Assessment of Test-driven Development in an industrial context

DANIEL GUSTAVSSON
Assessment of Test-driven Development in an industrial context

Utvärdering av testdriven utveckling i industriell miljö

Daniel Gustavsson – danielgu@kth.se
Degree project subject: Computer science
Programme: Master of science in Media Technology
Supervisor: Filip Kis – Examinator: Roberto Bresin
Principal: Decerno AB
2017-07-02

Abstract
Test-driven development (TDD) has been the target of many articles in which the authors try to reveal the impact of TDD compared to the traditional iterative test-last approach (ITL). Most of the existing articles conduct case studies in academic setting or in industrial setting with focus on metrics of little relevance for the industry. The metrics Defect Density per 1000 lines of code (DD/KLOC) and McCabe’s cyclomatic complexity number are frequently used to show differences between ITL and TDD in quality. However, these metrics are outdated and irrelevant in today’s industry according to several articles regarding TDD. The reason is the introduction of high-level programming languages such as Java, C# and Python. To compare these languages with assembler languages is not feasible. In this master thesis, the author suggests measuring defect density per hour instead (DD/h) of per 1000 lines of code to establish the quality and success of a software project. DD/h together with total hours needed to develop the software is a better measurement and is not dependent on programming language and complexity as DD/KLOC is to describe quality and the scope of the development project. By examine DD/h from five case studies comparing ITL and TDD the author shows a possible positive change in quality and resources needed in software development projects.

Sammanfattning
Testdriven utveckling (TDD) har varit i fokus många artiklar där författarna försöker kartlägga TDD: s inverkan jämfört med det traditionella tillvägagångssättet där tester körs sist i utvecklingsprocessen (ITL). De flesta av de befintliga artiklarna genomför fallstudier i akademisk miljö eller i industriella miljöer med fokus på måtvärden som saknar relevans för industrin. Defektdensitet per 1000 linjer kod (DD/KLOC) och McCabes cyklomatiska komplexitetsnummer används ofta för att visa skillnader mellan ITL och TDD i kvalitet. Dessa måtvärden är emellertid föråldrade och irrelevanta i dagens bransch enligt flera akademiska artiklar. Anledningen är introduktionen av programmeringsspråk så som Java, C# och Python. Att jämföra dessa språk med assembler-språk är inte möjligt. I detta examensarbete föreslår författaren att man mäter defektkanslänk pot timme (DD/h) istället för DD/KLOC för att fastställa kvaliteten för ett mjukvaruprojekt. DD/h tillsammans med de totala timmar som behövs för att utveckla programvaran är ett bättre måttvärde och är inte beroende av programmeringsspråk och komplexitet som DD/KLOC är för att beskriva kvaliteten och omfattningen av utvecklingsprojektet. Genom att undersöka DD/h från fem fallstudier som jämför ITL och TDD visar författaren en möjlig positiv förändring av kvalitet och resurser som behövs för att genomföra ett mjukvaruutvecklingsprojekt.
Assessment of Test-driven Development in an industrial context

Daniel Gustavsson
KTH Royal Institute of Technology
Lindstedtsvägen 3
114 28, STOCKHOLM
+46707797291
danielgu@kth.se

ABSTRACT
Test-driven development (TDD) has been the target of many articles in which the authors try to reveal the impact of TDD compared to the traditional iterative test-last approach (ITL). Most of the existing articles conducts case studies in academic setting or in industrial setting with focus on metrics of little relevance for the industry. The metrics Defect Density per 1000 lines of code (DD/KLOC) and McCabe’s cyclomatic complexity number are frequently used to show differences between ITL and TDD in quality. However, these metrics are outdated and irrelevant in today’s industry according to several articles regarding TDD. The reason is the introduction of high-level programming languages such as Java, C# and Python. To compare these languages with assembler languages is not feasible. In this master thesis, the author suggests measuring defect density per hour instead (DD/h) of per 1000 lines of code to establish the quality and success of a software project. DD/h together with total hours needed to develop the software is a better measurement and is not dependent on programming language and complexity as DD/KLOC is to describe quality and the scope of the development project. By examine DD/h from five case studies comparing ITL and TDD the author shows a possible positive change in quality and resources needed in software development projects.

1 INTRODUCTION
In this master thesis, I explore Test-Driven Development (TDD) and evaluate the value of TDD in a system development SME. The master thesis has been conducted in close cooperation with Decerno (referred to as the company in the text), a system development company located in Stockholm, Sweden. They plan, build and maintain large digital systems for companies and organizations.

For any professional software development project, testing is an important part of the development process. If defects are detected and fixed early in the development process the company saves both time and money in the long run. According to Boehm a defect can cost up to 100 times more if it is detected after the system is delivered [8]. In traditional development methods such as the Waterfall model, the code is tested after the implementation is completed [30, 38]. This approach makes it difficult to react to requirements changes due to shifted customer needs and shifting external environment, which is to consider normal circumstances when developing large software systems of today [25, 30, 56]. With the Waterfall approach, the code will need rework after the implementation is completed. Up to 40-50% of the total time spent in software projects is avoidable by catching defects early on in the project [15].

TDD is a practice within Extreme Programming (XP) that calls for the developers to write tests before any functional code is written [4, 5]. Only when the first unit test is passed do the developers continue with refactoring the code as needed and then write a new test for the next functionality to be implemented in the system. With this iterative process the developers lay the foundation for a structured code and raise the confidence within the project that the code is of high quality and easy to implement new functionality to if needed [5]. XP is an agile development methodology and one of the core elements in agile development is to only document the important parts of a project [3], [8], [9]. TDD is one of the main contributors towards this approach [10], [2] by serve as the requirement specification for the system [1, 5, 25, 42]. Furthermore, TDD is an important variable when making decisions in the design and planning of the system.

1.1 Background
Decerno use the agile method Scrum, which is a responsive development process in contrast to the waterfall model [47][23], but the testing of the functional code is normally conducted at the end of the project or at the end of each sprint. The company is aware that defects which are detected after release are generally more complex to handle and takes more time and effort to address. Which in turn drains resources from both the company and the customer. By examine how TDD affects the density of defects per hour rather than per 1000 lines of code, the company hopes to get a more understandable metric to present for their customers and a metric that is compatible and useful in industrial context.

1.2 Previous work
If TDD is preferred over the conventional iterative test-last (ITL) development process have been examined before in the literature. Various metrics have been used to establish the difference between these two development practices, the most commonly used metrics in the literature to determine code quality and size of the project is defect density per 1000 lines of code and McCabe’s cyclomatic complexity metric [37]. Articles that examine
the impact of TDD exclusively uses these two metrics to describe a software development project’s quality, scope and size. To display the differences between ITL and TDD the case studies in the articles focus on internal or external quality. Internal quality focus on the code design and external quality on the functionality and behavior of the system (the code design). The purpose of measuring internal quality is to ensure that the external quality is achieved [6].

In an article from 2010 [49] Forrest et. al suggested that TDD is better than ITL in delivered quality, internal code quality and in productivity. Their conclusion is based on a literature study with 22 published articles and 33 unique case studies. 13 of the case studies were performed in a professional setting with graduates or professional programmers with several years of experience. If case studies in academic settings are to be excluded the result is the same except for the internal code quality which in this case did not favor either ITL or TDD. In the latter case the difference between ITL and TDD is not as clear as when studies in academic settings are included in the comparison. This observation and conclusion is supported in the article by Sinaialto and Abrahamsion [52] “Does test-driven development improve the program code?”.

The results from the five case studies presented in the article suggests that the change in internal quality is not enough to differentiate between ITL and TDD and they criticize the existing empirical research to mainly focus on external quality instead of internal quality [52]. A suggested reason to why external quality usually is the focus in case studies is that external quality is of more interest in an industrial context [13] because it is easier to map external quality to used resources and because external quality is often target of quality check up from the customer part by acceptance testing.

In another article from 2013 by Rafique and Misic [43] it is suggested that practicing TDD instead of ITL results in a small improvement in both quality and productivity when both academic and industrial case studies are included in the result. Rafique and Misic also divided the 27 studies into two subgroups of academic and industrial case studies and conclude that the change in quality and productivity are considerably higher when they examined only the industrial subgroup. They observed in the subgroup that the quality of the code was raised and the productivity was lowered. The authors argue that the change in quality is due to more experienced programmers in conjunction with larger projects in regards of lines of code, time and complexity of the projects. They use the same attributes to explain the change in productivity, specifically the more experienced programmers in the industrial setting because they are more capable to write unit tests and because of this they can practice TDD more extensively than a student. Rafique’s and Misic’s conclusion is in line with the article written by Forrest et al. but expose a clearer distinction between case studies performed in an academic setting versus studies performed in an industrial setting.

Previous articles, including the three mentioned in this section frequently uses the following metrics to differentiate between ITL and TDD: Defect density per 1000 lines of code (DD/KLOC), cyclomatic complexity and the number of external acceptance tests passed performed by the customer. While these metrics are easy to extract today some argues that they are a bad fit for industrial context. Fenton and Neil [13] presents several reasons to why the mentioned metrics are a bad fit for the industrial context. The metric DD/KLOC dates to the beginning of 1960 and is a bad fit because of today’s high level programming languages. A notable difference in lines of code needed to complete the same application in different high level programming language exists and because of this DD/KLOC is not a good metric to use when measuring software quality or project success [13]. They also discuss the relevance that existing case studies have in an industrial context and presents two main concerns. The first is that most case studies lacks in scope, that they mainly focus on metrics that is only replicable for smaller programs and that the content in the studies focus on detailed code metrics when the industry asks for metrics that are relevant for development process improvements. They also questioning metrics relating to defect counts and McCabe’s cyclomatic number [13], much because of the many high level programming language available today and the difficulty to map the results from different case studies examine different programming languages. Fenton and Neil is not alone to questioning the metrics used in existing case studies and there is even voices that argues that DD is not the most important to measure but rather mean time to fix defects (MTTF) [50]. MTTF is, according to Grigori [50] more interesting in the long run and better shows the advantage with TDD over ITL. Despite this the DD/KLOC and the cyclomatic number continues to be popular choices in case studies.

The difficulties to introduce TDD in a company’s development process is something that Causevic et al. address in their article from 2011. They presents 7 major difficulties when companies introduce TDD in the development process as seen in Table 1: The table is the same as presented in the article by Causevic et al. [9].

Table 1. Factors limiting industrial adoption of test driven development: A systematic review.

<table>
<thead>
<tr>
<th>#</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Development time</td>
</tr>
<tr>
<td>2</td>
<td>Experience/knowledge</td>
</tr>
<tr>
<td>3</td>
<td>Design</td>
</tr>
<tr>
<td>4</td>
<td>Skill in testing</td>
</tr>
<tr>
<td>5</td>
<td>TDD adherence</td>
</tr>
<tr>
<td>6</td>
<td>Domain and tool specific issues</td>
</tr>
<tr>
<td>7</td>
<td>Legacy code</td>
</tr>
</tbody>
</table>

Of these seven identified difficulties development time, experience and skill in testing are of interest for this master thesis. When introducing TDD into a company an obvious up-front loss could make a company to reconsider if TDD is to be introduced in their business or not, despite that the long-term gain could be larger than the up-front loss. The programmers practical experience in TDD differs even in a professional setting and Causevic et al. [9] observed that in several studies the participants got a
minor introduction to the TDD approach. The authors argue that this implies that knowledge about TDD is low in industrial settings as well, and not only something that is a problem in academic settings. The last of the three difficulties that will be of interest for this master thesis is general skill in testing. This include the programmer’s ability to write automated test cases with or without TDD in mind. The authors states that there is no explicit existing work that investigate the quality of test cases produced by programmers when TDD is practiced.

1.3 Thesis objective
The aim with this Master Thesis is to explore if TDD adds value to software development projects in the aspect of time and money by decreasing the defect density per hour of development. By answering the sub-research question:

“How to assess relevant quality changes in software development projects in an industrial context?”

The author of this thesis will seek to answer the main research question:

“Does TDD contribute relevant positive changes to the business of software development SME?”

1.4 Limitations
Due to the nature of the master thesis, including the time span, no test cases with developers were performed for this report. The projects at the company are normally several years long which make it impossible to follow a project from start to end and use that to evaluate TDD. The data extracted in this master thesis is only extracted from two projects at the company because of limited access for non-employees at the company. Instead a literature research was conducted on test cases that focus on TDD.

2 THEORY

2.1 Traditional software development
The waterfall model is a strictly linear process where you complete each phase in sequence, from top to bottom [15], as seen in Figur 1. Det går inte att hitta någon referenskälla. The functional code is tested first after the implementation is complete in the step prior to deployment and maintenance of the system. This model has come to be modified over the years to better meet the demands from modern software development and one of these alteration is the V-shaped waterfall model, which include testing in all phases of the development process. The tests in the V-model is still executed at the end of each phase [2, 38]. The strength of traditional development practices such as the Waterfall model is the ability to plan. With the Waterfall model a project can be detailed planned from start to end, the downside is when external circumstances change the conditions for the project. Which is normal in software development today.

![Figure 1. The five phases in the waterfall model: Requirements analysis, design, coding, testing and maintenance.](image)

2.2 Agile development
In modern software development, an agile approach is normally used to develop software systems. This keeps the development project responsive to the changing variables that can affect the project and the system versatile to implementation of new functionalities as needed [21, 54]. It is important to understand that Agile development is a family name that includes different practices and methods, which help the development process to be Agile. In the Manifesto for Agile Software Development [3] it is stated:

- Individuals and Interaction over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over negotiation
- Responding to challenge over contract negotiation

The authors of the agile manifesto, including Beck, stress that not all agile methods are a good fit for every project and that one should pick and choose the elements that makes most sense for a specific project. Two of the agile methods are Scrum and Extreme Programming.

2.2.1 Scrum
Scrum is a method that has been used to manage complex projects since the early 1990s. For software projects where it is difficult to plan the entire project at the start of the project, Scrum is a good method that uses the feedback from the customer to constantly improve the software throughout the project. The Scrum framework consists of Scrum Teams and their associated roles, events, and rules. One member per team is also appointed the Scrum master, which is in charge to identify and solve problems that hinder the teams productivity. [46–48]. The software is developed iterative in increments, called sprints and the Scrum process is as follow [48]:

- Planning
- Put features to implement in a backlog
- Product owner prioritize items in backlog
- Sprint is started
- Daily stand-up meetings within the teams to plan the daily work
- Sprint ends
- Review of the sprint to the customer
Despite Scrum’s popularity in industrial settings the method has not been the target of much research. In an empirical study by Dybå and Dingsoyr from 2008 it is revealed that a large portion of the existing case studies regarding agile methods focus on XP. Dybå and Dingsoyr examined 33 articles regarding agile methods and only one of them focused on Scrum while 25 of the articles focused on XP. This is remarkable because Scrum seems to be so popular in the industry, yet no evidence exists in the literature that supports that Scrum is superior to other methodologies such as XP [11]. In the article the authors conclude, from the study that address Scrum, that the positive aspect of Scrum is that it brings the developers closer to the customer and give them confidence that they develop the product the customer wants. All of the developers favored Scrum for future projects and overtime was reported to reduce after introducing Scrum in this particular study [11].

2.2.2 Extreme programming
Focus in extreme programming is on best practice for development. The values: communication, simplicity, feedback and courage together with the four basic activities: coding, testing, listening and debugging forms the method know as Extreme programming [4]. These values and activities leads to the 12 core practices in XP: the planning game, metaphor, small releases, simple design, testing, refactoring, pair programming, collective ownership, continuous integration, 40 hours week, on-site customers, and coding standards [4]. All these practices are taken to its extreme when practiced with the XP method. Especially testing and small releases.

Unlike Scrum, XP has been extensively examined and researched in the literature. In the same article that is mentioned in 2.2.1 76% of the 33 case studies focused on XP. Although XP has support from the literature that it adds value to software development projects only 95% of the participants in the 25 studies said that they would recommend XP instead of traditional development methods. In contrast to Scrum, XP raise the sense of productivity amongst the developers. Pair programming in particular did contribute to this according to the participants in the 25 studies examined by Dybå and Dingsoyr [11].

2.2.3 Test-Driven development
Extreme programming consists of 12 core practices and two of them is testing and refactoring. The testing in XP consist of test-first testing, meaning that you write the unit test before you write any functional code [4]. Test-first with automated test suits, together with refactoring of the code is normally referred to as TDD. TDD is a software development practice that relies on small increments and many iterations of the code. first the developer writes a test case that defines a new function or a new improvement in the existing code, then they write the minimum amount of code to pass the written test case, and finally refactors the new code to acceptable standards defined by the company. TDD involves tests but is also highly present in the planning and design phases of a software development project [1, 25, 34].

![Figure 2. The life cycle of TDD: fail, pass, refactor.](image)

The workflow of TDD can be seen in Figure 2. By developing with TDD a clean code is produced which is easy to implement new functionality to by the instant feedback from the automated test suits. You also ensure that the requirement specification from the customer is in focus when planning for the implementation because the tests in TDD specify the system before any functional code is written [1, 34]. By this the tests aims to satisfy the requirements of the customer and not the implementation that is the possibility when developing with ITL.

2.3 Defect density
One of the most common way to measure code quality is to calculate defects per 1000 lines of code which is referred to as DD/KLOC [14] and is calculated as:

\[
\frac{\text{Defects}}{\text{LOC}} \times 1000 = \text{Defect Density/KLOC}
\]

DD/KLOC has been a measure of quality since the beginning of the 60’s but was first established in the 70’s with the book Software Metrics [20] that used DD/KLOC to predict the defects in a project based on size, where size is defined by LOC. DD has been questioned in several articles, mainly because defects has no universal definition [13]. The common definition of defect is a fault, but a fault will generate or not generate a failure in the system and as such a defect is not necessarily a good measure for quality of the system in regards of how operational the system is. In an article by Fenton and Neil they argue that DD is a product of academic research and not a good fit for industry situation in regards of content and scope [13].

2.3.1 Defects per man hour
To evaluate if TDD adds value to development projects at Decerno the metric Defects Density Per Hour (DD/h) will be calculated both for the case studies and in the projects at Decerno instead of DD/KLOC. DD/KLOC is dependent on many other factors such as programming language, the nature of the project and the size of the project. DD/h includes the size of the project by how many man hours it takes to the first release of the software and is less dependent on external factors, this also mitigate the need to include how many developers was needed to develop the software. The measure DD/h have been suggested in consultation with Mats Höstbo (a former employee at the
company) and based on the article by Kaner and Bond [27] Software Engineering Metrics: What Do They Measure and How Do We Know? together with the book Metrics and models in Software Quality Engineering by Kan [26].

With 25 years of experience from quality management in the industry and DD/h as the main tool to successfully assess the quality of large IT projects, Höstbo gives DD/h more relevance in an industrial context than DD/KLOC have. One of the gains with TDD is that the automated test suites is reused every time a new functionality is introduced to the system and with DD/h it is possible for project managers and companies to estimate when a system is ready for delivery by comparing previous DD/h values with previous external quality values such as acceptance tests performed by the customer. With DD/KLOC it is not as simple because different projects might use different programming languages and sometimes multiple languages in the same project. This makes it hard to see DD/KLOC as a relevant measure of software quality in an industrial context.

2.4 Defects

There is no unique definition of defects in the existing literature and the articles that define defects are not detailed about how they are counted when calculating DD/KLOC. Fenton and Neil defines defects as faults in the development phase that may or may not lead to failures in functionality [13]. Despite the different definitions on defects all the previous articles with TDD and DD/KLOC measure defects before the first release.

2.4.1 Defects defined at Decerno

In the company’s manual for project managers you can read:

“It is important that the developer ensures that what is developed corresponds to the requirement of the customer before the functionality is marked as completed. This requires that an understandable requirement specification is available. Otherwise it is not possible to determine what is a deviation”

Defects are an unplanned behavior of the system, a deviation from the customer defined requirement specification according to the company’s view.

3 METHOD

The methodologies used in this master thesis are systematic literature review (SLR) with guidance from Kitchenham [29], a survey created with Google forms and data extraction from the company’s project tool JIRA. A SLR is an approach to identifying, interpreting, and evaluating available research relevant to a specific research question or topic area, according to Kitchenham. The SLR approach was selected because of the differences in the articles regarding programming languages, the settings of the case studies and measured metrics. The survey was conducted to collect data regarding the company’s capacity to implement TDD in the development process and the data extraction from JIRA was to calculate the possible change in DD/h for projects in the company if developed with TDD.

3.1 Survey

A survey was sent out to the company’s employees to catch the state of development process today (Appendix A). The survey was divided into three parts targeting different parts of software development; Agile development, TDD and project managers. The respondents position at the company was recorded as developer or project manager.

3.2 Literature study

The following databases was used to search for articles and reports; KTH Primo, Google scholar and The ACM digital library. In these databases, the following keywords, in combination and as stand-alone words were used; Waterfall, ITL, TDD, iterative test last, test-driven development, agile development, agile model-driven development, agile, project management, case studies, acceptance test, acceptance test-driven development, ATDD.

To be able to evaluate if TDD adds value to software projects, articles with case studies focusing on comparing TDD with ITL were systematically reviewed. The examined metrics in the case studies were; number of defects and the total time needed to complete the software. The data was then used to calculate the DD/h as seen in 2.3.1. Out of 23 papers examined 18 included metrics on defect rate and 15 on time.

The examined cases are listed in Table 2 and will be referred to as report #1-23 in the tables and text to follow. The plus and minus signs in the table represents increase and decrease in comparison with ITL.
Table 2. Set: Setting (P = Professional, A = Academic), DD: Defect Density, T: Time needed to first release, - : Decrease, + : Increase, = : No change

<table>
<thead>
<tr>
<th>#</th>
<th>Title of the report</th>
<th>Set</th>
<th>DD</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Assessing TDD at IBM [36]</td>
<td>P</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>2</td>
<td>A dissection of the TDD process [16]</td>
<td>P = N/A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>TDD as a defect-reduction practice [57]</td>
<td>P</td>
<td>-</td>
<td>=</td>
</tr>
<tr>
<td>4</td>
<td>Realizing quality improvement through TDD [40]</td>
<td>P</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>5</td>
<td>A structured experiment of TDD [18]</td>
<td>P</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>6</td>
<td>Evaluating the efficacy of TDD [7]</td>
<td>P</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>7</td>
<td>Productivity of TDD [10]</td>
<td>P</td>
<td>=</td>
<td>+</td>
</tr>
<tr>
<td>8</td>
<td>Long-term effects of TDD [35]</td>
<td>P</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>9</td>
<td>An experimental evaluation of the effectiveness and efficiency of the TDD [22]</td>
<td>A</td>
<td>N/A</td>
<td>-</td>
</tr>
<tr>
<td>10</td>
<td>Implications of TDD [28]</td>
<td>A</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>A comparative case study on the impact of TDD on program design and test coverage [51]</td>
<td>P</td>
<td>-</td>
<td>N/A</td>
</tr>
<tr>
<td>12</td>
<td>The effect of experience on the TDD development process [39]</td>
<td>PA</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>13</td>
<td>Evaluating TDD in an industry-sponsored capstone project [55]</td>
<td>A</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>14</td>
<td>Towards empirical evaluation of TDD in a university environment [41]</td>
<td>A</td>
<td>=</td>
<td>N/A</td>
</tr>
<tr>
<td>15</td>
<td>A prototype empirical evaluation of test driven development [19]</td>
<td>P</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>16</td>
<td>A longitudinal study of the use of a test-driven development practice in industry [45]</td>
<td>P</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>17</td>
<td>The impact of test-driven development on software development productivity [33]</td>
<td>P</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>18</td>
<td>The impact of test-driven development on the evolution of a reusable framework of components [58]</td>
<td>P</td>
<td>-</td>
<td>N/A</td>
</tr>
<tr>
<td>19</td>
<td>Preliminary analysis of the effects of pair programming and test-driven development on the external code quality [32]</td>
<td>A</td>
<td>=</td>
<td>N/A</td>
</tr>
<tr>
<td>20</td>
<td>Empirical investigation towards the effectiveness of test first programming [24]</td>
<td>A</td>
<td>=</td>
<td>N/A</td>
</tr>
<tr>
<td>21</td>
<td>Impact of using test-driven development [56]</td>
<td>A</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>22</td>
<td>Comparison between test driven development and waterfall development in a small-scale project [31]</td>
<td>A</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Ratio of reports supporting decrease in DD: 0.70
Ratio of reports supporting increase in time: 0.35

3.3 Data from JIRA

The data from the company’s project manager tool JIRA was extracted with the help from three project managers at the company. The data consisted of total time for development, time spent on fixing defects and number of defects. In JIRA defects are tagged as reported, in progress or completed. Only defects with the tag “in progress” or “completed” were recorded as a defect in the projects. This is because not all the reported defects are in fact defects but sometimes the customer reports defects that are missing functionality due to that the functionality was not specified in the requirement specification.

4 RESULT

The data was collected in two separate sets. The first set of data is the results from a survey that was sent to the employees at the company, the second set of data is from the company’s ongoing development projects.

4.1 Literature review

In the literature review the distribution of the case studies is represented in Figure 3. 8.7% of the studies did not include data for DD. For change in time the number is 35%.

Figure 3. Summary of the articles in section 3.2 presenting the change in defect density and time between ITL and TDD.

70% of the articles in the literature review supports that TDD decreases DD and 35% supports the assumption that TDD increases time needed to the first release of the software developed.

61% of the articles included case studies performed in a professional setting such as at a company or with professional programmers. 35% was held in an academic setting at a university with undergraduates and 4% of the articles was held in a semi setting in both an academic and industrial setting.

The extra time needed to develop with TDD is minimal according to the reports. In report 1 the authors observed a decrease in DD by 50% and with only minimal impact in time needed to complete the software. The same observation can be found in report 3 with a decrease in DD with 40% and an unchanged time needed to complete the software. In addition to this, in some of the articles the change in time needed to develop are estimates of the project managers [report 4 and report 6].

One of the issues addressed in several of the reports are the scope of the case studies. Not many of the reports include case studies longer than a week. In these reports the authors argues that the long-term effects of TDD could be even more important than the results supporting decrease in DD in the first release of the software. Both report 2 and report 3 discuss this. This is something that report 8 and report 17 address by conducting long case studies (three and five years long) at companies. The authors of report 8 presents evidence that TDD simplify the maintenance of the software and the authors of report 17 came to the same conclusion, both reports contain evidence that TDD also decrease the degree to which code complexity increase the longer a software is operational and new functionalities are added.
A few of the case studies in the reports had both professionals and student in the same case study and the report #12 were conducted both in an academic and an industrial setting. The authors of this report concluded that a significant difference between professionals and students exists in the ability to practice TDD.

4.2 Survey
In the first data collection, a survey was sent out to developers and project managers at the company. 13 out of 35 receivers responded on the survey. Three of the respondents worked as project managers and 10 worked as developers at the company, referred to as PM1-PM3 and D1-D10. The survey consisted of 18 questions targeting both developers and project managers and three questions only targeting project managers. The first two parts; Part I: Scrum at Decerno" and “Part II: Testing" targeted both developers and project managers and the last part “Part III: Project managers” targeted only the project managers.

4.2.1 Part I: Scrum at Decerno
The first part of the survey covered the employees view on how agile the company is today in the development process. Most of the respondents said that the company’s projects are managed with the Scrum methodology, although 9 of the respondents said that the method is not practiced to 100%. The part that was not in line with Scrum was the sprints and how they experience the sprints more flexible as described in the Scrum methodology. PM2 said that “We practice most of the parts from Scrum but one part that we do not practice is sprint demos. We also take methods from other Agile methodologies such as pair programming from extreme Programming.”

The respondents also experienced that Scrum helps with the structure of everyday tasks. At an individual level the developer experienced the standup meetings as the most central part of Scrum. The project managers experienced that revaluation of the tasks in the sprint is the central part of Scrum. As a team, the respondents said that Scrum helps the team to work towards a common goal. D5 answered that “We work individually on tasks in the sprint. But in the end of the sprint we try to work together (pair programming or group programming) and help each other to complete all tasks in the sprint before it ends. We also have daily Scrum-meetings to check up on how everyone is doing and team members is free to ask questions or ask for help (if they have not already received help)”

Overall the respondents experienced Scrum as a good agile framework that offers flexibility, clearly stated goals and a good picture of the overall progress of the project. The disadvantage of Scrum was mainly experienced as stress according to the developers [D1, D2, D4]. They answered that it’s caused by the sprints. Especially at the end of the sprints when tasks are not fully completed. D4 said that “Stress is caused at the end of sprints if tasks in the sprint is not completed and the end of a sprint is close.”

While the project managers were more concerned with the planning of the projects, PM1 stated that “Some developers lack the ability to plan the effort needed in the future”.

4.2.2 Part II: Testing
In the second part of the survey the respondents answered on questions regarding testing at the company and the general idea of what tests contribute with, together with questions regarding TDD and the respondents view on implementing TDD at the company. Testing at the company today is performed, according to eight of the respondents, as test-last. Five of the respondents said that test cases were written prior to the functional code in a test-first approach. The different views are because the project managers per project decide how the code should be tested. On the question if the code is tested sufficient 8 responded “yes” and 5 responded “No, the code should be tested more”. As presented in Figure 4 none of the project managers though that the code in their projects needed to be tested more.

![Figure 4. Respondents answer on the question if the code at the company is tested sufficient today.](image)

The overall view on what tests contribute with is quality assurance in regards of functionality according to the developers. The project managers view on tests are that they add responsibility to the individual developer and to function as a check for when the specific task is done. Which in turn generate satisfied customer according to a project manager at the company.

Before the section that presented questions about TDD the respondents had to answer if they knew about TDD. All but PM3 knew about TDD at the time of the survey. PM3 was given the opportunity to read a short summary about TDD before answering the questions regarding TDD in part II of the survey.

The questions in part II focused on the company’s ability to implement TDD and the respondents view if TDD is to be introduced in the company’s development process. One of the questions focused on the ability to write unit tests, a fundamental part of TDD. The respondents could answer on a scale from zero to 10, where zero is “no experience with unit testing” and 10 is “Experienced in writing unit tests”. The result of the question can be seen in Figure 5. If TDD is to be introduced at the company the respondents are willing to write their own unit tests but
The survey also included questions regarding the respondents experience with pair programming and if they accept to practice pair programming if TDD is introduced at the company. The result from the first question can be seen in Figure 6. All the respondents answered “Yes” to the second question.

The survey also included questions regarding the respondents experience with pair programming and if they accept to practice pair programming if TDD is introduced at the company. The result from the first question can be seen in Figure 6. All the respondents answered “Yes” to the second question.

The result from the last question of part II of the survey “Are you positive to the idea of introducing TDD in your active project?” is presented in Figure 7.

4.2.3 Part III: Project Managers

The respondents that worked as project managers at the company answered three questions in the last part of the survey.

The first question focused on customer requirements and how to translate them into system specification. The process at the company includes meetings, requirements analysis and reformulation of the requirements if needed. The meetings with the customer is used to document and to establish an understanding between the developers and the customer how to define the functionality. The analysis and reformulation follows the meetings.

The second question targeted how requirements changes is handled at the company. All project managers said that they discuss the changes with the customer and implement them as planned with the customer, usually in the tool Jira.

On the last question “How involved is the customer in the development process?” the general view from the project managers was that the customer presents input and ideas on the regular meetings. They are also active as product owner and are responsible for the acceptance tests.

4.3 Defect density

Data was extracted from five of the case studies in the literature study. The reports that are not included in the table were excluded because they did not include numbers on defects and development time, which are both needed to calculate DD/h. The data in Table 3 is from reports number 4, 5, 9, 13 and 22. The data extracted is DD/h using ITL, DD/h using TDD, change in DD/h using TDD and change in development time needed for first release of the software using TDD.

Instead of the frequently used metric DD/KLOC, the optional metric DD/h, is presented in Table 3. DD/h is calculated as presented in 2.3.1. The metric DD/h is better fitted for real life projects because of the independency of complexity and programming language used. Report number four did not include specific numbers on defects and time needed to calculate DD/h. The report is still included in the table because change in time is stated in the report.

Table 3. Calculated DD/h for both ITL and TDD. The change in DD and development time between ITL and TDD are also presented in the table.

<table>
<thead>
<tr>
<th>Report</th>
<th>DD/h (ITL)</th>
<th>DD/h (TDD)</th>
<th>Change in DD</th>
<th>Dev. Time with TDD</th>
</tr>
</thead>
<tbody>
<tr>
<td>#4</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>+24%</td>
</tr>
<tr>
<td>#5</td>
<td>1.07</td>
<td>0.38</td>
<td>-64%</td>
<td>+16%</td>
</tr>
<tr>
<td>#9</td>
<td>0.38</td>
<td>0.30</td>
<td>-21%</td>
<td>+52%</td>
</tr>
<tr>
<td>#13</td>
<td>0.46</td>
<td>0.40</td>
<td>-13%</td>
<td>+40%</td>
</tr>
<tr>
<td>#22</td>
<td>0.12</td>
<td>0.11</td>
<td>-13%</td>
<td>-25%</td>
</tr>
<tr>
<td>Avg. values</td>
<td>0.51</td>
<td>0.30</td>
<td>-41%</td>
<td>+21%</td>
</tr>
</tbody>
</table>

Data was also collected from three projects at the company. The data consisted of defects, man hours needed to develop the software and time spent to fix defects. The result is presented in Table 4. The column “DD/h (TDD)”
holds possible values for the company’s projects if they had been developed with TDD. The values are calculated based on the average percentage change in DD/h from Table 3 as following:

\[
\frac{DD}{h} = \text{avg. value of *Change in DD* from table III)}
\]

Table 4. Data extracted from JIRA. The values in the column to the far right are possible values that could have been achieved if the company had developed the projects with TDD instead of ITL.

<table>
<thead>
<tr>
<th>Project</th>
<th>Defects</th>
<th>Dev. Time with ITL</th>
<th>DD/h (ITL)</th>
<th>DD/h (TDD)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>928</td>
<td>8615 h</td>
<td>0.11</td>
<td>0.06</td>
</tr>
<tr>
<td>B</td>
<td>35</td>
<td>1088 h</td>
<td>0.03</td>
<td>0.02</td>
</tr>
<tr>
<td>C</td>
<td>414</td>
<td>5315 h</td>
<td>0.08</td>
<td>0.05</td>
</tr>
<tr>
<td>Total</td>
<td>1377</td>
<td>15018</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Avg. values</td>
<td>459</td>
<td>5006 h</td>
<td>0.07</td>
<td>0.04</td>
</tr>
</tbody>
</table>

The number of man hours needed to fix the defects in project A, B and C at the company were 1620 hours for project A and 80 hours for project B and 1550 for project C. This reveals that 21.5% of the development time is spent on fixing defects in the company’s projects. Defects were reported faults in the system detected during the development of the system from both the customer and the company. Only reported faults that was marked as “in progress” and “completed” are included in Table 4. Some of the faults reported was due to missing functionality that the customer has not stated in the system requirements and was never a defect but a non-specified functionality.

5 DISCUSSION

5.1 Survey

The answers collected during the survey give the context of a system development company that practice Scrum but adjust the Scrum methodology to fit needs of the company, much like the authors of the agile manifesto had in mind when they wrote the manifesto. The company also holds the expertise in-house to write tests for the functional code, something that will be needed if they are to implement TDD in their developing process. If the developers in the company are familiar with automated tests is not revealed in the survey because of lack of coverage in that area. Furthermore, a certain headroom exists in the company to test more than they do for now according to 50% of the developers. The overall experience of Scrum in the company was positive but the project managers seemed to be a bit more positive than the developers. When the project managers only saw advantages in using Scrum few of the developers mentioned stress as one of the disadvantages of Scrum, this was nothing that was noted in the article by Causevic et al. [9], in fact they only presented advantages with Scrum. The project managers at the company said that Scrum mitigated stress because of the planning of stories in the sprints. These sprints caused most of the stress in projects according to the developers which contradicts the view that the project managers had on sprints.

Regarding testing at the company the general idea is that projects use ITL today. It was also revealed that projects in general could be tested more according to the developers which in turn could help with the attitude if TDD is to be introduced in the company. Half of the developers at the company stated that they think they are experienced in writing unit tests which should be enough to practice pair programming if the company introduce TDD. The confidence in writing unit tests was also high as all the respondents answered that they would not mind writing their own unit tests. To help the company to adopt to TDD, pair programming from XP can be used to mitigate the lowered productivity that Dybå and Dingsoyr states as one of the seven major problems that limiting industrial adaption of TDD. In the survey 82% of the developers have experience with pair programming, unfortunate only 10 of the company’s developers answered on the survey and there is no guaranty that the share is this high if all the developers at the company had responded to the survey.

In the section of the survey targeting only the project managers nothing relevant for the sub- and main question was revealed. Meetings with the customer to decide on requirement specification is frequently held and that they are often revised during the development process. The customer is involved in the development process mainly by the meetings but also by external quality assurance with acceptance tests of the system.

5.2 Defect density per hour

A comparison, with regards of DD/h, between the projects at Decerno and the case studies in the reports shows that projects developed with ITL at Decerno have 76% less DD/h than the case studies projects developed with TDD. This seems to be a huge difference in software quality generated by the company, but keep in mind that only one of the five case studies were conducted in an industrial setting and included professional programmers and as report #12 concluded, there is a significant difference between students and professionals in the ability to practice TDD.

In this master thesis DD/h is used to measure the quality and success of a software project. To calculate DD/h the numbers of defects in the project and total development are needed. Only five of the 23 reports in the literature review included both. The DD/h while using ITL at the company is based on numbers from the company, but DD/h when developing with TDD is only estimates based on the change in DD/h calculated from the case studies as seen in table III. The average DD/h (TDD) for projects at the company was calculated as following:

\[
0.07 \frac{DD}{h} = \left( \frac{0.07 \frac{DD}{h}}{100} \right) \times 41 = 0.04 \frac{DD}{h}
\]
From the data presented in Table 3 and Table 4 the development time and number of defects in project A, B and C at the company changes according to Table 5. The values in the column Development time with TDD were calculated as following:

\[
(\text{Dev. Time with ITL}) + (\text{avg. dev. Time with TDD}) = \text{Dev. Time with TDD}.
\]

The defects in the TDD column are calculated as following:

\[
\left(\frac{\text{time}}{100}\times 21\right) + \text{avg. } \frac{\text{DD}}{h} \text{ with TDD}
\]

Table 5. Possible number of defects and extended development time if the company had developed the projects with TDD instead of ITL.

<table>
<thead>
<tr>
<th>Project</th>
<th>ITL</th>
<th>TDD</th>
<th>Dev. Time with TDD</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>928</td>
<td>417</td>
<td>10424</td>
</tr>
<tr>
<td>B</td>
<td>35</td>
<td>53</td>
<td>1316</td>
</tr>
<tr>
<td>C</td>
<td>414</td>
<td>257</td>
<td>6431</td>
</tr>
<tr>
<td>Total</td>
<td>1377</td>
<td>727</td>
<td>18171</td>
</tr>
</tbody>
</table>

The company spend 3250 man hours to fix the 1377 defects in the three projects. The company’s capacity to fix defects is calculated as following:

\[
\frac{1377 \text{ defects}}{3250 \text{ h}} = 0.42 \frac{\text{Defects}}{h}
\]

From the column TDD in Table 5 the time needed to fix the 727 defects is calculated as following:

\[
\frac{727 \text{ defects}}{0.42 \frac{\text{defects}}{h}} = 1731 \text{ h}
\]

1731 man hours are needed to fix the 727 estimated defects generated in the projects if developing with TDD instead of ITL. If the company developed the projects with TDD possible reduction in time spent on fixing defects would be 1519 man hours. That is 8% of the total development time spent on fixing defects instead of the 21.5% it is now with ITL. This will save the company and the customer resources in both time and money. The result is in line with the literature review regarding that TDD is to prefer over ITL but with DD/h instead of DD/KLOC it is easier to compare projects regardless of which programming language is used in the projects, the complexity of the project or number of team members needed to complete the software. The downside for this master thesis is the lack of real numbers for the company’s DD/h when develop with TDD.

6 TREATS TO VALIDITY

One issue to address is the choice of measure to calculate quality in this master thesis. The frequently used DD/KLOC was not seen as a good fit to answer the main question of this master thesis. The optional measure DD/h seems to fit industrial needs better but is a recommendation and product from Höstbo, a professional in the industry and not a researcher. No projects at the company were developed with TDD and because of this the results are at best estimates on how DD/h could change at the company if they used TDD instead of ITL.

The five articles that were used to calculate DD/h were rather limited case studies that lacks relevance in industrial context, much like Fenton and Neil’s article points out [13] the studies in the articles is too small in scope to have much relevance for the industry.

7 CONCLUSION

To be able to answer the main question of this master thesis a more relevant measure than the traditional DD/KLOC was needed. The solution was a measure that describes numbers of defects generated per man hour or defect density per man hour. This measurement suggests an answer to the research question “How to assess relevant quality changes in software development projects in an industrial context?”.

The answer to the main question “Does TDD contribute relevant positive changes to the business of software SME?” is still to be answered because the validity of the literature review in combination with the developed measure DD/h is not enough to suggest that TDD contributes with positive and relevant changes to the business of software development SME. The result is still an indicator to what is to be revealed if a more extensive study is to be conducted with the measurement DD/h.

7.1 Future work

The next step in proving that TDD impacts the industry in a positive way is to set up a clear definition of what a defect is. Also, most of the costly defects are detected after the first release of the software and as such, more studies should be performed in an industrial setting that measure DD/h both pre- and post-release in TDD projects and compare the post defects for TDD projects with post defects for ITL projects as suggested by Grigori [49]. If it can be proven that TDD can reduce post-release defects at the same level as pre-release then TDD could be practiced more extensively in the industry than it is today.
8 REFERENCES


[31] Lei Zhang, Shunsuke Akifuji, Katsumi Kawai,


APPENDIX: A – The Google survey

For the complete and intended structure of the digital survey please visit the address:

https://goo.gl/forms/slpE7np3QWpKV9sH2

Agila projekt och testdriven utveckling (TDD)

Denna enkät ingår i ett exjobb som utförs i samarbete med Decerno. Med svaren från denna enkät hoppas jag samla grundläggande information för att se om TDD är något som Decerno kan (om intresset finns) införa på ett enkelt sätt i sin utvecklingsprocess.

*Obligatorisk

1. "Jag är medveten om och godkänner att denna enkät och dess svar kan komma att publiceras i Daniel Gustavssons exjobb." *
   *Markera endast en oval.
   ○ Ja Hoppa till fråga 4 efter den sista frågan i detta avsnitt.
   ○ Nej Hoppa till fråga 3 efter den sista frågan i detta avsnitt.

2. Vilken roll har du på Decerno idag? *
   *Markera endast en oval.
   ○ Utvecklare
   ○ Projektledare
   ○ Annat

Vill du inte svara på enkäten?
Om du känner att du inte vill få dina ord hugget i sten men ändå är lite nyfiken på vad mitt exjobb går ut på eller kan tänka dig att hjälpa mig med mitt exjobb genom att prata lite om arbetsflödet och din roll på Decerno är du ändå varmt välkommen att kommna förbi mig på kontoret (Sitter bredvid Sten Bäckström) och prata lite "off the records" så att säga. Jag skulle vara dig evigt tacksam för din tid, oavsett om det är 5 minuter eller 1 timme du har möjlighet att prata :)

3. Tack för din tid! :D
   *Markera endast en oval.
   ○ Coolio! jag kommer förbi dig och berättar allt jag vet! :D Sluta fylla i det här formuläret.
   ○ Jag har tyvärr inte tid men önskar dig lycka till med ditt exjobb! Sluta fylla i det här formuläret.

Arbetsflödet på Decerno

4. Används Agila metoder för utveckling på Decerno enligt dig? *
   *Med Agila metoder menar jag t.ex. XP, TDD, Scrum, Crystal osv...
   *Markera endast en oval.
   ○ Ja Fortsätt till frågan 5.
   ○ Nej Fortsätt till frågan 12.

Agila metoder
Det är inte helt ovanligt att man idag plockar det "bästa" från olika Agila metoder för att bäst anpassa arbetsflödet till sin egen verksamhet, så markera gärna fler metoder om så är fallet i just ditt projekt.
5. Markera den eller de Agila metoder som används i det projekt du medverkar i nu:
Markera alla som gäller.

☐ Scrum
☐ Crystal
☐ Extreme Programming
☐ Agile Model Driven Development
☐ Adaptive Software Development
☐ Pragmatic Programming
☐ Dynamic Systems Development Method
☐ Feature Driven Development
☐ Internet-Speed Development

Agila metoder

6. Tycker du att den Agila metoden som används i ditt projekt utövas till 100%? *
Markera endast en oval.

☐ Ja Fortsätt till frågan 7.
☐ Nej Fortsätt till frågan 9.

Agila metoder

7. Kan du i korta drag beskriva hur den Agila metoden styr ditt dagliga arbete? *

____________________________________________________________________
____________________________________________________________________
____________________________________________________________________
____________________________________________________________________

8. Kan du i korta drag beskriva hur den Agila metoden styr din projektgrupps dagliga arbete? *

____________________________________________________________________
____________________________________________________________________
____________________________________________________________________
____________________________________________________________________

Fortsätt till frågan 10.

Agila metoder
9. Vad är det som INTE utövas i projektet som är en del av den Agila metoden? *


Fortsätt till frågan 7.

Agila metoder

10. Vad är fördelarna med den Agila metoden som används i det projektet du är med i? *


11. Vad är nackdelarna med den Agila metoden som används i det projektet du är med i? *


Testning av programkod

12. När testas kod i det projekt du är med i? *

Markera endast en oval.

☐ Testfall skrivs innan någon funktionskod skrivs (Testdriven utveckling)
☐ först när en funktion är helt implementerad testas funktionskoden för den specifika funktionen
☐ Endast systemet testas och endast när första MVP är klar
☐ Jag vet inte hur och när kod testas i projektet

13. Tycker du att koden i ditt projekt testas på en rimlig nivå innan leverans till kund? *

Markera 2 om du tycker att koden i ditt projekt testas tillräckligt som det är nu

Markera endast en oval.

1 2 3

☐ Nej, den testas för lite ☐ ☐ ☐ Nej, den testas för mycket
14. Vad bidrar testerna med i projektet enligt dig? *

Testdriven utveckling

15. Vet du vad testdriven utveckling innebär?  
Markera endast en oval.

☐ Ja 
Fortsätt till frågan 16.

☐ Nej 
Fortsätt till Vad är testdriven utveckling (TDD)?.

Vad är testdriven utveckling (TDD)?

Testdriven utveckling innebär att man alltid skriver enhetstest först och sen funktionskoden som ska testas.


Tanken med TDD är att det är själva testerna som är centrala i själva utvecklingen, de ska ligga som underlag för framtida beslut gällande design och implementering av funktionalitet. Då testerna skrivs innan man skriver funktionskod så bidrar också TDD till att fungera som specifikation för system som ska utvecklas.

Utveckling med TDD tar upp till 18% längre tid att slutföra men ger i regel ett system som är lättare att införa nya funktioner i och som har tester som täcker nästan hela systemet när det väl är klart för leverans.

Testdriven utveckling (TDD)

16. Har du erfarenhet av att skriva enhetstester (unit tests)? *  
Markera endast en oval.

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>Ja, väldigt mycket</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nej, vad är enhetstester?</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>Nej</td>
</tr>
</tbody>
</table>

17. Om TDD skulle införas i projekt du är med i: Skulle du kunna tänka dig att testa din egen kod? (skriva enhetstester innan du börjar programmera) *  
Markera endast en oval.

☐ Ja

☐ Nej

18. Om TDD skulle införas i projekt du är med i: Skulle du kunna tänka dig att få din kod testad av någon annan? (Enhetstester skrivs innan du börjar programmera) *  
Markera endast en oval.

☐ Ja

☐ Nej
19. Har du erfarenhet av parprogrammering? * 
*Markera endast en oval.

- [ ] Ja
- [ ] Nej

20. Om TDD skulle införas i ditt projekt, skulle du kunna tänka dig att köra parprogrammering i början? * 
*Markera endast en oval.

- [ ] Ja
- [ ] Nej

21. Skulle du ställa dig positivt inställd till att införa TDD i ditt projekt? * 
*Markera endast en oval.

- [ ] Ja
- [ ] Nej

**Projektledare**

22. Är du eller har du varit projektledare inom Decerno? * 
*Markera endast en oval.

- [ ] Ja Fortsätt till frågan 23.
- [ ] Nej Sluta fylla i det här formuläret.

**Projektledning**

Följande frågor är helt valbart att svara på. Dock är svaren här till stor hjälp för mig i mitt exjobb.

23. Hur arbetar ni idag för att ta kundens krav till systemspecifikation?

________________________________________________________________________
________________________________________________________________________
________________________________________________________________________
________________________________________________________________________

24. Hur hanterar ni önskmål om förändringar i systemet från kund?

________________________________________________________________________
________________________________________________________________________
________________________________________________________________________
________________________________________________________________________
25. Hur involverad i utvecklingen är kunden?
Tar kundens medverkan i projektet slut efter att kravspecifikationen är klar? eller är de aktiva i projektet på annat sätt också?
