExtQlty). The results suggest that TDD has the potential to reduce the number of defects using static code analysis however, more data is needed to gain further confidence. The box plots regarding the number of acceptance tests passed, depicted in Figure 2, indicate some differences, however not significant.

Similarly, the box plots regarding the internal code quality, see Figure 3, indicate that TDD delivers slightly better branch coverage and lower complexity than TLD. Again statistical hypothesis testing was unable to confirm this hypotheses and thus we were unable to reject the null hypothesis H_o^2 associated with the internal code quality (Hyp IntQlty).

The differences in the productivity aspects of the studied methods seems to be smaller than the internal and external code quality, see Figure 4. Figure 4 suggest that subjects using TDD achieved slightly lower productivity median in terms of LOC but slightly higher median in terms of the produced user stories. Again statistical hypothesis testing was unable to confirm this hypotheses and thus we were unable to reject the null hypothesis H_o^3 associated with productivity (Hyp Prod).

One important reason for the study not being able to reject the null hypotheses is low statistical power of the test associated with only 13 data points divided into two groups. Due to the same reason, the survey results were not statistically analyzed. However, the results of this study are found consistent with the previously performed experiments [13] [34] [14] [5] [35] [36] [37] [38] [39] [11] [19] [40] [41] revealing little or no significant difference in terms of internal code quality, external code quality and productivity.

Furthermore, the survey results, see Table 3, indicate that the majority of developers favor TLD over TDD, especially when it comes to attributes and factors such as the higher learning curve, extra effort required to maintain and understand the code all of which consequently negatively affect the adoption of TDD as the first development method of choice. The subjects were queried to provide feedback based on their coding experience. The feedback revealed that the majority of the TDD subjects was mostly unfamiliar with TDD and the assignment was their first experience working with TDD. Some participants also indicated that TDD requires a strict discipline to actually write the test first, think about the underlying interfaces and components to be able to adequately come up with a legitimate test case, all of which contributed to the extra learning curve. Some also noted that writing test cases added little up-front value seemingly because writing unit tests took the developer away from the actual implementation and completing functional use cases which from the developer's and client's point of view had the most value.

Another issue that possibly contributed to participants aversion to TDD adoption is noted by the fact that most developers struggled with the design of a proper test case. One TDD developer noted:
"*Making a strategy on how to start the test case, and the calculation of the random test values is very difficult. It is hard to keep the two streams that are test and code decoupled*".
Similarly, another notes "*TDD is rather a design technique, not a testing technique. The created tests and the green bar may stimulate a false sense of security, i.e. that the application is well tested while in fact, that only depends on the*

quality of the developed tests". This only goes to further indicate that TDD adoption requires not only adherence to guidelines from all aspects of the software development but also adequate and sufficient training in improving one's skill set in testing. In brief, it must be understood that TDD requires sufficient testing skills and decent adherence to guidelines, which at times require a certain level of commitment from all involved parties in software development. TDD is not to replace software design, modeling and proper architecture but is only a tool that addresses certain aspects of software development in order to help drive the system requirements. The trade-offs pointed out in this research must carefully be studied and understood within an appropriate context so as to qualify TDD as the proper development technique for any software development project.

Future work recommendations. Based on the experiment conducted with 13 professional Java developers, there are number of things that should be considered and used as guidelines for future experiments to check positive or negative impacts of TDD over TLD. It was found that majority of the studies investigated the dependent variables such as productivity, external code quality and internal code quality since it has more industry relevance. Therefore, the selection of variables is of prime importance and productivity, internal code quality and external code quality should be considered as dependent variables for future experiments. Second, a right set of metrics in order to measure the above mentioned 3 dependent variables is also very important to ensure their objectivity. The measures used for productivity in this study are number of user stories implemented per person hours and lines of code per person hours. Similarly, McCabe's Cyclomatic complexity and branch coverage are used to measure the internal code quality. Likewise, number of acceptance test cases passed and number defects found through statics code analysis are used to measure the external code quality. All these measures are consistent with the previous studies and should be considered as standard measures for future experiments.

As far as the selection of tools for measuring these dependent variables is concerned, PMD[2] could be considered as a good static code analysis tool for small examples used in the experiments. Similarly, Java metrics plug-in[3] is also well capable of producing meaningful results for McCabe's cyclomatic complexity and lines of code as it was the case in this study. Finally, Java code coverage plug-in[4] could also be used in future experiments to measure the code coverage. Moreover, it is desirable to conduct the experiment with professional Java developers instead of students with at-least 50 participants to ensure the industry relevance.

Finally, the majority of the previous experiments did not report their experimental values such as mean, median, standard deviation, confidence intervals and P-values which makes the comparison and meta-analysis of two development methods (TDD and TLD) significantly difficult. Moreover, future studies should consider the rigor and relevance criteria in their designs because it enables researchers to gain much more conclusive results, whether or not TDD gives a significant difference in terms of internal code quality, external code quality and productivity.

Acknowledgement

This work is partly founded by the SYNERGIES project, Swedish National Science Foundation, grant 621-2012-5354.

6. REFERENCES

[1] C. Larman and V.R. Basili. Iterative and incremental developments. a brief history. *Computer*, 36(6):47–56, June 2003.

[2] Kent Beck and Cynthia Andres. *Extreme Programming Explained: Embrace Change*. Addison-Wesley Professional, 2 edition, November 2004.

[3] M. Pancur, M. Ciglaric, M. Trampus, and T. Vidmar. *Towards empirical evaluation of test-driven development in a university environment*, volume 2. IEEE, 2003.

[4] H. Cibulski and A. Yehudai. Regression test selection techniques for test-driven development. In *2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 115–124. IEEE, March 2011.

[5] Liang Huang and Mike Holcombe. Empirical investigation towards the effectiveness of test first programming. *Information and Software Technology*, 51(1):182–194, 2009.

[6] S. Ambler. Quality in an agile world. *Software Quality Professional*, 7(4):34, 2005.

[7] Sumanth Yenduri and Louise A Perkins. Impact of using test-driven development: A case study. In *Software Engineering Research and Practice*, pages 126–129, 2006.

[8] Thirumalesh Bhat and Nachiappan Nagappan. Evaluating the efficacy of test-driven development: Industrial case studies. In *ISCE'06 - 5th ACM-IEEE International Symposium on Empirical Software Engineering*, volume 2006 of *ISESE'06 - Proceedings of the 5th ACM-IEEE International Symposium on Empirical Software Engineering*, pages 356–363, Rio de Janeiro, Brazil, 2006. ACM.

[9] M.M. Muller and O. Hagner. Experiment about test-first programming. *IEE Proceedings-Software*, 149(5):131–6, October 2002.

[10] Atul Gupta and Pankaj Jalote. An experimental evaluation of the effectiveness and efficiency of the test driven development. In *1st International Symposium on Empirical Software Engineering and Measurement, ESEM*, Proceedings - 1st International Symposium on Empirical Software Engineering and Measurement, ESEM 2007, pages 285–294, Madrid, Spain, 2007. Inst. of Elec. and Elec. Eng. Computer Society.

[11] Matjaz Pancur and Mojca Ciglaric. Impact of test-driven development on productivity, code and tests: A controlled experiment. *Information and Software Technology*, 53(6):557–573, 2011.

[12] Sami Kollanus. Test-driven development - still a promising approach? In *7th International Conference on the Quality of Information and Communications Technology, QUATIC*, Proceedings - 7th International Conference on the Quality of Information and Communications Technology, QUATIC 2010, pages 403–408, Porto, Portugal, 2010. IEEE Computer Society.

[13] Hakan Erdogmus, Maurizio Morisio, and Marco Torchiano. On the effectiveness of the test-first approach to programming. *IEEE Transactions on Software Engineering*, 31(3):226–237, 2005.

[14] A. Geras, M. Smith, and J. Miller. A prototype

empirical evaluation of test driven development. In *Proceedings - 10th International Symposium on Software Metrics, METRICS 2004, September 14, 2004 - September 16, 2004*, Proceedings - International Software Metrics Symposium, pages 405–416, Chicago, IL, United states, 2004. IEEE Computer Society.

[15] Boby George and Laurie Williams. A structured experiment of test-driven development. volume 46 of *Information and Software Technology*, pages 337–342. Elsevier, 2004.

[16] E.M. Maximilien and L. Williams. Assessing test-driven development at IBM. In *IEEE 25th International Conference on Software Engineering, 3-10 May 2003*, pages 564–9, Los Alamitos, CA, USA, 2003. IEEE Comput. Soc.

[17] Hussan Munir, Misagh Moayyed, and Kai Petersen. Considering rigor and relevance when evaluating test driven development: A systematic review. *Information and Software Technology*, (0):–, 2014.

[18] Forrest Shull, Grigori Melnik, Burak Turhan, Lucas Layman, Madeline Diep, and Hakan Erdogmus. What do we know about test-driven development? *IEEE Software*, 27(6):16–19, 2010.

[19] M. Pancur, M. Ciglaric, M. Trampus, and T. Vidmar. Towards empirical evaluation of test-driven development in a university environment. In *Proc. IEEE Region 8 EUROCON 2003 - Computer as a Tool*, volume vol.2 of *IEEE Region 8 EUROCON 2003. Computer as a Tool. Proceedings (Cat. No.03EX655)*, pages 83–6, Piscataway, NJ, USA, 2003. IEEE.

[20] Boby George and Laurie Williams. An initial investigation of test driven development in industry. In *Proceedings of the 2003 ACM Symposium on Applied Computing, March 9, 2003 - March 12, 2003*, Proceedings of the ACM Symposium on Applied Computing, pages 1135–1139, Melbourne, FL, United states, 2003. Association for Computing Machinery.

[21] Gerardo Canfora, Aniello Cimitile, Felix Garcia, Mario Piattini, and Corrado Aaron Visaggio. Evaluating advantages of test driven development: A controlled experiment with professionals. In *ISCE'06 - 5th ACM-IEEE International Symposium on Empirical Software Engineering, September 21, 2006 - September 22, 2006*, volume 2006, pages 364–371, Rio de Janeiro, Brazil, 2006. Association for Computing Machinery.

[22] Lars-Ola Damm and Lars Lundberg. Results from introducing component-level test automation and test-driven development. *Journal of Systems and Software*, 79(7):1001–1014, 2006.

[23] Nachiappan Nagappan, E. Michael Maximilien, Thirumalesh Bhat, and Laurie Williams. Realizing quality improvement through test driven development: Results and experiences of four industrial teams. *Empirical Software Engineering*, 13(3):289–302, 2008.

[24] L. Williams, E.M. Maximilien, and M. Vouk. Test-driven development as a defect-reduction practice. In *14th International Symposium on Software Reliability Engineering, 17-20 Nov. 2003*, 14th International Symposium on Software Reliability Engineering, pages 34–45, Los Alamitos, CA, USA, 2003. IEEE Comput. Soc.

[25] Claes Wohlin, Per Runeson, and Martin Host. *Experimentation in Software Engineering: An Introduction*. Springer, 1st edition, December 1999.

[26] Eclipse IDE for java developers | eclipse packages.

[27] Downloads for KentBeck's junit - GitHub.

[28] Robert C. Martin. *Agile Software Development, Principles, Patterns, and Practices*. Prentice Hall, 1st edition, October 2002.

[29] Norman E. Fenton and Shari Lawrence Pfleeger. *Software Metrics: A Rigorous and Practical Approach, Revised*. Course Technology, 2 edition, February 1998.

[30] Brian Henderson-Sellers. *Object-Oriented Metrics: Measures of Complexity*. Prentice Hall, 1 edition, December 1995.

[31] John Wilder Tukey. *Exploratory Data Analysis*. Addison-Wesley Publishing Company, 1977.

[32] Sam Kash Kachigan. *Statistical Analysis: An Interdisciplinary Introduction to Univariate & Multivariate Methods*. Radius Press, January 1986.

[33] T. Dyba, V.B. Kampenes, and D.I.K. Sjoberg. A systematic review of statistical power in software engineering experiments. *Information and Software Technology*, 48(8):745–55, 2006.

[34] C. Desai, D. Janzen, and K. Savage. A survey of evidence for test-driven development in academia. *SIGCSE Bulletin*, 40(2):97–101, June 2008.

[35] David S. Janzen and Hossein Saiedian. A leveled examination of test-driven development acceptance. In *29th International Conference on Software Engineering, ICSE 2007, May 20, 2007 - May 26, 2007*, Proceedings - International Conference on Software Engineering, pages 719–722, Minneapolis, MN, United states, 2007. Inst. of Elec. and Elec. Eng. Computer Society.

[36] D.S. Janzen and H. Saiedian. Does test-driven development really improve software design quality? *Software, IEEE*, 25(2):77–84, 2008.

[37] David S. Janzen and Hossein Saiedian. On the influence of test-driven development on software design. volume 2006, pages 141 – 148, Turtle Bay, HI, United states, 2006.

[38] Lech Madeyski. 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, 2010.

[39] L. Madeyski and L. Szala. The impact of test-driven development on software development productivity an empirical study. *Software Process Improvement*, pages 200–211, 2007.

[40] John Huan Vu, Niklas Frojd, Clay Shenkel-Therolf, and David S. Janzen. In *6th International Conference on Information Technology: New Generations, ITNG*, pages 229–234, Las Vegas, NV, United states, 2009. IEEE Computer Society.

[41] Thomas Flohr and Thorsten Schneider. Lessons learned from an XP experiment with students: Test-first needs more teachings. In *Product-Focused Software Process Improvement*, volume 4034, pages 305–318. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.