UNIVERSITY OF HELSINKI

https://helda.helsinki.fi

Effects of Test-Driven Development : A Comparative Analysis of Empirical Studies

Mäkinen, Simo; Münch, Jürgen

Winkler, Dietmar; Biffl, Stefan; Bergsmann, Johannes

2014

http://hdl.handle.net/10138/42741

Mäkinen, S & Münch, J 2014, Effects of Test-Driven Development : A Comparative Analysis of Empirical Studies. in D Winkler, S Biffl & J Bergsmann (eds), Software Quality. Model-Based Approaches for Advanced Software and Systems Engineering : 6th International Conference, SWQD 2014, Vienna, Austria, January 14-16, 2014. Proceedings. Lecture Notes in Business Information Processing, vol. 166, Springer-Verlag, pp. 155-169, Software Quality. Model-Based Approaches for Advanced Software and Systems Engineering, Vienna, Austria, 14/01/2014. https://doi.org/10.1007/978-3-319-03602-1_10

Downloaded from Helda, University of Helsinki institutional repository. https://helda.helsinki.fi
This is an electronic reprint of the original article.
This reprint may differ from the original in pagination and typographic detail.
Please cite the original version.

Effects of Test-Driven Development: A Comparative Analysis of Empirical Studies

Simo Mäkinen and Jürgen Münch

University of Helsinki,
Department of Computer Science,
P.O. Box 68 (Gustaf Hällströmin katu 2b),
FI-00014 University of Helsinki, Finland
{simo.makinen,juergen.muench}@cs.helsinki.fi

Abstract. Test-driven development is a software development practice where small sections of test code are used to direct the development of program units. Writing test code prior to the production code promises several positive effects on the development process itself and on associated products and processes as well. However, there are few comparative studies on the effects of test-driven development. Thus, it is difficult to assess the potential process and product effects when applying test-driven development. In order to get an overview of the observed effects of test-driven development, an in-depth review of existing empirical studies was carried out. The results for ten different internal and external quality attributes indicate that test-driven development can reduce the amount of introduced defects and lead to more maintainable code. Parts of the implemented code may also be somewhat smaller in size and complexity. While maintenance of test-driven code can take less time, initial development may last longer. Besides the comparative analysis, this article sketches related work and gives an outlook on future research.

Key words: test-driven development; test-first programming; software testing; software verification; software engineering; empirical study

1 Introduction

Red. Green. Refactor. The mantra of test-driven development [1] is contained in these words: *red* refers to the fact that first and foremost implementation of any feature should start with a failing test, *green* signifies the need to make that test pass as fast as possible and *refactor* is the keyword to symbolize that the code should be cleaned up and perfected to keep the internal structure of the code intact. But the question is, what lies behind these three words and what do we know about the effects of following such guidelines? Test-driven development reshapes the design and implementation of software [1] but does the change propagate to the associated software products and in which way are the processes altered with the introduction of this alternative way of development? The objective here was to explore these questions and to get an overview of the observed effects of test-driven development.

© Springer 2014. This is the author's version of the work. The definite version was published in Proceedings of the 6th International Conference Software Quality Days (SWQD 2014), Vienna, Austria, January 14-16, 2014. Proceedings, Lecture Notes in Business Information Processing. Springer, 2014. The final version is available at link.springer.com.

To seek out answers to these questions, we performed an integrative literature review and analyzed the experiences from existing empirical studies from the industry and academia that reported on the effects of test-driven development. A quality map containing ten quality attributes was formed using the data from the studies and the effects were studied in greater detail. The results of the comparative analysis suggest that test-driven development can affect quality attributes. Positive effects were reported in particular for defect densities and maintainability in industrial environments. Some studies, however, reported an increased development effort at the same time. Many of the other observed effects are neutral or inconclusive.

This paper has been divided into five sections. Related work is presented in Section 2 and the research method for this review is described in Section 3. The effects derived from the reviewed primary studies are detailed in Section 4 and finally Section 5 concludes the paper.

2 Related Work

Test-driven development has been subject of reviews before. For instance, Turhan et al. [2] performed a systematic literature review on test-driven development that highlighted internal and external quality aspects. The review discovered that during the last decade, there have been hundreds of publications that mention test-driven development but few report empirically viable results.

Based on the reports, Turhan et al. were able to draw a picture of the overall effect test-driven development might have. They categorized their findings as internal and external quality, test quality and productivity. Individual metrics were assigned to these categories which were labeled to have *better*, *worse*, *mixed* or *inconclusive* effects. The rigor of each study was assessed by looking at the experimental setup and studies were further categorized into four distinct rigor levels.

Results from the review of Turhan et al. for each of the categories indicate that the effects vary. Internal quality—which consisted of size, complexity, cohesion and other product metrics—was reported to increase in several cases but more studies were found where there either was no difference or the results were mixed or worse. External quality, however, was seen to be somewhat higher as the majority of reviewed studies showed that the amount of defects dropped; relatively few studies showed a decrease in external quality or were inconclusive. The effect of test-driven development on productivity wasn't clear: most industrial studies reported lowered productivity figures while other experiments had just the opposite results or were inconclusive. Surprisingly, Turhan et al. conclude that test quality, which means such attributes as code coverage and testing effort, was not superior in all cases when test-driven development was used. Test quality was considered better in certain studies but some reported inconclusive or even worse results.

A few years earlier, Jeffries and Melnik wrote down a short summary of existing studies about test-driven development [3]. The article covered around twenty

studies both from the industry and academia, describing various context factors such as the number of participants, programming language and the duration for each study. The reported effects were categorized into *productivity* and generic *quality effects* which included defect rates as well as perceptions of quality.

Jeffries and Melnik summarize that in most of the industrial studies, more development effort was needed when the organizations took test-driven development into use. In the academic environment, effort increases were noticed as well but in some academic experiments test-driven development was seen to lead to reduced effort levels. As for the quality effects, a majority of the industrial studies showed a positive effect on the amount of defects and in certain cases the differences to previous company baselines were quite significant. Fewer academic studies reported reduced defect rates and the results were not quite as significant; in one, defect rates actually went up with test-driven development.

Recently, Rafique and Mišić [4] gathered experiences from 25 test-driven development studies in a statistical meta-analysis which focused on the dimensions of external quality and productivity. Empirical results from existing studies were used as much as data was available from the primary studies. The studies were categorized as academic or industrial and the development method of the respective control groups was noted as well. It seemed to matter which development method the reference group was using in terms of how effective test-driven development was seen to be.

Rafique and Mišić conclude that external quality could be seen to have improved with test-driven development in bigger, and longer, industrial projects but the same effect was not noticed in all academic experiments. For productivity, the results were indecisive and there was a bigger gap between the academic experiments and industrial case studies than with external quality. Desai et al. [5] came to a similar conclusion in their review of academic studies: some aspects of internal and external quality saw improvement but the results for productivity were mixed—experience of the students was a seen as a factor in the experiments.

All of the previous reviews offer valuable insights into the effects of test-driven development as a whole by gathering information from a number of existing studies. While the research method is similar to the aforementioned reviews, in this review the idea is to extend previous knowledge by breaking down the quality attributes to more atomic units as far as data is available from the empirical studies. We expect that this will lead to deeper understanding of the effect test-driven development has on various attributes of quality.

3 Research Method

Empirical studies in software engineering can be conducted by using research methods which support the collection of empirical findings in various settings [6]. Runeson and Höst write that surveys, case studies, experiments and action research serve different purposes: surveys are useful as descriptive methods, case studies can be used for exploring a phenomena whereas experiments can at best be explanatory while action research is a flexible methodology that can be used

for improving some aspects that are related to the research focus [6]. Test-driven development has been the object of study in a number of research endeavours that have utilized such methods.

Literature reviews can be used for constructing a theoretical framework for a study which helps to formulate a problem statement and in the end identify the purpose of a study [7]. In integrative literature reviews [8], existing literature itself is the object of study and emerging or mature topics can be analyzed in more detail in order to create new perspectives on the topic. Systematic literature reviews [9] have similar objectives as integrative reviews but stress that the review protocols, search strategies and finally both inclusion criteria and exclusion criteria are explicitly defined.

The research method in this study is an integrative literature review which was performed to discover a representative sample of empirical studies about the effects of test-driven development from the industry and academia alike. Creswell [10] writes that literature reviews should start by identifying the keywords to use as search terms for the topic. Keywords that were used were such as *test driven development*, *test driven*, *test first programming* and *test first*. The second step suggested by Creswell is to apply these search terms in practice and find relevant publications from catalogs and publication databases. Several search engines from known scientific publishers were used in the process, namely the keywords were entered into search fields at the digital libraries of the Institute of Electrical Engineers (IEEE), Association of Computer Machinery (ACM), Springer and Elsevier. The titles and abstracts of the first few hundred highest-ranked entries from each service were manually screened. Although the search was repeated several times with the aforementioned keywords with multiple search engines on different occasions, the review protocol wasn't entirely systematic since there wasn't a single, exact, search string and entries were not evaluated if they had a low rank in the search results.

A selection of relevant publications from a larger body requires that there is at least some sort of inclusion and exclusion criteria. Merriam [7] suggests that the criteria can for instance include the consideration of the seminality of the author, date of publication, topic relevance to current research and the overall quality of the publication in question. There was no strict filter according to the year of publication so the main criterion was whether the publication included empirical findings of test-driven development and presented results either from the industry or academia. The quality of publications and general relevance were used as exclusion criteria in some cases. While the objective was to gather all the relevant research, there was a limited number of publications that could be reviewed in greater detail due to the nature of the study.

After applying the criteria, 19 publications remained to be analyzed further. These publications, listed in Table 1, were conference proceedings and journal articles that had relevant empirical information about test-driven development. In 2009, Turhan et al. [2] identified 22 publications in their systematic literature review of test-driven development and 7 of these publications are also included in

Table 1. An overview of the publications included in the review and the effects of test-driven development on quality factors.

Author Name and Year	Context	Defects	Coverage	Complexity	Coupling	Cohesion	Size	Effort	External Quality	Productivity	Maintainability
Bhat and Nagappan 2006 [12]	Industry	x^+	x				x	x			
Canfora et al. 2006 [13]	Industry							x_-		x_-	
Dogša and Batič 2011 [14]	Industry	x^+		x			x	x_-	x^+	x_-	x^+
George and Williams 2003 [15]	Industry		x					x_-	x^+	x_-	
Geras et al. 2004 [16]	Industry	x	x					x		x	
Maximilien and Williams 2003 [17]	Industry	x^+					x				
Nagappan et al. 2008 [18]	Industry	x^+	x				x	x_-			
Williams et al. 2003 [19]	Industry	x^+					x				
Janzen and Saiedian 2008 [20]	Industry/Academia		x	x^+	x	x	x^+				
Madeyski and Szała 2010 [21]	Industry/Academia							x		x	
Müller and Höfer 2007 [22]	Industry/Academia		x				x	x	x	x	
Desai et al. 2009 [23]	Academia		x					x	x	x	
Gupta and Jalote 2007 [24]	Academia							x^+	x	x	
Huang and Holcombe 2009 [25]	Academia							x_-	x	x	
Janzen and Saiedian 2006 [26]	Academia		x	x	x		x	x	x	x	
Madeyski 2010 [27]	Academia		x				x				
Pančur and Ciglarič 2011 [28]	Academia		x	x					x	x	
Vu et al. 2009 [29]	Academia	x	x	x^+			x	x	x	x	
Wilkerson et al. 2012 [30]	Academia	x_-					x				

this integrative review but some of the previously identified publications remain outside the analysis.

The findings of the studies of this review were used to construct a map of quality attributes and the perceived effects noted in each study. Construction of the quality map proceeded so that an attribute was added to the map if a particular study contained empirical data about the attribute so the attributes were not predetermined. This map lead to a more detailed analysis of the individual attributes. The literature review itself was completed in 2012 [11].

4 Effects of Test-Driven Development

Test-driven development has had a role in certain empirical studies but how are the different sofware engineering areas affected in practice? This section describes the results of the literature review and gathers the information about the perceived effects from the empirical studies included in the review.

Quality attributes of products, processes and resources can be related to the internal quality of objects or to the external quality which requires the consideration of not just the object but its external circumstances as well [31]. These

attributes can be either direct and readily measured or indirectly composed of several different attributes.

In the review, empirical findings for 10 different, internal and external, quality attributes were gathered from the 19 reviewed research reports which covered the effects of test-driven development. The extracted attributes or qualities which are analyzed further below include the following: the amount and density of defects, code coverage, code complexity, coupling, cohesion, size, effort, external quality, productivity and finally maintainability.

An overview of the results indicates that various effects on the quality attributes have been recorded in the test-driven development studies. The quality map which summarizes the effects is illustrated in Table 1 where the included publications are sorted by author and the study context which here is either industry or academia or a mixture of both. Case studies and experiments with industrial professionals in their own environment are considered industrial studies. Controlled and quasi-controlled experiments with student subjects are categorized as academic.

Across all the studies in the review, there are a total of 73 reported quality attribute effects that are shown in the quality map. Out of these effects, 12 have been labeled as significant positive effects whereas there are 9 effects labeled significantly negative. The rest of the 52 effects are either neutral or inconclusive. Significant positive effects mean that there was a statistically significant difference between the development practices compared in the study and the difference was in favor of test-driven development or there was enough significant qualitative data that suggested a meaningful difference. Significant negative effects means that test-driven development was in similar terms considered worse than other development practices it was compared against. In the quality map, significant positive effects are denoted by a superscript plus symbol x^+ and significant negative effects by a subscript minus symbol x_-; an undecorated x symbol signifies inconclusiveness, neutrality or the fact that the attribute was merely mentioned in the study results but not necessarily contemplated further as was the case with the size of code products in many reports. The following in-depth analysis of the effects and individual quality attributes shows what kind of role test-driven development can play in the software engineering process. A coarse-grained aggregation of the results is given at the end of the section.

4.1 Defect Density and the Number of Defects

In test-driven development, test code is written before the implementation which could theoretically lead to a reduced number of defects as fewer sections of code remain untested. Indeed, several industrial case studies show a relatively large reduction of defects compared to sister projects in the organizations where the trials were carried out.

Defect densities have reduced in a number of organizations and projects of different size. At IBM, a team that employed test-driven development for the first time was able to create software that had only half the defect density of previous comparable projects, and on average there were only around four defects

per one thousand lines of code [15]. While the overall amount of reported defects dropped, there were still the same relative amount of severe defects as with the previous release that didn't use test-driven development methods [19]. Microsoft development teams were able to cut down their defect rates, as well, and the selected test-driven projects had a reduced defect density of sixty to ninety percent compared to similar projects [18]. In the telecommunications field, test-driven development seemed to slightly improve quality by reductions in defect densities before and especially after releases [14].

Outside the industry, test-driven development hasn't always lead to drastically better outcomes than other development methods. Geras et al. experimented with industry professionals [16] but in a setting that resembled an experiment rather than a case study and didn't notice much of a difference in defect counts between experimental teams that were instructed to use or not to use test-driven development in their task. Student developer teams reported in the research of Vu et al. [29] fared marginally better in terms of the amount of defects when teams were using test-driven development but it seems that process adherence was relatively low and the designated test-driven team wrote less test code than the non-test-driven team. In another student experiment, Wilkerson et al. [30] noticed that code inspections were more effective at catching defects than test-driven development; that is, more defects remained in the code that was developed using test-driven development than in the code that was inspected by a code inspection group.

4.2 Code Coverage

Covering a particular source code line requires that there is a corresponding block of executable test code that in turn executes the source code line. Besides the basic statement coverage, it is also possible to measure how well the test code covers code blocks or branches in the code [32]. Without test code, there is no coverage and the code product must be tested with other methods or left untested. Test-driven development should encourage the developers to write scores of tests which should lead to a high code coverage.

The message from the industry seems to be that development teams were able to achieve good coverage levels when they were using test-driven development. At Microsoft and IBM, block coverages were up to 80 and 90 percent for several projects which took advantage of test-driven development although for one larger project the block coverage was around 60 percent [18]. In the longitudinal case study of Janzen et al. [20], coverage for the products of an industry partner were reported to be on the same high levels as the projects at Microsoft and IBM. George and Williams observed similar coverage ratings in a shorter industry experiment earlier [19].

When development teams are writing tests after the implementation and not focusing on incremental test-driven development, coverage ratings tend to be lower although the team's prior exposure to test-driven development can affect the way the developers behave in future projects [20]. Experience of the developers in general seems to be a factor in explaining how individual developers

work and how well they're able to adhere to the test-driven development process [22, 27]. Students who are more unfamiliar with the concept might not be able to achieve as high a coverage as their industry peers [29]. Very low coverage rates might be a sign that design and implementation is not done incrementally with the help of tests.

Good coverage doesn't necessarily mean that the tests are able to effectively identify incorrect behavior. Mutation testing consists of transforming the original source code with mutation operators; good tests should detect adverse mutations to the source code [32]. Pančur and Ciglarič [28] noticed in their student experiment that even though the branch coverage was higher for the test-driven development students, the mutation score indicator was actually worse than the score of other students who were developing their code with a test-last approach. In another student experiment, the mutation score indicators were more or less equal [27].

4.3 Complexity

Complexity measures of code products can be used to describe individual code components and their internal structure. For instance, McCabe's complexity [33] is calculated from the relative amount of branches in code while modern views of complexity take the structure of methods and classes into account [34]. Because code is developed in small code fragments with test-driven development, a reduction in the complexity of code is possible.

Reductions in complexity of classes have been observed both in the industry and academia but not all results are conclusive. Classes created by industrial test-driven developers seem to be more coherent and contain fewer complex methods [20] or the products seem less complex overall [14]. Student developers have on occasion constructed less complex classes with test-driven development [20, 29] but in some cases the differences in complexity between development methods have been small [28].

4.4 Coupling and Cohesion

Coupling and cohesion are properties of object-oriented code constructs and measure the interconnectivity and internal consistency of classes [34]. Through incremental design, test-driven development could encourage developers to create classes which have more distinct roles.

Relatively few test-driven development studies have reported coupling or cohesion metrics: coupling measures, for instance, are not entirely straightforward to analyze as classes can be naturally interconnected in an object-oriented design pattern [20]. In an industrial case study, Janzen et al. noted that coupling was actually greater in the project which utilized test-driven development [20] and the experiences were similar in another student study [26]. As for cohesion, classes constructed by test-driven developers can have fine cohesion figures but the effect is not reported to be constant across all cases and at times cohesion has been lower when test-driven development has been used [20].

4.5 Size

Besides examining the structure of code products and the relationships of objects, it is possible to determine the size of these elements. The amount of source code lines is one size measure which can be used to characterize code products. Test-driven development involves writing a considerable amount of automated unit tests that contribute to the overall size of the code base but the incremental design and implementation could have an effect on the size of the classes and methods written.

The ratio between test source code lines and non-test source code lines is one way to look at the relative size of test code. The studies from the industry show that the ratios can be substantial in projects where test-driven development is used. At Microsoft, the highest ratings were reported to be at 0.89 test code lines/production code lines, lowest at 0.39 test code lines/production code lines for a larger project and somewhere in between for other projects depending on the size of the project [18]. The numbers from IBM fall within this range at around 0.48 test code lines/production code lines [17]. Without test-driven development and with proper test-last principles, it is possible to reach fair ratios but the ratios tend to fall behind test-driven projects [14]. For student developers, the ratios have been observed to be on the same high level as industry developers [27] or somewhat lower, albeit in one case students were able to achieve a ratio of over 1 test code lines/production code lines without test-driven development [29]. It seems to be apparent that with test-driven development, the size of the overall code base increases due to the tests written if for every two lines of code there is at least a line of test code.

The size of classes and methods has in certain cases been affected under test-driven development. In the longitudinal industry study of Janzen [20], classes and methods were reported to be smaller although the same didn't apply to several other case studies mentioned in the report. Similarly, students wrote lighter classes and methods in one case [20] but not in another [26]. Madeyski and Szała found in a quasi-controlled experiment that there was less code per user story when the developer was designing and implementing the stories with test-driven development [21]. Müller and Höfer conclude from an experiment that test-driven developers with less experience don't necessarily create test code which has a larger code footprint but there might be some differences in the size of the non-test code in favor of the experts [22].

4.6 Effort

Effort is a process quality attribute [31] and here, it can be defined as the amount of exertion spent on a specific software engineering task. Typically, there is a relation between effort and duration and with careful consideration of the context the duration of a task could be seen as an indicator for effort spent. Test-driven development changes the design and implementation process of code products so the effort of these processes might be affected. The effects might not be limited

to these areas as the availability of automated tests is related to verification and validation activities as well.

Writing the tests seems to take time or then there are other factors which affect the development process as there have been experiences which suggest that the usage of test-driven development increases the effort. At Microsoft and IBM, managers estimated that development took about 15 to 30 percent longer [18]. Effort also increased in the study of Dogša and Batič [14] where the development took around three to four thousand man-hours more in an approximately six-month project; the developers felt the increase was at least partly due to the test-driven practices. George and Williams [15] and Canfora et al. [13] came to the same conclusion in their experiments that developers used more time with test-driven development. Effort and time spent on testing has also been shown to increase in academical experiments [25]. However, the correlation between test-driven development and effort isn't that straightforward as Geras et al. [16] didn't notice such a big difference and students have been shown to get a faster start into development with test-driven development [24].

Considering effort, it is not enough to look at the initial stages of development, as more effort is put on the development of the code products in later stages of the software life cycle when the code is being maintained or refactored for other purposes. Here, the industrial case study of Dogša and Batič [14] provides interesting insights into the different stages of development. Even though the test-driven development team had used more time in the development phase before the major release of the product, maintenance work on the code that was previously written with test-driven development was substantially easier and less time-consuming. The observation period for the maintenance period was around nine months, and during this time the test-driven team was quickly making up for the increased effort in the initial development stage, although they were still several thousand man-hours behind the aggregated effort of the non-test-driven teams when the observation ended.

4.7 External Quality

External quality in this review refers to qualitative, subjective and environment specific information from the empirical studies that doesn't easily convert to quantifiable metrics. For instance, mixed-method research approaches have been applied in several studies which have utilized various surveys and interviews to gather subjective opinions about test-driven development.

When customers are shown products which have been developed with test-driven development, they don't necessarily perceive a shift in quality. Huang and Holcombe [25] interviewed customers of a student project with a detailed questionnaire and asked how they felt about the quality of the product after having used it for a month. According to the results, the development method didn't affect the quality that the customers saw: test-driven products got similar ratings as the products of other development methods.

Perhaps developers who participate in the design and implementation of software can better explain what the true effects of test-driven development are?

Dogša and Batič [14] investigated this question with a survey for the industry developers who took part in the study and they felt that they were able to provide better code quality through test-driven development. The students in the study of Gupta and Jalote [24] had the sense that test-driven development improved the testability of their creation but at the same time reduced the confidence in the design of the system.

4.8 Productivity

Productivity is an indirect resource metric based on the relation between the amount of some tangible items produced and the effort required to produce the output. The resources can, for instance, be developers while the output can be products resulting from development work like source code or implemented user stories. Productivity is an external attribute which is sensitive to the environment [31]. Test-driven development seemed to be a factor in the increased effort of the developers as previously described but could the restructured development process and the tests written somehow accelerate the implementation velocity of the user stories or affect the rate by which developers write source code lines?

There have been a number of studies which have featured test-driven development and productivity. Dogša and Batič [14] reported that the industrial test-driven developer team produced code at a slightly lower rate than the other teams involved in the study and the developers also thought themselves that their productivity was affected by the practices required in test-driven development. In the experiment of Madeyski and Szała [21], there were some signs of increased productivity but it was noted that there were certain validity threats for this single-developer study. Student developers who used test-driven development in the study of Gupta and Jalote [24] were on average on the same level or a bit faster in producing source code lines than student teams developing without test-driven development. In the study of Huang and Holcombe [25], students were also faster with test-driven development, although the difference didn't exceed statistical significance. But then again there have been test-driven student teams whose productivity has been lower in terms of implemented features or source code lines [29]. In some cases, no differences between the productivity of student teams have been found [28].

The time it takes to produce one line of code might depend on the type of code being written as well. Müller and Höfer [22] examined the productivity rates when experts and students were developing code in a test-driven development experiment and noticed that both experts and students wrote test code faster than they wrote the implementation code. Experts were generally faster in writing code than students but both groups of developers wrote test code three times faster reaching maximum rates of 150 lines of code per hour. Test-driven development involves writing a lot of test code but based on this result, writing an equal amount of test code doesn't take as long as writing implementation code which is something to consider.

4.9 Maintainability

Maintainability is a property that is related to some of the evolution aspects of software: the easiness of finding out what to change from existing code, the relative effort to make a change and the sustained confidence that everything works well after the change with sufficient mechanisms to verify the effects [35]. An array of automated tests might at least help to increase the testability and stability of software which implies that test-driven has a chance to affect maintainability.

Few empirical studies about test-driven development mention maintainability and there seems to be room for additional research in this area. The industrial case study of Dogša and Batič [14] considers maintainability and the nine months maintenance period seems long enough to draw some initial conclusions. As previously described, serving the change requests for the code that had been developed with test-driven development took less time and was thus more effortless. In addition, when interviewed, developers in the study answered to a closed question that the development practice helped them to make the software more maintainable. While more research could verify whether the effect is of a constant nature, the idea is still encouraging.

4.10 Aggregation of Results

A summary of the reported effects and the frequency of individual quality attributes in research reports is shown in Figure 1. In the figure, the attributes are ordered by the number of publications showing significant positive effects associated to a particular attribute. Besides showing the positive effects, the figure illustrates the number of publications showing neutral or inconclusive effects and significant negative effects as reported in the respective publications. It should be considered that this is a coarse-grained aggregation of the study results that ignores the different contexts. A summary tailored for a specific context might look different. However, the study gives an overview of the trends and can be seen as a starting point for deeper analysis and interpretation.

Many of the significant positive effects are associated with defects but there are some positive effects related to external quality, complexity, maintainability and size. The negative effects are mostly related to effort and productivity although in one study test-driven development was seen to reduce effort. There are also several studies showing no effect with respect to effort. In case of effort and productivity, this might indicate that there are hidden context factors that have an influence on the effect of test-driven development. Code coverage is a common attribute that is mentioned in the studies but it is rarely highlighted as a key dependent variable in the studies or the results have been found inconclusive; the same can be said for size.

4.11 Limitations

The results of the individual studies have a limited scope of validity and cannot be easily generalized and compared. Therefore, the findings presented in this

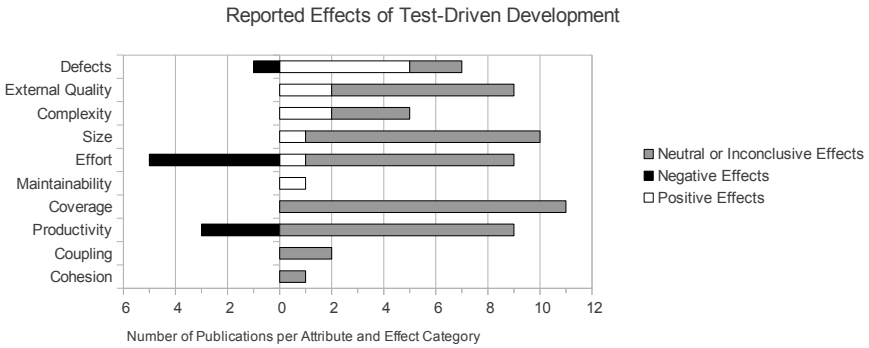

Fig. 1. The occurrence of positive, neutral and negative effects for each quality attribute as reported by the test-driven development publications included in the review

article need a careful analysis of the respective contexts before applying in other environments. The completeness of the integrative literature review was based on the ranking algorithm of the search engines and might have been enforced more strictly. Other threats to validity concern the use of qualitative inclusion and exclusion criteria as well as the selection of databases, search terms, and the chosen timeframe. Due to these factors, there could be a selection bias related to the selection of the publications. This needs to be taken into care when interpreting and using the results of this integrative literature review.

5 Conclusion

This integrative literature review analyzed the effects of test-driven development from existing empirical studies. The detailed review collected empirical findings for different quality attributes and found out varying effects to these attributes. Based on the results, prominent effects include the reduction of defects and the increased maintainability of code. The internal quality of code in terms of coupling and cohesion seem not to be affected so much but code complexity might be reduced a little with test-driven development. With all the tests written, the whole code base becomes larger but more source code lines are being covered by tests. Test code is faster to write than the code implementing the test but many of the studies report increased effort in development.

The quality map constructed as part of the review shows some possible directions for future research. One of the promising effects was the increased maintainability and reduced effort it took to maintain code later but at the time of the review there was only a single study from Dogša and Batič [14] which had specifically focused on maintainability. This could be one of the areas for further research on test-driven development.

References

1. Beck, K.: Test-Driven Development: By Example. Addison-Wesley (2003)
2. Turhan, B., Layman, L., Diep, M., Erdogmus, H., Shull, F.: How Effective is Test-Driven Development. In Oram, A., Wilson, G., eds.: Making Software: What Really Works, and Why We Believe It. O'Reilly (2010) 207–219
3. Jeffries, R., Melnik, G.: Guest Editors' Introduction: TDD–The Art of Fearless Programming. IEEE Software **24**(3) (May-June 2007) 24–30
4. Rafique, Y., Misic, V.: The Effects of Test-Driven Development on External Quality and Productivity: A Meta Analysis. IEEE Transactions on Software Engineering **39**(6) (2013) 835–856
5. Desai, C., Janzen, D., Savage, K.: A Survey of Evidence for Test-Driven Development in Academia. SIGCSE Bulletin **40**(2) (June 2008) 97–101
6. Runeson, P., Höst, M.: Guidelines for Conducting and Reporting Case Study Research in Software Engineering. Empirical Software Engineering **14** (2009) 131–164
7. Merriam, S.B.: Qualitative Research: A Guide to Design and Implementation. John Wiley & Sons (2009)
8. Torraco, R.J.: Writing Integrative Literature Reviews: Guidelines and Examples. Human Resource Development Review **4**(3) (2005) 356–367
9. Kitchenham, B., Charters, S.: Guidelines for Performing Systematic Literature Reviews in Software Engineering. Technical Report Version 2.3, Keele University and University of Durham (July 2007)
10. Creswell, J.W.: Research Design: Qualitative, Quantitative, and Mixed Methods Approaches. Sage Publications (2009)
11. Mäkinen, S.: Driving Software Quality and Structuring Work Through Test-Driven Development. Master's thesis, Department of Computer Science, University of Helsinki (October 2012)
12. Bhat, T., Nagappan, N.: Evaluating the Efficacy of Test-driven Development: Industrial Case Studies. In: Proceedings of the 2006 ACM/IEEE International Symposium on Empirical Software Engineering. ISESE '06, New York, NY, USA, ACM (2006) 356–363
13. Canfora, G., Cimitile, A., Garcia, F., Piattini, M., Visaggio, C.A.: Evaluating Advantages of Test Driven Development: A Controlled Experiment with Professionals. In: Proceedings of the 2006 ACM/IEEE International Symposium on Empirical Software Engineering. ISESE '06, New York, NY, USA, ACM (2006) 364–371
14. Dogša, T., Batič, D.: The Effectiveness of Test-Driven Development: An Industrial Case Study. Software Quality Journal **19** (2011) 643–661
15. George, B., Williams, L.: An Initial Investigation of Test Driven Development in Industry. In: Proceedings of the 2003 ACM Symposium on Applied Computing. SAC '03, New York, NY, USA, ACM (2003) 1135–1139
16. Geras, A., Smith, M., Miller, J.: A Prototype Empirical Evaluation of Test Driven Development. In: Proceedings of the 10th International Symposium on Software Metrics. METRICS 2004 (September 2004) 405–416
17. Maximilien, E., Williams, L.: Assessing Test-Driven Development at IBM. In: Proceedings of the 25th International Conference on Software Engineering. ICSE 2003 (May 2003) 564–569
18. Nagappan, N., Maximilien, E., Bhat, T., Williams, L.: Realizing Quality Improvement through Test Driven Development: Results and Experiences of Four Industrial Teams. Empirical Software Engineering **13** (2008) 289–302

19. Williams, L., Maximilien, E., Vouk, M.: Test-Driven Development as a Defect-Reduction Practice. In: Proceedings of the 14th International Symposium on Software Reliability Engineering. ISSRE 2003 (November 2003) 34–45
20. Janzen, D.S., Saiedian, H.: Does Test-Driven Development Really Improve Software Design Quality? IEEE Software **25**(2) (March-April 2008) 77–84
21. Madeyski, L., Szała, L.: The Impact of Test-Driven Development on Software Development Productivity — An Empirical Study. In Abrahamsson, P., Baddoo, N., Margaria, T., Messnarz, R., eds.: Software Process Improvement. Volume 4764 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg (2007) 200–211
22. Müller, M., Höfer, A.: The Effect of Experience on the Test-Driven Development Process. Empirical Software Engineering **12**(6) (2007) 593–615
23. Desai, C., Janzen, D.S., Clements, J.: Implications of Integrating Test-Driven Development Into CS1/CS2 Curricula. In: Proceedings of the 40th ACM Technical Symposium on Computer Science Education. SIGCSE '09, New York, NY, USA, ACM (2009) 148–152
24. Gupta, A., Jalote, P.: An Experimental Evaluation of the Effectiveness and Efficiency of the Test Driven Development. In: Proceedings of the First International Symposium on Empirical Software Engineering and Measurement. ESEM 2007 (September 2007) 285–294
25. Huang, L., Holcombe, M.: Empirical Investigation Towards the Effectiveness of Test First Programming. Information and Software Technology **51**(1) (2009) 182–194
26. Janzen, D., Saiedian, H.: On the Influence of Test-Driven Development on Software Design. In: Proceedings of the 19th Conference on Software Engineering Education and Training. CSEET '06 (April 2006) 141–148
27. Madeyski, L.: The Impact of Test-First Programming on Branch Coverage and Mutation Score Indicator of Unit Tests: An Experiment. Information and Software Technology **52**(2) (2010) 169–184
28. Pančur, M., Ciglarič, M.: Impact of Test-Driven Development on Productivity, Code and Tests: A Controlled Experiment. Information and Software Technology **53**(6) (2011) 557–573
29. Vu, J., Frojd, N., Shenkel-Therolf, C., Janzen, D.: Evaluating Test-Driven Development in an Industry-Sponsored Capstone Project. In: Proceedings of the Sixth International Conference on Information Technology: New Generations. ITNG '09 (April 2009) 229–234
30. Wilkerson, J., Nunamaker, J.J., Mercer, R.: Comparing the Defect Reduction Benefits of Code Inspection and Test-Driven Development. IEEE Transactions on Software Engineering **38**(3) (May-June 2012) 547–560
31. Fenton, N.E., Pfleeger, S.L.: Software Metrics: A Rigorous and Practical Approach. PWS Publishing Company, Boston (MA) (1997)
32. Pezzè, M., Young, M.: Software Testing and Analysis: Process, Principles and Techniques. Wiley, Chichester (2008)
33. McCabe, T.: A Complexity Measure. IEEE Transactions on Software Engineering **SE-2**(4) (December 1976) 308–320
34. Chidamber, S., Kemerer, C.: A Metrics Suite for Object Oriented Design. IEEE Transactions on Software Engineering **20**(6) (June 1994) 476–493
35. Cook, S., He, J., Harrison, R.: Dynamic and Static Views of Software Evolution. In: Proceedings of the IEEE International Conference on Software Maintenance. (2001) 592–601