# Test-Driven Development

Lech Madeyski

# Test-Driven Development

An Empirical Evaluation of Agile Practice

Springer

Lech Madeyski
Wrocław University of Technology
Institute of Informatics (I-32)
Software Engineering Department
Wybrzeże Wyspiańskiego 27
50-370 Wrocław
Poland
lech.madeyski@pwr.wroc.pl
http://madeyski.e-informatyka.pl/

ISBN 978-3-642-04287-4 e-ISBN 978-3-642-04288-1
DOI 10.1007/978-3-642-04288-1
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2009939143

ACM Computing Classification (1998): D.2, K.6

*Cover design*: KuenkelLopka GmbH

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

*To Iwona, Lech, Natalia, Judyta and Małgorzata*

# Foreword

This book makes an important and timely contribution. Agile methods are gaining more and more interest both in industry and research. Many industries transform their way of working from traditional waterfall projects with long duration to more incremental, iterative and agile practices. Furthermore, the need to evaluate and to obtain evidence for different processes, methods and tools has been emphasized in research in particular over the last decade. The main contribution comes primarily from combining agile methods (or more precisely XP and test-first programming) with empirical evaluation.

The book's contribution is important since it evaluates empirically a way of working which is more and more embraced by companies developing software or software-intensive systems. Thus, it evaluates new way of working which is a challenge given the pace in which new technologies evolve, and in particular it provides evidence in relation to test-first programming or test-driven development. However, this is not the sole contribution. The book presents three experiments in much more depth than normally is possible in a research article. This means that it provides readers with in-depth insights into experimental methods in the context of agile development. This includes presentation of statistical methods having concrete examples of how to conduct the statistical analysis with SPSS and a thorough discussion about the different validity threats in empirical studies. Furthermore, the book presents how meta-analysis can be conducted when having several separate experiments.

In summary the book provides many valuable insights both to practitioners in terms of the evidence for test-first programming and to researchers in terms of clear illustrations of how new processes, methods and tools can be evaluated using experimentation in software engineering. It is pleasure to recommend this book to practitioners and researchers being interested in agile methods or empirical evaluation or both of them.

Karlskrona, Sweden

*Professor Claes Wohlin*
Blekinge Institute of Technology

# Preface

*There is strong shadow where there is much light.*
–Johann Wolfgang von Goethe

Following the recent recommendations in medicine and psychology [21, 98], as well as the guidelines offered by the empirical software engineering community [109, 127, 140, 141, 227], this preface takes the form of a structured abstract.[1] For the sake of clarity, statistical results are not included in the preface itself but they may be found in Appendix.

## Background

The quality of the methods used to evaluate new software development techniques, practices, processes, technologies, tools, etc., indicates the maturity of the software engineering research discipline. Consequently, experimentation, as a systematic, disciplined, quantifiable and controlled way of evaluation becomes a fundamental part of both research and practice in software engineering. The importance of properly conducted and replicated experiments has become widely accepted in the software engineering community. Owing to the empirical studies and their analysis or meta-analysis, industry may take advantage of the already accumulated knowledge. The roots of that fundamental shift in software engineering research can, to a large extent, be found in evidence-based medicine. Controlled experiments, quasi-experiments and case studies become the primary research methods by which the choice of software development techniques should be justified. Systematic reviews and meta-analyses are gaining increasing acceptance as the methods of summarizing

[1] Structured abstracts organize the summaries of publications with the following common headings: background or context, objective or aim, method, results, limitations and conclusions [127]. Several researchers who compared structured abstracts with traditional ones support the claim that structured abstracts are an improvement over traditional ones. Not only is more information presented, which is helpful for the reader, but also the format requires that the authors organize and present their findings in a systematic way, easier to read [109].

the results of a number of empirical studies. Evidence-based software engineering (EBSE) undermines anecdotal evidence and unsystematic experience as sufficient grounds for decision making while stressing instead the empirical evidence from software engineering research.

Another interesting shift in software engineering has been created by the agile movement in general, and eXtreme Programming (XP) in particular. Agile teams shape software systems using a collaborative process, with executable software and automated tests at its heart, whilst marginalising the documents. That creates a shift away from tools for managing requirements to tools (originating from the XP values, principles and practices) supporting collaboration and the gradual distillation of requirements into automated test suites [58]. The Test-First Programming (TF) practice, also called Test-Driven Development, is considered the flagship and one of the most influential practices of the XP methodology [15, 23, 144], as well as the one of the most controversial ones [179].

Both the experimentation in software engineering and the agile movement influenced this book. The latter attracted the attention of the author to the agile methodologies, XP and then the flagship XP practice, i.e. TF. The former influenced the way the research was conducted and reported (e.g. the stress on the effect size estimates and meta-analysis).

## Objective

The purpose of this book was to evaluate the effects of the TF agile software development practice with respect to the percentage of acceptance tests passed (considered an external code quality indicator [87, 88]), design complexity metrics (that have been found significant for assessing fault proneness by several researchers [19, 32, 33, 95, 200, 238]) and the number of acceptance tests passed per development hour (which is an indicator of development speed). Moreover, the aim is to present the preliminary evaluation of the impact of the TF practice on mutation score indicator and branch coverage, the indicators of the fault detection effectiveness and the thoroughness of unit tests, respectively.

An additional (but auxiliary) objective of the book was to present how to perform an analysis of experiments in software engineering using the Statistical Package for the Social Sciences (SPSS). Conference or even journal papers usually present short and thus superficial descriptions of the performed analyses, while the existing excellent books [131, 227, 259] cover a wide range of topics related to Empirical Software Engineering (ESE) and, therefore, do not focus on the joint analysis of closely related experiments.

## Method

The effects of the TF programming practice were evaluated by conducting three experiments named ACCOUNTING (experiment on the development of an accounting system), SUBMISSION (experiment on the development of a paper submission and review system), and SMELLS&LIBRARY (experiment on the development of both a tool for identifying bad smells in Java source code through the use of a set of software metrics and a library application). Those experiments, described in Chap. 4, were carried out in academic setting with over 200 graduate MSc students, using both between-groups (in Experiments ACCOUNTING and SUBMISSION) and repeated measures (in Experiment SMELLS&LIBRARY) experimental designs. Furthermore, the Pair Programming (PP) practice was used along with the TF programming practice in the first experiment to check whether there is a synergy between both XP practices. The data were collected with the help of different measurement tools. Some of them (Judy, Aopmetrics, ActivitySensor and SmartSensor Eclipse plugins) have been developed especially for the sake of the experiments. The statistical analysis of experiments has been described in Chaps. 5, 6 and 7, while the meta-analysis has been performed in Chap. 9. A selective analysis and selective meta-analysis have been carried out to minimize threats to the validity (e.g. process conformance threat). Effect sizes were reported and interpreted with respect to their practical importance.

## Results

The main result observed on the basis of the meta-analysis of Experiments ACCOUNTING, SUBMISSION and SMELLS&LIBRARY is that programmers using Test-First Solo Programming (TFSP) technique produce a code that is significantly less coupled than that produced by programmers using Test-Last Solo Programming (TLSP) technique. This finding has also been confirmed by the selective analysis of Experiments SUBMISSION and SMELLS&LIBRARY. Furthermore, the mean effect size represents a medium (but close to large) effect on the basis of meta-analysis of all the experiments, as well as selective meta-analysis, which is a substantial finding. It suggests a better modularization (i.e. a more modular design), easier reuse and testing of the developed software products [43] due to the TF programming practice.

However, the superiority of the TF practice in the investigated context was not confirmed with respect to the two remaining areas of investigation. The mean value of weighted methods per class ($\text{WMC}_{\text{Mean}}$) was not significantly affected by the TF programming practice according to the meta-analysis, as well as selective meta-analysis, while the mean effect size represents a small effect according to the meta-analysis, as well as the selective meta-analysis.

The mean value of response for a class ($\text{RFC}_{\text{Mean}}$) was not significantly affected by the TF practice based on Fisher's method of combining $p$-values. The mean effect size represents a small effect according to meta-analysis as well as selective meta-analysis.

Moreover, the results revealed that the TF practice does not have a statistically significant impact, neither on the percentage of acceptance tests passed (PATP), which is an indicator of external code quality, nor on the number of acceptance tests passed per development hour (NATPPH), which is an indicator of development speed. The mean effect size of TF on the percentage of acceptance tests passed (PATP) represents a small effect. The mean effect size of TF on the number of acceptance tests passed per hour (NATPPH) represents a small effect, too.

Furthermore, the effect of the TF practice on unit tests was measured by branch coverage (BC) and mutation score indicator (MSI), which are indicators of the thoroughness and the fault detection effectiveness of unit tests, respectively. Relying on the preliminary results, BC was not significantly higher in the TFSP than in the TLSP group. However, the effect size was medium in size and therefore the effect of TF on branch coverage is a substantive effect. TLSP and TFSP did not significantly differ in MSI and the effect size was small.

## Limitations

The threats to the validity of the conducted experiments (e.g. relevance to industry) are thoroughly discussed in Sect. 10.5. The generalization of results is limited, since the analysed TF practice was applied to develop systems smaller than 10,000 lines of code. Further experimentation (e.g. in industrial context) is needed to establish evidence.

## Conclusions

The results reinforced the evidence regarding the superiority of the TF practice over the Test-Last Programming (TL) practice, with respect to the lower coupling between objects ($CBO_{Mean}$). However, the superiority of the TF programming practice in the investigated context was not supported with respect to the percentage of acceptance tests passed (PATP), the number of acceptance tests passed per development hour (NATPPH), weighted methods per class $WMC_{Mean}$, and response for a class ($RFC_{Mean}$).

Wroclaw, Poland Lech Madeyski

# Acknowledgements

*The movement called Extreme Programming is to my mind the most encouraging trend in software development today. It focuses us all on the real essentials: talent, discipline without dogma, teamwork, risk-taking, and light process. It poses a particular challenge to the manager, since it pushes control downward (managing people who are empowered to make decisions and even make their own mistakes is a lot harder than managing people who are obliged to shut up and do what you tell them to). I think XP will be a new generation's answer to the mindless regimentation embodied in the C.M.M. and other fat-book methodologies.*

Tom DeMarco

First of all, I would like to thank the head of my institute, Prof. Zbigniew Huzar, for a friendly atmosphere, making an excellent environment to work in, and for supporting me during my post-doctoral research. In 2007 I was invited to take part in XP, PROFES, ENASE, EuroSPI and CEE-SET conferences, either to present accepted papers or as an invited speaker. Without his support and encouragement, I would not have been able to manage that. I would also like to thank my Ph.D supervisor, Prof. Zygmunt Mazur, especially for introducing me to the Polish Information Processing Society, and Prof. Adam Grzech for his encouragement to start my Ph.D at the Wrocław University of Technology. Also, I would like to thank Prof. Janusz Górski, who confirmed my intuition about the importance of experimentation for software engineering research during our short, but fruitful discussion in 2003. That conversation had serious consequences and this book is one of them. I would also like to thank Prof. Jerzy Nawrocki, who taught me that valuable research results should be presented at major international conferences and published in recognized international journals. I hope I followed his advice. Furthermore, I would like to thank Prof. Leszek Maciaszek with whom I had an opportunity to come up with the idea of a new series of ENASE (Evaluation of Novel Approaches to Software Engineering) conferences. He encouraged me to play the role of devil's advocate at ENASE conferences in 2006 and 2007, along with top-ranked software engineering researchers (e.g. Prof. Mehmet Aksit, Prof. Brian Henderson-Sellers, Prof. Ulrich Eisenecker, Dr Giuseppe Berio, Prof. Stefan Kirn). In fact, challenging

the basic XP assumptions and practices and presenting the available empirical evidence and lessons learnt was an exciting experience. I would also like to thank the co-authors of my research papers, i.e. Marcin Kubasiak, Wojtek Biela, Michał Stochmiałek, Łukasz Szała, Norbert Radyk and all the other people who have contributed to my understanding of the topics presented in this book (e.g. Dr Małgorzata Bogdan, Dr Andy Field and Prof. Scott B. Morris for helpful suggestions concerning statistical analysis). Thanks to my MSc students (e.g. Marcin Kubasiak, Nicos Karagieorgopulus, Wojtek Biela, Michał Stochmiałek, Adam Piechowiak, Łukasz Szała, Tomasz Poradowski, Bartłomiej Bogaczewicz, Norbert Radyk, Piotr Przybył and Piotr Wójcicki). Without you, my research would not have been performed or would not have been so interesting.

If I were to point out only one book that influenced the way this research was conducted, it would be "Experimentation in Software Engineering" by Wohlin et al. [259]. I was lucky when I came across that book one day in the library of the Royal Institute of Technology (KTH) located in Stockholm. The International Conference on Product Focused Software Process Improvement (PROFES), led by the people from Fraunhofer Institute for Experimental Software Engineering (IESE) (e.g. Dr Jürgen Münch and Andreas Jedlitschka) and VTT (e.g. Prof. Pekka Abrahamsson), has to be mentioned as an extremely friendly empirical software engineering community to grow up in. Therefore, it was an exceptionally pleasant occasion for the author to become the member of that community and the winner of the "Best Paper and Best Presentation Award" at the PROFES conference in 2007.

I would also like to thank anonymous reviewers, as well as Prof. Zbigniew Huzar, Marian Jureczko and Wojciech Biela, for their valuable comments. Special thanks go to Ralf Gerstner of Springer Germany for providing professional advice during the publishing process and Małgorzata Rybak for assisting with proofreading.

Last but not least, I do like to express gratitude to my family and friends. Nothing would have happened without you – Iwona and Lech. Nothing would have been so colourful without you – Natalia, Judyta and Małgorzata.

Wrocław, Poland *Lech Madeyski*

# Contents

**1 Introduction** . . . . . 1
1.1 Test-First Programming . . . . . 1
1.1.1 Mechanisms Behind Test-First Programming that Motivate Research . . . . . 2
1.2 Research Methodology . . . . . 4
1.2.1 Empirical Software Engineering . . . . . 4
1.2.2 Empirical Methods . . . . . 5
1.3 Software Measurement . . . . . 8
1.3.1 Measurement Levels . . . . . 8
1.3.2 Software Product Quality . . . . . 9
1.3.3 Software Development Productivity . . . . . 12
1.4 Research Questions . . . . . 13
1.5 Book Organization . . . . . 13
1.6 Claimed Contributions . . . . . 14

**2 Related Work in Industrial and Academic Environments** . . . . . 15
2.1 Test-First Programming . . . . . 15
2.2 Pair Programming . . . . . 20
2.3 Summary . . . . . 23

**3 Research Goals, Conceptual Model and Variables Selection** . . . . . 25
3.1 Goals Definition . . . . . 25
3.2 Conceptual Model . . . . . 26
3.3 Variables Selection . . . . . 28
3.3.1 Independent Variable (IV) . . . . . 28
3.3.2 Dependent Variables (DVs) — From Goals to Dependent Variables . . . . . 32
3.3.3 Confounding Variables . . . . . 36

**4 Experiments Planning, Execution and Analysis Procedure** . . . . . 39
4.1 Context Information . . . . . 39
4.2 Hypotheses . . . . . 41

4.3 Measurement Tools ........ 42
4.3.1 Aopmetrics ........ 42
4.3.2 ActivitySensor and SmartSensor Plugins ........ 43
4.3.3 Judy ........ 43
4.4 Experiment ACCOUNTING ........ 44
4.4.1 Goals ........ 44
4.4.2 Subjects ........ 44
4.4.3 Experimental Materials ........ 44
4.4.4 Experimental Task ........ 45
4.4.5 Hypotheses and Variables ........ 45
4.4.6 Design of the Experiment ........ 45
4.4.7 Experiment Operation ........ 46
4.5 Experiment SUBMISSION ........ 46
4.5.1 Goals ........ 47
4.5.2 Subjects ........ 47
4.5.3 Experimental Materials ........ 47
4.5.4 Experimental Task ........ 48
4.5.5 Hypotheses and Variables ........ 48
4.5.6 Design of the Experiment ........ 48
4.5.7 Experiment Operation ........ 48
4.6 Experiment SMELLS&LIBRARY ........ 50
4.6.1 Goals ........ 50
4.6.2 Subjects ........ 50
4.6.3 Experimental Materials ........ 51
4.6.4 Experimental Tasks ........ 51
4.6.5 Hypotheses and Variables ........ 51
4.6.6 Design of the Experiment ........ 52
4.6.7 Experiment Operation ........ 52
4.7 Analysis Procedure ........ 53
4.7.1 Descriptive Statistics ........ 53
4.7.2 Assumptions of Parametric Tests ........ 53
4.7.3 Carry-Over Effect ........ 54
4.7.4 Hypotheses Testing ........ 55
4.7.5 Effect Sizes ........ 55
4.7.6 Analysis of Covariance ........ 56
4.7.7 Process Conformance and Selective Analysis ........ 57
4.7.8 Combining Empirical Evidence ........ 60

**5 Effect on the Percentage of Acceptance Tests Passed** ........ 61
5.1 Analysis of Experiment ACCOUNTING ........ 61
5.1.1 Preliminary Analysis ........ 61
5.1.2 Selective Analysis ........ 85
5.2 Analysis of Experiment SUBMISSION ........ 101
5.2.1 Preliminary Analysis ........ 101

5.2.2 Selective Analysis . . . . . 110
5.3 Analysis of Experiment SMELLS&LIBRARY . . . . . 116
5.3.1 Preliminary Analysis . . . . . 117
5.3.2 Selective Analysis . . . . . 121
5.4 Instead of Summary . . . . . 125

**6 Effect on the Number of Acceptance Tests Passed per Hour . . . . . 127**
6.1 Analysis of Experiment ACCOUNTING . . . . . 127
6.1.1 Descriptive Statistics . . . . . 128
6.1.2 Non-Parametric Analysis . . . . . 128
6.2 Analysis of Experiment SUBMISSION . . . . . 129
6.2.1 Descriptive Statistics . . . . . 129
6.2.2 Assumption Testing . . . . . 131
6.2.3 Non-Parametric Analysis . . . . . 131
6.3 Analysis of Experiment SMELLS&LIBRARY . . . . . 136
6.3.1 Descriptive Statistics . . . . . 136
6.3.2 Assumption Testing . . . . . 138
6.3.3 Non-Parametric Analysis . . . . . 139
6.4 Instead of Summary . . . . . 140

**7 Effect on Internal Quality Indicators . . . . . 141**
7.1 Confounding Effect of Class Size on the Validity of Object-Oriented Metrics . . . . . 141
7.2 Analysis of Experiment ACCOUNTING . . . . . 142
7.2.1 Descriptive Statistics . . . . . 142
7.2.2 Assumption Testing . . . . . 145
7.2.3 Mann–Whitney Tests . . . . . 145
7.3 Analysis of Experiment SUBMISSION . . . . . 147
7.3.1 Descriptive Statistics . . . . . 147
7.3.2 Assumption Testing . . . . . 150
7.3.3 Independent *t*-Test . . . . . 150
7.4 Analysis of Experiment SMELLS&LIBRARY . . . . . 152
7.4.1 Descriptive Statistics . . . . . 152
7.4.2 Assumption Testing . . . . . 153
7.4.3 Dependent *t*-Test . . . . . 155
7.5 Instead of Summary . . . . . 158

**8 Effects on Unit Tests – Preliminary Analysis . . . . . 159**
8.1 Analysis of Experiment SUBMISSION . . . . . 160
8.1.1 Descriptive Statistics . . . . . 160
8.1.2 Assumption Testing . . . . . 162
8.1.3 Mann–Whitney Test . . . . . 163

**9 Meta-Analysis** . . . . . 165
9.1 Introduction to Meta-Analysis . . . . . 166
9.1.1 Combining *p*-Values Across Experiments . . . . . 166
9.1.2 Combining Effect Sizes Across Experiments . . . . . 167
9.2 Preliminary Meta-Analysis . . . . . 171
9.2.1 Combining Effects on the Percentage of Acceptance Tests Passed (PATP) . . . . . 171
9.2.2 Combining Effects on the Number of Acceptance Tests Passed Per Development Hour (NATPPH) . . . . . 175
9.2.3 Combining Effects on Design Complexity . . . . . 177
9.3 Selective Meta-Analysis . . . . . 184
9.3.1 Combining Effects on the Percentage of Acceptance Tests Passed (PATP) . . . . . 185
9.3.2 Combining Effects on the Number of Acceptance Tests Passed Per Hour (NATPPH) . . . . . 187
9.3.3 Combining Effects on Design Complexity . . . . . 188

**10 Discussion, Conclusions and Future Work** . . . . . 197
10.1 Overview of Results . . . . . 197
10.2 Rules of Thumb for Industry Practitioners . . . . . 200
10.3 Explaining Plausible Mechanisms Behind the Results . . . . . 202
10.4 Contributions . . . . . 205
10.5 Threats to Validity . . . . . 206
10.5.1 Statistical Conclusion Validity . . . . . 206
10.5.2 Internal Validity . . . . . 209
10.5.3 Construct Validity . . . . . 211
10.5.4 External Validity . . . . . 213
10.5.5 Threats to Validity of Meta-Analysis . . . . . 216
10.6 Conclusions and Future Work . . . . . 217

**Appendix** . . . . . 219

**Glossary** . . . . . 223

**References** . . . . . 227

**Index** . . . . . 243

# Acronyms

| | |
|---|---|
| ANOVA | Analysis of Variance |
| ANCOVA | Analysis of Covariance |
| APA | American Psychological Association |
| BC | Branch Coverage |
| CBO | Coupling Between Objects (CK metric) |
| CK | Chidamber–Kemerer [44] |
| CVS | Concurrent Versions System |
| DevTech | Development Technique (e.g. TFSP) |
| DV | Dependent Variable |
| EBT | E-Business Technologies |
| EO | Effort Overhead |
| ES | Effect Size |
| ESE | Empirical Software Engineering |
| IDE | Integrated Development Environment |
| IV | Independent Variable |
| MSI | Mutation Score Indicator |
| NATP | Number of Acceptance Tests Passed |
| NATPPH | Number of Acceptance Tests Passed Per Hour |
| PATP | Percentage of Acceptance Tests Passed |
| PIJ | Programming in Java |
| PP | Pair Programming |
| RFC | Response for a Class (CK metric) |
| SbS | Side-by-Side Programming |
| SE | Software Engineering |
| SP | Solo Programming |
| SPSS | Statistical Package for the Social Sciences |
| SR | Speedup Ratio |
| SVN | Subversion (successor to the widely used CVS) |
| TDD | Test-Driven Development (also known as Test-First Programming) |
| TF | Test-First Programming (also known as Test-Driven Development) |
| TFSP | Test-First Solo Programming |
| TFPP | Test-First Pair Programming |
| TL | Test-Last Programming (also known as Test-Last Development) |

| | |
|---|---|
| TLD | Test-Last Development (also known as Test-Last Programming) |
| TLSP | Test-Last Solo Programming |
| TLPP | Test-Last Pair Programming |
| TVL | Test-Very-Last |
| VE | Virtual Enterprise |
| WMC | Weighted Methods per Class (CK metric) |
| WUT | Wroclaw University of Technology |
| XP | eXtreme Programming |