Jürgen Münch
Matias Vierimaa (Eds.)

LNCS 4034

Product-Focused Software Process Improvement

7th International Conference, PROFES 2006
Amsterdam, The Netherlands, June 2006
Proceedings

Springer

Volume Editors

Jürgen Münch
Fraunhofer Institute for Experimental Software Engineering
Fraunhofer-Platz, 67663 Kaiserslautern, Germany
E-mail: Juergen.Muench@iese.fraunhofer.de

Matias Vierimaa
VTT Electronics
Kaitovayla 1,90570 Oulu, Finland
E-mail: Matias.Vierimaa@vtt.fi

Library of Congress Control Number: 2006926730

CR Subject Classification (1998): D.2, K.6, K.4.2, J.1

LNCS Sublibrary: SL 2 – Programming and Software Engineering

ISSN 0302-9743
ISBN-10 3-540-34682-1 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-34682-1 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11767718 06/3142 5 4 3 2 1 0

Preface

The 7th International Conference on Product Focused Software Process Improvement (PROFES 2006) brought together researchers and industrial practitioners for reporting new research results and exchanging experiences and findings in the area of process and product improvement. The focus of the conference was on understanding, evaluating, controlling, and improving the relationship between process improvement activities (such as the deployment of innovative defect detection processes) and their effects on products (such as improved product reliability and safety). Consequently, major topics of the conference included the evaluation of existing software process improvement (SPI) approaches in different contexts, the presentation of new or modified SPI approaches, and the relation between SPI and new development techniques or emerging application domains.

The need for SPI is being widely recognized. Current trends in software intensive systems such as increased distribution of software development and growing dependability on software-intensive systems in everyday life emphasize this need. This implies the establishment of advanced process improvement capabilities and an adequate understanding of the impact of the processes on the generated products, services, and business value in different situations. Recent trends enforce the establishment of such capabilities: more and more products are being developed in distributed, global environments with many customer-supplier relations in the development chain. Outsourcing, off-shoring, near-shoring, and in-sourcing aggravate this trend. In addition, systems are being built from multiple disciplines (such as electronics, mechanics, and software). Supporting such distributed and multi-disciplinary development requires well-understood and accurately implemented development process interfaces, process synchronization, and process evolution. In addition, more and more organizations are forced to adhere to regulatory constraints that require the existence of explicit processes and the demonstration of adherence to those processes. Examples are the IEC 61508 standard for safety-related systems, the tailoring of ECSS (European Cooperation for Space Standardization) software engineering standards for ground segments in ESA (European Space Agency), or the German national standard V-Model XT for systems used by public authorities. Adhering to those standards requires systematic evolution of the existing processes. Finally, market dynamics force organizations to adapt better and faster to changes in the development environment and to enforce innovations (e.g., increase of reliability levels). These process changes impose risk challenges for SPI approaches. Advanced SPI is required to support the assessment of the impact of process changes and the flexible adaptation of processes. Due to the fact that software development processes are human-based and depend on the development context (including domain characteristics, workforce capabilities, and organizational maturity), changes to these processes typically cause significant costs and should be considered carefully. Alternative improvement options need to be evaluated with respect to their implementation cost and their potential impact on business goals.

Currently, two types of SPI approaches are mainly used in practice: a) continuous SPI approaches (also referred to as problem-oriented approaches) and b) model-based SPI approaches (also referred to as solution-oriented approaches).

Continuous SPI approaches (such as the Quality Improvement Paradigm, PDCA, or Profes) focus on selected problems of a software development organization and usually involve improvement cycles based on an initial baseline. One important advantage of continuous approaches is that they focus on solving specific problems by analyzing the problem at hand, implementing and observing problem-focused improvement actions, and measuring the effects of the actions. The interpretation of the measurement data is used as input for further optimization of the solution. In addition, solving one problem typically reveals further improvement potential in related areas. Continuous approaches are focused and, therefore, it is difficult to create an overall awareness for quality issues in a very large software organization with thousands of employees.

Model-based SPI approaches (such as ISO/IEC 15504, CMMI, or BOOTSTRAP) compare the current processes and practices of a development organization against a reference model or a benchmark. They provide so-called capability maturity levels with different sets of processes and practices. These levels define an improvement roadmap. The advantage of such models is that they can be easily used to enforce an awareness for quality issues in large organizations because many developers are involved in the improvement of the maturity level. From the management point of view, reaching a specific capability level can be defined as a clear and assessable goal. One important disadvantage is that model-based SPI approaches typically do not assess the impact of processes on product characteristics and therefore cannot be used to analytically identify and tackle process problems that cause concrete product deficiencies. Typically, it is checked whether a process or practice is in place, but its impact on a business goal or its value for the organization is not evaluated. The practices of the reference models are usually of a generic type and based on hypothesis. Having a high maturity level does not mean that the organization is successful in fulfilling its business goals (such as an appropriate trade-off between time-to-market and product quality).

Continuous and model-based SPI approaches can be seen as being complementary: model-based approaches can be used to identify problem areas and potential improvement options, and continuous approaches can be used to implement and optimize solutions. Although continuous approaches can be successfully applied without having a high maturity level, model-based approaches usually require continuous improvement at a certain maturity level.

In practice, the typical question is no longer whether process improvement is necessary, but how to define and implement a strategy for introducing advanced process improvement step by step and how to evaluate its success. Along with this, many research questions need to be solved.

The technical program was selected by a committee of leading experts in software process modeling and software process improvement research. This year, 55 papers from 26 nations were submitted, with each paper receiving at least three reviews. The Program Committee met in Amsterdam for one full day in February 2006. The Program Committee finally selected 26 technical full papers. The topics indicate that software process improvement remains a vibrant research discipline of high interest for industry. Emerging technologies and application domains, a paradigm shift from software to system engineering in many domains (such as automotive or space), and the need for better decision support for software process improvement is reflected in these papers.

The technical program consisted of tracks-decision support, embedded software and system development, measurement, industrial experiences, process improvement, agile development practices, and product line engineering. In addition, a track with 12 selected short paper presentations was added in order to demonstrate the variety of approaches, to support the discussions, and to exchange experience. We were proud to have four keynote speakers, Jan Bosch, Jan Jaap Cannegieter, Michiel van Gnuchten, and Barbara Kitchenham, as well as interesting tutorials and co-located workshops.

We are thankful for the opportunity to serve as program co-chairs for this conference. The Program Committee members and reviewers provided excellent support in reviewing the papers. We are also grateful to the authors, presenters, and session chairs for their time and effort to make PROFES 2006 a success. The General Chair, Rini van Solingen, and the Steering Committee provided excellent guidance. We wish to thank the Fraunhofer Institute for Experimental Software Engineering (IESE), the Centrum for Wiskunde en Informatika (CWI), VTT, the University of Oulu, Drenthe University, and Eindhoven University of Technology for supporting the conference. We would like to thank the Organizing Committee and all the other supporters for making the event possible. Last but not least, many thanks to Timo Klein at IESE for copyediting this volume.

April 2006

Jürgen Münch
Matias Vierimaa

Conference Organization

General Chair

Rini van Solingen, Drenthe University (The Netherlands)

Program Co-chairs

Jürgen Münch, Fraunhofer IESE (Germany)
Matias Vierimaa, VTT Electronics (Finland)

Organizing Chair

Mark van den Brand, Hogeschool van Amsterdam and CWI (The Netherlands)

Tutorial Chair

Dirk Hamann, Fraunhofer IESE (Germany)

Industry Chair

Carol Dekkers, Quality Plus Technologies, Inc.

PR Chair

Pasi Kuvaja, University of Oulu (Finland)

Publicity Chairs

Central Europe:	Michael Ochs, Fraunhofer IESE (Germany)
Southern Europe:	Gerardo Canfora, University of Sannio at Benevento (Italy)
USA:	Ioana Rus, Fraunhofer Center-Maryland (USA)
Canada:	Dietmar Pfahl, University of Calgary (Canada)
Japan:	Kenichi Matumoto, NAIST (Japan)
Korea:	Ho-Won Jung, Korea University (Korea)
Finland:	Tua Huomo, VTT Electronics (Finland)
Scandinavia:	Tora Dyba, Chief Scientist, SINTEF (Norway)
Benelux:	Ko Doorns, Philips
France:	Pierre-Etienne Moreau, INRIA/LORIA Nancy (France)
Oceania:	Bernard Wong, University of Technology, Sydney (Australia)
South America:	Christiane Gresse von Wangenheim (Brazil)

Program Committee

Pekka Abrahamsson, VTT Electronics, Finland
Andreas Birk, SD&M, Germany
Mark van den Brand, HvA & CWI, The Netherlands
Gerardo Canfora, University of Sannio at Benevento, Italy
Reidar Conradi, NTNU, Norway
Paolo Donzelli, University of Maryland - College Park, USA
Tore Dybå, SINTEF, Norway
Martin Höst, Lund University, Sweden
Frank Houdek, DaimlerChrysler, Germany
Tua Huomo, VTT Electronics, Finland
Hajimu Iida, Nara Institute of Science & Technology, Japan
Katsuro Inoue, Osaka University, Japan
Yasushi Ishigai, IPA, Japan
Janne Järvinen, Solid Information Technology, Finland
Erik Johansson, Q-Labs, Sweden
Philip Johnson, University of Hawaii, USA
Natalia Juristo, Universidad Politecnica de Madrid, Spain
Haruhiko Kaiya, Shinshu University, Japan
Kari Känsälä, Nokia Research Center, Finland
Masafumi Katahira, JAXA, Japan
Pasi Kuvaja, University of Oulu, Finland
Makoto Matsushita, Osaka University, Japan
Kenichi Matsumoto, NAIST, Japan
Pierre-Etienne Moreau, INRIA/LORIA, France
Maurizio Morisio, University of Turin, Italy
Jürgen Münch, Fraunhofer IESE, Germany
Paolo Nesi, University of Florence, Italy
Risto Nevalainen, STTF, Finland
Mahmood Niazi, Keele University, UK
Michael Ochs, Fraunhofer IESE, Germany
Hideto Ogasawara, Toshiba, Japan
Dietmar Pfahl, University of Calgary, Canada
Teade Punter, LAQUSO, The Netherlands
Karl Reed, La Tobe University, Australia
Günther Ruhe, University of Calgary, Canada
Ioana Rus, Fraunhofer Center - Maryland, USA
Kurt Schneider, University of Hannover, Germany
Carolyn Seaman, UMBC, Baltimore, USA
Veikko Seppäen, Elektrobit Ltd., Finland
Dag Sjöberg, University of Oslo, Norway
Matias Vierimaa, VTT Electronics, Finland
Otto Vinter, DELTA, Denmark
Giuseppe Visaggio, University of Bari, Italy
Hironori Washizaki, National Institute of Informatics, Japan
Isabella Wieczorek, Federal Ministery of Research and Education, Germany

Claes Wohlin, Blekinge Institute of Technology, Sweden
Bernard Wong, University of Technology Sydney, Australia

External Reviewers

Silvia Acuña, University of Madrid, Spain
Fabio Bella, Fraunhofer IESE, Germany
Jens Heidrich, Fraunhofer IESE, Germany
Sira Vegas, University of Madrid, Spain
Stein Grimstad, University of Oslo, Norway

Short Papers

Workshops

Tutorials

Productivity of Test Driven Development: A Controlled Experiment with Professionals

Gerardo Canfora[1], Aniello Cimitile[1], Felix Garcia[2], Mario Piattini[2], and Corrado Aaron Visaggio[1]

[1] RCOST- Research Centre on Software Technology
University of Sannio, Italy
{canfora, cimitile, visaggio}@unisannio.it
[2] ALARCOS Research Group- Information Systems and Technologies Department
UCLM-Soluziona Research and Development Institute
University of Castilla-La Manch Paseo de la Universidad, 4 – 13071 Ciudad Real, Spain
{Felix.Garcia, Mario.Piattini}@uclm.es

Abstract. With the growing interest for Extreme Programming, test driven development (TDD) has been increasingly investigated, and several experiments have been executed with the aim of understanding if and when it is preferable to the traditional practice of testing the code after having written it (named TAC in the paper). However, the research concerning TDD is at its beginning and the body of knowledge is largely immature. This paper discusses an experiment carried out within a Spanish software company with the aim of comparing productivity in TDD and TAC.

1 Introduction

Test driven development (TDD) belongs to the set of the extreme programming [1] practices, even if it might be adopted in any kind of software development process. According to TDD, unit testing results drive code development. As a first step, the developer defines the classes of the system in terms of public interfaces. Then, the test suite for each class is written: the suite must contain all the tests helpful for verifying that each method in the class exposes the correct behavior. Finally, the body of each method is completed throughout an iterative process, consisting of two activities: to execute the tests and, when some of them fail, to change the code in order to remove the bugs that potentially caused the failure. The process ends when all the tests succeed. TDD is widely considered a practice of code development rather than code testing, although the role of unit testing is relevant for defining the design strategy to adopt. TDD might be considered also an alternative to the traditional approach of testing the code after having written it, named 'test after coding' (TAC) in this paper. Recently some researchers investigated TDD: some experiments [2], [4], [7], and [9] produced evidence about the improvement of code quality achieved with TDD; however, some authors [2], and [6] did not find particular differences between TDD and TAC. There is not a wide consensus about the relationship between TDD and productivity, despite several studies obtained evidence that TDD is able to increase the productivity with respect to TAC [3], and [4]. Since both the practices involve

J. Münch and M. Vierimaa (Eds.): PROFES 2006, LNCS 4034, pp. 383 – 388, 2006.

coding and testing in a tightly interleaved process, we aim at understanding differences in the productivity in the two practices. We have carried out an experiment with the collaboration of professionals working in a Spanish Software Company, aiming at meeting the following research goal: **Analyse** *Test Driven Development and Test After Coding* **With the purpose of** *comparing them* **With respect to** *productivity* **From the point of view of** *the developers* **In the context of** *a group of professionals.* The work comprises two research questions:

R.1 *Is test driven development more productive than TAC from the viewpoint of testing?* In this case, productivity is intended as the time needed to write and execute assertions.

R.2 *How is the time employed in the two practices?* In order to have a deep insight of the two practices, we try to understand if the increasing of productivity means reducing the time for developing or for coding.

The paper proceeds as follows: section 2 describes the experimental design; data are analyzed in section 3; and, finally, section 4 draws the conclusions.

2 The Experiment Characterization

The experiment aimed at testing the following null hypotheses:

$\mathbf{H_{01}}$ **:** there is no difference in the productivity between test driven development and test after coding (helps to answer the research question R.1).

$\mathbf{H_{02}}$ **:** there is no correlation between productivity and the number of assertions in test driven development (helps to answer the research question R.2).

$\mathbf{H_{03}}$ **:** there is no correlation between the productivity and the number of assertions in test after coding (helps to answer the research question R.2).

The experiment was carried out in the facilities of the company Soluziona Software Factory located in Ciudad Real (Spain). The variables used in the data analysis are described in Table 1.

Table 1. Variables used in the experiment

Variable	Description
MeanTPA	*Mean Time per Assertion.* It is the time required to write and execute an assertion in the test suite. In both the practices the time for executing the assertion includes also the changing in the code, suggested by the failures of the test. It is assumed as an indicator of the *productivity.*
AssertTot	*Total Number of Assertions.* It is the total amount of assertions in the project. It is an indicator of the *quality of test* in the overall project. The greater is the number of assertions the greater is the number of aspects of the code which are tested.
MeanAPM	*Mean Assertion per Method.* It is the mean number of assertions written for a method. It is an indicator of the *accuracy of testing.* A high value of this metrics could indicate that all the methods of the classes received the same attention in the test.

28 employees of the company took part to the experiment: they have a BsC in Computer Science and a wide knowledge in software programming and modeling (UML, databases, etc.). The subjects were required to implement two assignments in two different runs, one assignment per run. The programming language was java, while ECLIPSE [10] and JUnit [11] were chosen as development environments. Subjects received a form for each run, which they had to fulfill with the information about the time and the assertions written for each assignment.

The experiment consisted of two runs; each run lasted five hours. Every subject implemented both the requirements and performed both the practices but in two different runs. The experimental design is illustrated in Table 2.

Table 2. The Experimental Design

	RUN I		RUN II	
Subjects	Treatment	Assignment	Treatment	Assignment
S_1	TDD	A1	TAC	A2
S_2	TAC	A1	TDD	A2
S_i	TDD	A2	TAC	A1
S_n	TAC	A2	TDD	A1

3 Data Analysis

Figure 1 compares the performances of the subjects when using the two practices, TAC and TDD.

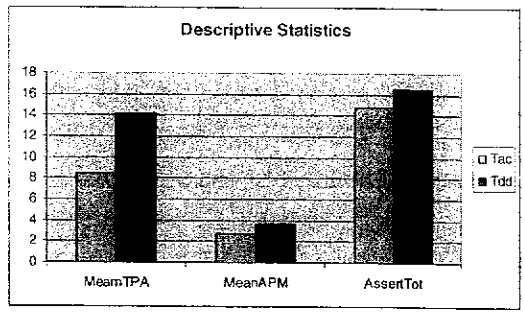

Fig. 1. Performances of the experimental samples

The data suggests that differences between the two practices exist. TAC reduces the mean time spent for each assertion (MeanTPA) , thus it is more productive than TDD; conversely, TDD increases the density of assertions per method (MeanAPM) and the total number of assertions (AssertTot); thus, TDD might determine more accuracy and quality in the testing than TAC. On one hand, the iterative approach of TDD requires more time than TAC in order to deliver a method. On the other hand, since the developer writes the tests before the code, greater attention is devoted to

testing and, consequently, the developer increases the number of assertions. Table 3 shows that the difference in productivity is statistically significant; Mann-Whitney test was used and the p-level was fixed at 0.05. Analysis of correlation can help answer the research question R.2, and understand more precisely how the additional time in TDD is employed.

Table 3. Tests of Hypothesis H_{01}

Testing	Rank Sum (a)	Rank Sum (b)	p-level
MeanTPA (TDD)-MeanTPA (TAC)	846.0000	585.000	**0.0037374**

Fig.2 shows that in TDD the MeanAPM decreases when the MeanTPA increases this might be due to the iterative process of TDD which forces the developer to modify recursively the code of a method until the correspondent tests do not succeed. The tests are defined before the code and the additional time is mainly used to improve the code rather than the tests.

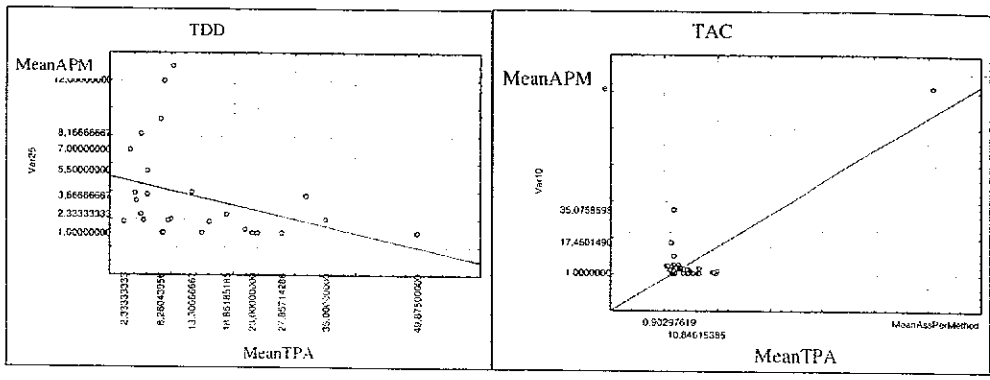

Fig. 2. Correlation between MeanTPA and MeanAPM in TDD and TAC

In TAC the opposite phenomenon appears: the time for writing assertions increases with the number of assertions. This indicates that the additional time is used mainly to improve the test cases for each method, by defining further assertions, rather than to modify the code: as a matter of fact the developer writes the tests only after having defined the code. Fig. 3 shows that in both the techniques the mean time for assertions determines a decrement in the total number of assertions. This is explainable by deriving the variables with respect to the MeanTPA variable.

If we consider that AssertTot= (number of Methods*MeanAPM), then:

d(AssertTot)/d(MeanTPA)=(MeanAPM)*d(number of Methods)/d(MeanTPA)
+ (number of Methods)*d(MeanAPM)/d(MeanTPA).

If d(AssertTot)/d(MeanTPA) < 0, then:

- in TAC d(MeanAPM)/d(MeanTPA) > 0, and consequently d(number of Methods)/d(MeanTPA) should be negative. This means that to increase the time spent for the assertions reduces the number of methods, which is an indicator of modularity: the additional time is not used to increase the number of methods in the code;
- in TDD d(MeanAPM)/d(MeanTPA) <0, thus it is not possible inferring the sign of d(number of Methods)/d(MeanTPA). It is possible to deduce only that if it is positive, it grows slower than the absolute value of d(MeanAPM)/d(MeanTPA). In TDD, if the additional time is dedicated to increase modularity (number of methods), this is smaller than the increasing of assertions density, anyway: TDD advantages the testing aspect.

Table 4 shows that there is statistical evidence for the correlation of all the discussed cases. Spearman's method was applied for testing hypotheses because the sample data set is not normally distributed and the p-level was fixed at 0.05.

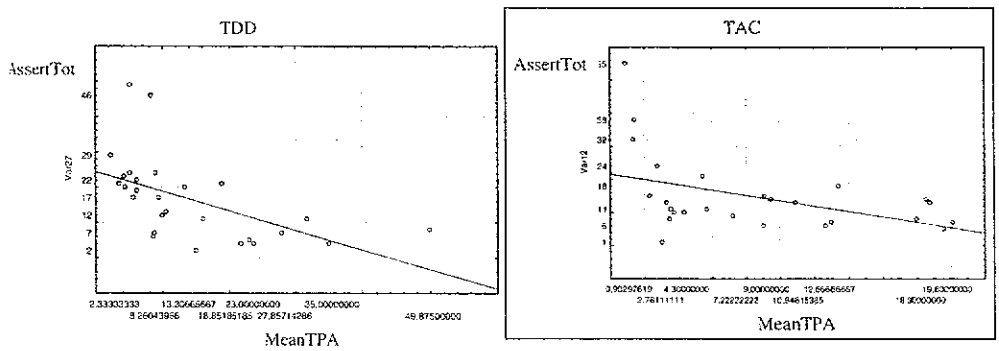

Fig. 3. Correlation between MeanTPA and AssertTot in TDD and TAC

Table 4. Statistical Tests of Hypotheses H_{02} and H_{03}

Correlation	Valid N	Spearman R	T(N-2)	p-level
MeanTPA-MeanAPM [TDD]	27	-0.442272	-2.46561	**0.0208**
MeanTPA-MeanAPM [TAC]	31	-0.387853	-2.26603	**0.003108**
MeanTPA-AssetTot [TDD]	27	0.549229	3.286153	**0.00300**
MeanTPA-AssetTot [TAC]	27	0.777829	6.188211	**0.00002**

4 Conclusions

This paper discusses the results of an experiment carried out in a Spanish Software Company. The research aimed at comparing productivity in test driven development and testing after coding. It emerged that: TAC reduces the time for writing assertions and modifying the code, according to the feedback of the tests; TDD increases the

total number of assertions and the density of assertions per method, which could be indicators of unit testing quality and accuracy; and, finally, the additional time while testing is mainly exploited to improve code in TDD, whereas in TAC it is used to improve the testing strategy. In both the practices, the additional time is never used to increase the number of methods, which is indicator of modularity.

Acknowledgements

We would like to thank professionals of Soluziona Software Factory for their active participation and collaboration. This research has been partially funded by the projects: MAS (Dirección General de Investigación del Ministerio de Ciencia y Tecnología, TIC 2003-02737-C02-02), MECENAS (Junta de Comunidades de Castilla-La Mancha, Consejería de Educación y Ciencia, PAI06-0024-2494) and FAMOSO (Ministerio de Industria, Turismo y Comercio, FIT-340000-2005-161).

Bibliography

[1] Beck K. Extreme Programming explained: Embrace change. Addison-Wesley: Reading Massachusetts, 1999.

[2] Edwards S. Using test-driven development in the classroom: Providing students with automatic, concrete feedback on performance. *Proc. of the Int'l Conference on Education and Information Systems: Technologies and Applications*, EISTA'03, Orlando, Florida USA, 2003.

[3] Erdogmus, H. and Morisio, M. On the effectiveness of test-first approach to programming. *IEEE Transactions on Software Engineering*, 31(1), 2005, IEEE CS Press), pp. 1-12.

[4] George B. and Williams L. A structured experiment of test-driven development *Information and Software Technology*, 46(5), 2004, Elsevier,pp.337–342.

[5] Geras A., Smith M. and Miller J. A Prototype Empirical Evaluation of Test Driven Development. *Proc. of the 10th Inter'l Symposium on Software Metrics (METRICS'04)* Sidney, Australia, 2004, IEEE CS.

[6] Muller M. and Hagner O. Experiment about Test-first programming. *Proc. of Empirical Assessment in Software Engineering*, Keele, UK, 2002.

[7] Pankur M., Ciglaric M., Trampus M. and Vidmar T. Towards empirical evaluation of test-driven development in a university environment. Proc. of EUROCON 2003 Computer as a Tool. Ljublijana, Slovenia, 2003,IEEE CS Press.

[8] Sjoberg D., Anda B., Arisholm E., Dyba T., Jorgensen M., Karahasanovic A., Koren E and Vokác M. Conducting Realistic Experiments in Software Engineering. *Proc. of the 2002 Int'l Symposium on Empirical Software Engineering (ISESE'02)*, Nara, Japan 2002, IEEE CS Press.

[9] Williams L., Maximilien E., and Vouk M. Test-driven development as a defect-reduction practice. *Proc. of the 14th IEEE Int'l Symposium on Software Reliability Engineering* Denver, Colorado, USA, 2003, IEEE CS Press.

[10] The ECLIPSE IDE. Available in http://www.eclipse.org/

[11] The JUnit Testing Framework. Available in http://www.junit.org