# How Effective is Test Driven Development

**Chapter** · October 2010

CITATIONS
26

READS
1,566

**5 authors**, including:

Burak Turhan
Monash University (Australia)
**152** PUBLICATIONS **1,938** CITATIONS
SEE PROFILE

Lucas Layman
Fraunhofer Center for Experimental Software Engineering
**50** PUBLICATIONS **1,115** CITATIONS
SEE PROFILE

Madeline Diep
Fraunhofer Center for Experimental Software Engineering
**20** PUBLICATIONS **214** CITATIONS
SEE PROFILE

Hakan Erdogmus
Carnegie Mellon University
**135** PUBLICATIONS **1,840** CITATIONS
SEE PROFILE

CHAPTER TWELVE

# How Effective is Test-Driven Development?

*Burak Turhan*
*Lucas Layman*
*Madeline Diep*
*Hakan Erdogmus*
*Forrest Shull*

Test-Driven Development (TDD) [Beck 2002] is one of the most referenced, yet least used agile practices in industry. Its neglect is due mostly to our lack of understanding of its effects on people, processes, and products. Although most people agree that writing a test case before code promotes more robust implementation and a better design, the unknown costs associated with TDD's effects and the inversion of the ubiquitous programmer "code-then-test" paradigm has impeded TDD's adoption.

To provide an overview of current evidence on the effects of TDD, we conducted a systematic review of TDD research in online databases and scientific publications. Systematic review is a research method popularized in the medical community for aggregating and analyzing the results of clinical trials. A systematic review seeks to answer the general question, "*What does the published evidence say about the effects of using technique X*?" In medicine, systematic reviews play a critical role in evaluating the effectiveness of pharmaceutical drugs and alternative treatments for illnesses. Empirical software engineering researchers have adopted this approach for summarizing and analyzing evidence about the effects of software development

practices. It is covered in Chapter 3, *What We Can Learn From Systematic Reviews*, by Barbara Kitchenham, and in [Kitchenham 2004] and [Dybå et al. 2005].

In this chapter, we treat TDD as an imaginary medical pill and describe its effects with a narrative from a pharmacological point of view, instead of providing a formal systematic review report. We invite the reader to imagine that the rest of this chapter is a medical fact sheet for the TDD "pill" and to continue reading with the following question in mind:

> "If TDD were a pill, would you take it to improve your health?"

## The TDD Pill—What is It?

The ingredients of the TDD pill are as follows and should be prepared following the given order *exactly*:

1. Choose a small task.
2. Write a test for that task.
3. Run all the tests to verify that the new test fails.
4. Write minimal production code to complete the task.
5. Run all tests (including the new one) to verify that they pass.
6. Refactor the code as necessary.
7. Repeat from step 1.

The active ingredient in the TDD pill is the authoring of test cases before production code. Authoring test cases before code requires the patient to consider the design of the solution, how information will flow, the possible outputs of the code, and exceptional scenarios that might occur. Running the newly written test case before writing production code helps to verify that the test is written correctly (a passing test case at this point is not testing the intended effects) and that the system compiles. The TDD pill also involves writing just enough production code to pass the test case, which encourages an uncluttered, modular design. Furthermore, TDD users create a growing library of automated test cases that can be executed at any time to verify the correctness of the existing system whenever changes are made.

Like many drugs, the TDD pill has some official variants, including ATDD (Acceptance-Test-Driven Development), BDD (Behavior-Driven Development) and STDD (Story-Test-Driven Development). ATDD replaces the "small task" step with "functional level business logic tasks," whereas BDD uses "behavioral specifications" instead. The ordering of tasks in TDD differentiates it from other treatments, but official varieties of TDD pills may also contain sets of other ingredients, such as breaking work down into simple tasks, refactoring, keeping the test-code cycles short, and relentless regression testing.

WARNING

**Except for refactoring, some key ingredients may be missing in many "generic" TDD pills due to different interpretations in practice and the dominance of the active ingredient.**

## Summary of Clinical TDD Trials

The focus of our review was to gather *quantitative* evidence on the effects of the TDD pill on internal code quality (see "Measuring Code Quality" on page 209), external quality, productivity, and test quality. The evaluation of the TDD pill is based on data gathered from 32 clinical trials. In the first quarter of 2009, the authors gathered 325 TDD research reports from online comprehensive indices, major scientific publishers (ACM, IEEE, Elsevier), and "gray literature" (technical reports, theses). The initial set of 325 reports was narrowed down to 22 reports through a two-level screening process. Four researchers filtered out studies conducted prior to 2000, qualitative studies, surveys, and wholly subjective analyses of the TDD pill. Some of these reports contained multiple or overlapping trials (i.e., the same trial was reported in multiple papers); in such cases, the trial was counted only once. A team of five researchers then extracted key information from the reports regarding study design, study context, participants, treatments and controls, and study results. In total, the research team analyzed 22 reports containing 32 unique trials.

### MEASURING CODE QUALITY

The *internal quality* of a system is related to its design quality, usually with the interpretation that good designs are simple, modular, and easy to maintain and understand. Though TDD is primarily interpreted as a development practice, it is considered as a design practice as well. An incremental and simple design is expected to emerge when using the TDD pill. The simple design is driven by the modularity needed to make code testable, by writing minimal production code to complete simple tasks, and by constant refactoring. To assess the internal quality of a system, the TDD trials use one or more of the following measures:

- Object-oriented metrics. These involve weighted methods on a per-class basis (WMC, depth of inheritance tree), DIT, etc. [Chidamber et al. 1994]
- Cyclomatic complexity
- Code density (e.g., lines of code per method)
- Code size per feature

The *external quality* of a system is usually measured by the number of pre-release or post-release defects. TDD is associated with the important claim that it increases external quality because it encourages writing lots of test cases, developers work on simple tasks that are easy to comprehend, the system is under frequent regression testing, and errors due to changes can be easily detected by the fine-grained tests. In TDD trials, external quality is reported by one or more of the following:

- Test cases passed
- Number of defects
- Defect density
- Defects per test
- Effort required to fix defects
- Change density
- Percentage of preventative changes

---

The 32 trials were conducted in academic or industrial settings in the form of controlled experiments, pilot studies, or commercial projects. Controlled experiments were conducted in academic laboratories or controlled industry environments with a defined research protocol, pilot studies were carried out using less-structured experimental tasks, and commercial projects described industry teams using TDD as part of their everyday work. Participants in these trials had different experience levels, ranging from undergraduate students to graduate students to professionals. The number of participants per trial ranged from a single individual to 132 persons. The effort spent on the trials spans a wide interval, ranging from a few person-hours to 21,600 person-hours. Each trial compares the effects of the TDD pill with respect to another treatment—usually traditional test-last development. The subjects in the treatment groups were comprised of various units, such as individuals, pairs, teams, and projects.

We have classified the 32 trials into four levels based on the experience of participants, the detail of the experimental construct, and the scale of the trial. The experience of participants is determined by whether they were undergraduate students, graduate students, or professionals.

The descriptions of the dynamics of TDD and the control treatment were used to evaluate the construct of the trial as either good, adequate, poor, or unknown. A *good* construct enforced all prescribed TDD ingredients from the prior section, an *adequate* construct prescribed writing tests first but not all of the TDD ingredients, a *poor* construct did not enforce the TDD steps, and an *unknown* construct did not specify whether the TDD steps were enforced or not.

Finally, the scale of a trial is recorded as small, medium or large, depending on the reported effort or estimated effort based on duration and number of participants. Small projects involved less than 170 person-hours of total effort across all subjects, whereas large projects ranged from 3,000 to 21,600 person-hours. A simple clustering algorithm was used to categorize the scale

of the projects, while the experience of participants and the details of the construct were based on descriptive data found in the trial reports.

Our confidence that the results of using the TDD pill will generalize to "real-life" cases increases as the level of the trial increases. The lowest level, L0, contains all small-scale trials. These trials report less than 170 person-hours of effort or less than 11 participants. The next level, L1, consists of medium- or large-scale trials with unknown or poor constructs. The L2 level consists of medium- or large-scale trials with adequate or good constructs and undergraduate student participants. Finally, the highest level, L3, contains medium- or large-scale trials with adequate or good constructs and graduate student or professional participants.

Table 12-1 summarizes the attributes we used to classify trials into levels, and Table 12-2 shows how many trials we examined at each level.

*TABLE 12-1. Levels of clinical TDD trials*

| | L0 | L1 | L2 | L3 |
|---|---|---|---|---|
| Experience | Any | Any | Undergraduate student | Graduate student or professional |
| Construct | Any | Poor or unknown | Adequate or good | Adequate or good |
| Scale | Small | Medium or large | Medium or large | Medium or large |

*TABLE 12-2. Types of clinical TDD trials*

| Type | L0 | L1 | L2 | L3 | Total |
|---|---|---|---|---|---|
| Controlled experiment | 2 | 0 | 2 | 4 | 8 |
| Pilot study | 2 | 0 | 5 | 7 | 14 |
| Industrial use | 1 | 7 | 0 | 2 | 10 |
| Total | 5 | 7 | 7 | 13 | 32 |

## The Effectiveness of TDD

We analyzed the TDD trials that reported quantitative results of the TDD pill's effects on productivity, internal and external quality, and test quality. Direct comparison of the quantitative results across trials was impossible, since the trials measured TDD's effectiveness in different ways. Instead, we assign each trial a summary value of "better," "worse," "mixed," or "inconclusive/no-difference." The summary value is determined by the quantitative results reported for the TDD pill compared with a control. The summary value also incorporates the report author's interpretation of the trial results. In trials with a summary value of "better," a majority of quantitative measures favor the TDD pill in comparison to the control treatment. In trials with a summary value of "worse," a majority of measures favor the control treatment. Trials with a summary value of "inconclusive/no-difference" were inconclusive or report no observed differences. Finally, in trials with a summary value of "mixed," some measures favor TDD while others don't. In all cases, the summary assignation was guided by the report author's

interpretation of the study findings because, in many cases, the reports omitted details of the trial that would have enabled an objective external evaluation.

In the following sections we do our best to draw some conclusions about the value of TDD from the trials.

## Internal Quality

Available evidence from the trials suggests that TDD does not have a consistent effect on internal quality. Although TDD appears to yield better results over the control group for certain types of metrics (complexity and reuse), other metrics (coupling and cohesion) are often worse in the TDD treatment. Another observation from the trial data is that TDD yields production code that is less complex at the method/class level, but more complex at the package/project level. This inconsistent effect is more visible in more rigorous trials (i.e., L2 and L3 trials). The differences in internal quality may be due to other factors, such as motivation, skill, experience, and learning effects. Table 12-3 classifies the trials according to internal quality metrics.

> NOTE
> **In the following tables, the first number in each cell reports all trials, whereas the number in parentheses reports only L2 and L3 trials.**

*TABLE 12-3. Effects on internal quality*

| Type | BETTER | WORSE | MIXED | INC \| NO-DIFF | Total |
|---|---|---|---|---|---|
| Controlled experiment | 1 (0) | 0 (0) | 0 (0) | 2 (2) | 3 (2) |
| Pilot study | 1 (1) | 1 (1) | 3 (1) | 2 (2) | 7 (5) |
| Industrial use | 3 (1) | 1 (1) | 0 (0) | 0 (0) | 4 (2) |
| Total | 5 (2) | 2 (2) | 3 (1) | 4 (4) | 14 (9) |

## External Quality

There is some evidence to suggest that TDD improves external quality. Although the outcomes of controlled experiments are mostly inconclusive, industrial use and pilot studies strongly favor TDD. However, the supporting evidence from industrial use and controlled experiments disappears after filtering out the less rigorous studies (i.e., L0 and L1 trials). Furthermore, the evidence from pilot studies and controlled experiments is contradictory once L0 and L1 trials are filtered out. If all studies are counted equally, however, the evidence suggests that the TDD pill can improve external quality. Table 12-4 classifies the trials according to external quality metrics.

*TABLE 12-4. Effects on external quality*

| Type | BETTER | WORSE | MIXED | INC \| NO-DIFF | Total |
|---|---|---|---|---|---|
| Controlled experiment | 1 (0) | 2 (2) | 0 (0) | 3 (3) | 6 (5) |
| Pilot study | 6 (5) | 1 (1) | 0 (0) | 2 (2) | 9 (8) |
| Industrial use | 6 (0) | 0 (0) | 0 (0) | 1 (1) | 7 (1) |
| Total | 13 (5) | 3 (3) | 0 (0) | 6 (6) | 22 (14) |

## Productivity

The productivity dimension engenders the most controversial discussion of TDD. Although many admit that adopting TDD may require a steep learning curve that may decrease the productivity initially, there is no consensus on the long-term effects. One line of argument expects productivity to increase with TDD; reasons include easy context switching from one simple task to another, improved external quality (i.e., there are few errors and errors can be detected quickly), improved internal quality (i.e., fixing errors is easier due to simpler design), and improved test quality (i.e., chances of introducing new errors is low due to automated tests). The opposite line argues that TDD incurs too much overhead and will negatively impact productivity because too much time and focus may be spent on authoring tests as opposed to adding new functionality. The different measures used in TDD trials for evaluating productivity included development and maintenance effort, the amount of code or features produced over time, and the amount of code or features produced per unit of development effort.

The available evidence from the trials suggests that TDD does not have a consistent effect on productivity. The evidence from controlled experiments suggests an improvement in productivity when TDD is used. However, the pilot studies provide mixed evidence, some in favor of and others against TDD. In the industrial studies, the evidence suggests that TDD yields worse productivity. Even when considering only the more rigorous studies (L2 and L3), the evidence is equally split for and against a positive effect on productivity. Table 12-5 classifies the trials according to effects on productivity.

*TABLE 12-5. Effects on productivity*

| Type | BETTER | WORSE | MIXED | INC \| NO-DIFF | Total |
|---|---|---|---|---|---|
| Controlled experiment | 3 (1) | 0 (0) | 0 (0) | 1 (1) | 4 (2) |
| Pilot study | 6 (5) | 4 (4) | 0 (0) | 4 (3) | 14 (12) |
| Industrial use | 1 (0) | 5 (1) | 0 (0) | 1 (0) | 7 (1) |
| Total | 10 (6) | 9 (5) | 0 (0) | 6 (4) | 25 (15) |

### Test Quality

Because test cases precede all development activities with TDD, testing the correctness of an evolving system is expected to be made easier by a growing library of automated tests. Further, the testing process is expected to be of high quality due to the fine granularity of the tests produced. In the trials, test quality is captured by test density, test coverage, test productivity, or test effort.

There is some evidence to suggest that TDD improves test quality. Most of the evidence comes from pilot studies and is in favor of TDD, even after filtering out less rigorous studies. Controlled experiments suggest that TDD fares at least as well as the control treatments. There is insufficient evidence from industrial use to reach a conclusion.

Therefore, the test quality associated with TDD seems at least not worse and often better than alternative approaches. Here we would have expected stronger results: since encouraging test case development is one of the primary active ingredients of TDD, the overall evidence should have favored TDD in promoting the test quality measures reported in these studies.

Table 12-6 classifies the trials according to test quality.

*TABLE 12-6. Effects on test quality*

| Type | BETTER | WORSE | MIXED | INC \| NO-DIFF | Total |
|---|---|---|---|---|---|
| Controlled experiment | 2 (1) | 0 (0) | 0 (0) | 3 (3) | 5 (4) |
| Pilot study | 7 (5) | 1 (1) | 0 (0) | 1 (1) | 9 (7) |
| Industrial use | 1 (0) | 1 (1) | 0 (0) | 1 (0) | 3 (1) |
| Total | 10 (6) | 2 (2) | 0 (0) | 5 (4) | 17 (12) |

## Enforcing Correct TDD Dosage in Trials

Although most of the trials did not measure or control the amount of the TDD pill taken (which in software parlance translates into a lack of attention to process conformance), we believe that the dosage ended up being variable across trials and subjects. Trials with poor or unknown constructs may not have strictly enforced TDD usage, and we believe it is highly likely that the trial participants customized the pill with a selection of ingredients rather than following the strict textbook definition of TDD. This issue poses a serious threat to drawing generalized conclusions. In the medical context, not enforcing or measuring TDD usage is analogous to failing to ensure that the patients took a pill for some treatment or not knowing which dosage the patient took. Thus, the observed effects of the TDD pill may be due to process conformance or other factors that are not adequately described or controlled. In future trials, conformance to the treatment and the control should be carefully monitored.

Regardless of the reporting quality of the TDD trials, a related question is raised: "Should the textbook definition of TDD be followed in all real-life cases?" Sometimes patients get better even with a half-sized or quarter-sized pill modified for their specific work context and personal

style. Micro-level logging tools for development activity are available and can be used to investigate these issues. Such logging tools can be helpful both for controlling the conformance to TDD processes and for understanding real-life, practical implementations of TDD.

## Cautions and Side Effects

In this section we pose several questions about the TDD pill that may temper TDD's effectiveness in different contexts.

*Is it reactive to the environment?*

There is no recommended best context for the use of TDD. We do not know whether it is applicable to all domains, to all kinds of tasks within a domain, or to projects of all sizes and complexities. For example, the trials do not make it clear whether TDD is an applicable practice for developing embedded systems or for developing highly decentralized systems where incremental testing may not be feasible. Furthermore, it is often considered a challenge to use TDD for legacy systems that may require considerable refactoring of existing code to become testable.

*Is it for everyone?*

One basic fact on which almost everyone agrees is that TDD is difficult to learn. It involves a steep learning curve that requires skill, maturity, and time, particularly when developers are entrenched in the code-then-test paradigm. Better tool support for test-case generation and early exposure in the classroom to a test-then-code mentality may encourage TDD adoption.

*Could it be addictive?*

Personal communications with TDD developers suggest that it is an addictive practice. It changes the way people think and their approach to coding in a way that is difficult to roll back. Therefore, leaving TDD practices may be as difficult as adopting them.

*Does it interact with other medications?*

No studies focus specifically on whether TDD performs better or worse when used with other medications. In one trial, it is suggested that, when coupled with up-front design, TDD results in a 40% improvement in external quality [Williams et al. 2003]. Another trial compares solo and pair developers who practice TDD and incremental test-last development [Madeyski 2005]. That trial reports no difference in the external quality of software produced by solo or pair programmers using TDD. It is not known which practices go well or poorly with TDD. Although there may be practices that stimulate its desired effects, there also may be some that inhibit them. The examples just mentioned are probably case-specific, but they point out the need to investigate further TDD's interaction with other medications.

## Conclusions

The effects of TDD still involve many unknowns. Indeed, the evidence is not undisputedly consistent regarding TDD's effects on any of the measures we applied: internal and external quality, productivity, or test quality. Much of the inconsistency likely can be attributed to internal factors not fully described in the TDD trials. Thus, TDD is bound to remain a controversial topic of debate and research.

For practitioners looking for some actionable advice, our expert panel recommends taking the TDD pill, carefully monitoring its interactions and side effects, and increasing or decreasing the dosage accordingly. So we end with some specific prescriptions from individual members of our team, after reviewing the data:

> We've been able to compile the evidence, but each reader has to make up his or her own mind. First, decide which qualities matter most to you. For example, do you care more about productivity or external quality? Can you justify spending more effort to create higher-quality tests? The evidence in this chapter is useful only for making decisions based on each reader's specific goals.
>
> I have taken the TDD pill and become hooked. My personal experience has been that TDD improves productivity, although evidence from our study is lacking in this regard. Perhaps mine was simply a perception. Based on these results, especially based on the evidence regarding its conservatively positive impact on external quality, if I weren't already using TDD, I'd start having my team take it in small doses and see whether they find a long-term productivity improvement of their own. If there are no adverse reactions, I'd increase the dosage gradually and keep observing.
>
> Although TDD is promising, its adoption can be impeded by uncertainties about its effectiveness and by high up-front adoption cost. Still, its ingredients seem to encourage good programming and development habits to flourish, yielding better-quality programmers and tests in the long run.
>
> TDD seems promising, but let's face it, it tastes bad when you first start. A lot of people like the old stuff better. After all, it's hard to feel productive when you spend a large amount of your time writing test cases that fail. On the other hand, I've never written cleaner code in my life, and it feels great to make a change to that old code, hit the "Run Tests" button, and be confident that I didn't break anything.
>
> The evidence packed into this chapter shows that TDD might be a cure for you, yet you should not try to use it as a panacea. Your TDD adventure is likely to vary with certain factors, including your experience and the context you are working in. As a practitioner, developing an insight about when to expect improvements from TDD would be a valuable asset.

## Acknowledgments

Dr. Janice Singer was one of the researchers who participated in the screening of the studies in the initial stages of the systematic review. We gratefully acknowledge her contributions to this work.

## General References

[Beck 2002] Beck, Kent. 2002. *Test-Driven Development: By Example*. Boston: Addison-Wesley.

[Chidamber et al. 1994] Chidamber, S.R., and C.F. Kemerer. 1994. A Metrics Suite for Object Oriented Design. *IEEE Transactions on Software Engineering*20(6): 476-493.

[Dybå et al. 2005] Dybå, Tore, Barbara Kitchenham, and Magne Jørgensen. 2005. Evidence-Based Software Engineering for Practitioners. *IEEE Software* 22(1): 58–65.

[Kitchenham 2004] Kitchenham, Barbara. 2004. Procedures for Performing Systematic Reviews. Keele University Technical Report TR/SE0401.

## Clinical TDD Trial References

[Canfora et al. 2006] Canfora, Gerardo, Aniello Cimitile, Felix Garcia, Mario Piattini, and Corrado Aaron Visaggio. 2006. Evaluating advantages of test-driven development: a controlled experiment with professionals. *Proceedings of the ACM/IEEE international symposium on Empirical software engineering*: 364-371.

[Erdogmus et al. 2005] Erdogmus, Hakan, Maurizio Morisio, and Marco Torchiano. 2005. On the Effectiveness of the Test-First Approach to Programming. *IEEE Transactions on Software Engineering*31(3):226-237.

[Flohr et al. 2006] Flohr, Thomas, and Thorsten Schneider. 2006. Lessons Learned from an XP Experiment with Students: Test-First Needs More Teachings. In *Product-Focused Software Process Improvement: 7th International Conference, PROFES 2006, Proceedings*, ed. J. Münch and M. Vierimaa, 305–318. Berlin: Springer-Verlag.

[George 2002] George, Boby. 2002. *Analysis and Quantification of Test-Driven Development Approach*. MS thesis, North Carolina State University.

[Geras 2004] Geras, Adam. 2004. *The effectiveness of test-driven development*. MSc thesis, University of Calgary.

[Geras et al. 2004] Geras, A., M. Smith, and J. Miller. 2004. A Prototype Empirical Evaluation of Test-Driven Development. *Proceedings of the 10th International Symposium on Software Metrics*: 405-416.

[Gupta et al. 2007] Gupta, Atul, and Pankaj Jaloye. 2007. An Experimental Evaluation of the Effectiveness and Efficiency of the Test-Driven Development. *Proceedings of the First International Symposium on Empirical Software Engineering and Measurement*: 285-294.

[Huang et al. 2009] Huang, Liang, and Mike Holcombe. 2009. Empirical investigation towards the effectiveness of Test First programming. *Information & Software Technology* 51(1): 182-194.

[Janzen 2006] Janzen, David Scott. 2006. *An Empirical Evaluation of the Impact of Test-Driven Development on Software Quality*. PhD thesis, University of Kansas.

[Kaufmann et al. 2003] Kaufmann, Reid, and David Janzen. 2003. Implications of test-driven development: a pilot study. *Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*: 298-299.

[Madeyski 2005] Madeyski, Lech. 2005. Preliminary Analysis of the Effects of Pair Programming and Test-Driven Development on the External Code Quality. *Proceedings of the 2005 conference on Software Engineering: Evolution and Emerging Technologies*: 113–123.

[Madeyski 2006] Madeyski, Lech. 2006. The Impact of Pair Programming and Test-Driven Development on Package Dependencies in Object-Oriented Design—An Experiment. In *Product-Focused Software Process Improvement: 7th International Conference, PROFES 2006, Proceedings*, ed. J. Münch and M. Vierimaa, 278–289. Berlin: Springer-Verlag.

[Madeyski et al. 2007] Madeyski, Lech, and Lukasz Szala. 2007. The Impact of Test-Driven Development on Software Development Productivity — An Empirical Study. *Software Process Improvement, 4th European Conference, EuroSPI 2007, Proceedings*, ed. P. Abrahamsson, N. Baddoo, T. Margaria, and R. Massnarz, 200–211. Berlin: Springer-Verlag.

[Muller et al. 2002] Muller, M. M., and O. Hagner. 2002. Experiment about test-first programming. *Software, IEEE Proceedings*149(5): 131-136.

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

[Pancur et al. 2003] Pancur, M., M. Ciglaric, M. Trampus, and T. Vidmar. 2003. Towards empirical evaluation of test-driven development in a university environment. *The IEEE Region 8 EUROCON Computer as a Tool*(2): 83-86.

[Siniaalto et al. 2008] Siniaalto, Maria, and Pekka Abrahamsson. 2008. Does Test-Driven Development Improve the Program Code? Alarming Results from a Comparative Case Study. In *Balancing Agility and Formalism in Software Engineering*, ed. B Meyer, J. Nawrocki, and B. Walter, 143–156. Berlin: Springer-Verlag.

[Slyngstad et al. 2008] Slyngstad, Odd Petter N., Jingyue Li, Reidar Conradi, Harald Ronneberg, Einar Landre, and Harald Wesenberg. 2008. The Impact of Test Driven Development on the Evolution of a Reusable Framework of Components—An Industrial Case Study. *Proceedings of the Third International Conference on Software Engineering Advances*: 214-223.

[Vu et al. 2009] Vu, John, Niklas Frojd, Clay Shenkel-Therolf, and David Janzen. 2009. Evaluating Test-Driven Development in an Industry-sponsored Capstone Project. *Proceedings of the 2009 Sixth International Conference on Information Technology: New Generations*: 229-234.

[Williams et al. 2003] Williams, Laurie, E. Michael Maximilien, and Mladen Vouk. 2003. Test-Driven Development as a Defect-Reduction Practice. *Proceedings of the 14th International Symposium on Software Reliability Engineering*: 34.

[Yenduri et al. 2006] Yenduri, Sumanth, and Louise A. Perkins. 2006. Impact of Using Test-Driven Development: A Case Study. *Proceedings of the International Conference on Software Engineering Research and Practice & Conference on Programming Languages and Compilers, SERP 2006*: 126-129.

[Zhang et al. 2006] Zhang, Lei, Shunsuke Akifuji, Katsumi Kawai, and Tsuyoshi Morioka. 2006. Comparison Between Test-Driven Development and Waterfall Development in a Small-Scale Project. *Extreme Programming and Agile Processes in Software Engineering, 7th International Conference, XP 2006, Proceedings*, ed. P. Abrahamsson, M. Marchesi, and G. Succi, 211–212. Berlin: Springer-Verlag.