# Evaluating Advantages of Test Driven Development: a Controlled Experiment with Professionals

Gerardo Canfora
Research Centre on Software Technology (RCOST)
viale Traiano, 1
82100 Benevento, Italy
+390824305555
canfora@unisannio.it

Aniello Cimitile
Research Centre on Software Technology (RCOST)
viale Traiano, 1
82100 Benevento, Italy
+390824305555
cimitile@unisannio.it

Felix Garcia
Alarcos Research Group, University of Castilla-La-Mancha
Paseo de la Universidad, 4
13071 Ciudad Real, Spain
+34926295300
Felix.Garcia@uclm.es

Mario Piattini
Alarcos Research Group,
University of Castilla-La-Mancha
Paseo de la Universidad, 4
13071 Ciudad Real, Spain
+34926295300
Mario.Piattini@uclm.es

Corrado Aaron Visaggio
Research Centre on Software Technology (RCOST)
viale Traiano, 1
82100 Benevento, Italy
+390824305555
visaggio@unisannio.it

## ABSTRACT
Test driven development (TDD) is gaining interest among practitioners and researchers: it promises to increase the quality of the code. Even if TDD is considered a development practice, it relies on the use of unit testing. For this reason, it could be an alternative to the testing after coding (TAC), which is the usual approach to run and execute unit tests after having written the code. We wondered which are the differences between the two practices, from the standpoint of quality and productivity. In order to answer our research question, we carried out an experiment in a Spanish Software House. The results suggest that TDD improves the unit testing but slows down the overall process.

## Categories and Subject Descriptors
D.2.8 [**Software Engineering**]: Metrics – *performance measures ,process metrics, product metrics*

**General Terms:** Management, Measurement, Performance, Design, Experimentation.

**Keywords:** Empirical Software Engineering, Test Driven Development, Process Quality.

## 1. INTRODUCTION
TDD is a key practice of extreme programming (XP) [1]: it prescribes that the code is developed or changed exclusively on the basis of the unit test results. As a first step, the developer defines the classes of the system together with the correspondent class interfaces. Then, the developer composes the test suite for each class: the test suite must include the assertions needed to verify the behavior of all classes' methods. 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, which are supposed to be the cause of the failure. The process is over when all the tests succeed.

The test suite is not only the container of the tests to be run, but it becomes an essential component of the system, too: it is used as (part of) design documentation, as it describes the dynamic aspects of the system, by mapping the expected values returned by each method with the ones passed as input. It entails the following advantages:

- such documentation is embedded in the code, thus the lifecycle of code and documentation should merge in only one.
- it provides an unambiguous and immediate definition of functional quality for the code: if the tests succeed, the code is accepted as good.
- the access is fast: the developer just needs to execute the suite in order to get the content of the documentation, rather than browsing many sheets full of different diagrams.

TDD is not intended to be a quality assurance technique, even if it lets the developer make preliminary assessments of the code while writing. TDD is considered as a practice of code development rather than code testing, but the role of unit testing is relevant for establishing the design strategy and the algorithms to adopt. Consequently, when dealing with TDD, the issues


*ISESE'06*, September 21-22, 2006, Rio de Janeiro, Brazil.

concerning testing must be taken into account, as well as the ones concerning coding. From this standpoint, TDD might be considered as an alternative to TAC, the more traditional approach to unit testing, consisting in writing and running the tests after that the code is written. We believe that: (i) TDD is more time consuming than TAC; but (ii) TDD improves the quality of unit testing. On one hand, as the developer is forced to continuously skip from test to code (and vice versa) until the tests succeed, the iterative process of TDD seems to us more costly than the traditional TAC. As a matter of fact, in TAC the two-phases, coding and testing, are executed quite in a rigorous sequence, except for the bug-fixing, which is usually very localized and entails a very few iterations code-tests. On the other hand, as the developer tends to obtain the greater information he can from the results of testing in order to write correctly the code, TDD facilitates the accuracy and the precision of test cases.

With the aim of verifying our conjecture, we have carried out an experiment with the collaboration of professionals working in a Spanish Software Company.

The research goal is stated as:

**Analyze** Test Driven Development and Test After Coding

**With the purpose of** comparing

**With respect to** performances of testing

**From the point of** view of the developers

**In the context of** a group of professionals.

The research goal consists of two research questions:

- R.1. Is TDD more productive than TAC from the viewpoint of testing? In the case, the product is intended as the set of test cases and correct code; the code is considered correct if all the related tests succeed. Thus, 'productivity' is seen as the efficiency in producing test cases and correct code.
- R.2. Can TDD improve the quality of unit testing? We evaluate the differences between TDD and TAC in terms of accuracy and precision of unit tests.

The paper proceeds as follows: section 2 illustrates the related work; section 3 describes the experimental design; data are analyzed in section 4; the limits of the experiment are discussed in section 5; and, finally, section 6 draws the conclusions.

## 2. RELATED WORK

TDD is gaining a wide acceptance, thanks to the growing popularity of XP. In the last few years, a number of empirical studies investigated mainly quality and productivity achieved with the test driven development.

George and Williams performed a set of structured experiments [5] in which 24 pairs of professional programmers were involved. One group developed a small JAVA program by applying TDD, whereas the other (control) group used the waterfall lifecycle model. The pairs using TDD produced a better quality code (18% higher) than the pairs who did not use TDD, although the former required 16% longer time. This study provided evidence that TDD increases the level of tests passed and improves the quality of the code.

Williams et al. carried out a case study in IBM [13]; the process consisted of developing an automatic package of test cases once the system was designed with UML. As a result, the code developed by applying TDD had 40% fewer defects when compared with the code of an experienced team using an ad-hoc testing approach. Besides, TDD had a minimal impact on the developer's productivity.

Edwards proposes the use of TDD as a testing practice in a classroom. TDD was evaluated with a pilot study in a computer science undergraduate classroom [3]. Students who applied TDD, produced code with 45% fewer defects than the students who did not use TDD.

Müller and Hagner [9] describe an experiment aiming at comparing TDD and TAC. The subjects were postgraduate students who had to write the code of a graphical library. The results indicated that there were no significant differences between TDD and TAC in terms of reliability and productivity. Geras et al. conducted a similar experiment with senior undergraduate students [6]. It emerged that there was a very little difference in productivity, but there were significant differences regarding the failure frequency in favor of TDD. Pancur et al. also investigated the differences between TDD and TAC [10], throughout an experiment with senior students attending the same class. Some of the students wrote the program code using TDD and the rest of students applied TAC; in both cases an iterative process was applied and automated support for logging the results of test runs (frequency of test runs, passed and not passed test cases) was in place. The differences between TAC and TDD were reduced by using many iterations and testing tools in both the processes.

Erdogmus and Morisio evaluated other relevant quality factors of TDD [4]. They aimed at understanding if programmers using TDD wrote more test cases than programmers who applied the traditional TAC approach. In order to achieve this, an experiment with undergraduate students was performed: subjects had to develop a JAVA program, consisting of several small stories, each one describing a concrete feature of the product. The students were divided in two groups: the experiment group applied TDD while the control group applied TAC. Both groups performed an incremental process: they could add new features and execute the corresponding regression tests for each increment. As a result, students who used TDD wrote a greater number of test cases and they tended to be more productive. However, this did not result in a proportional improvement of quality.

Although a discrete number of studies concerns TDD, the building of knowledge body around the practice is yet at an initial stage. Therefore, it is necessary to confirm the obtained results by comparing the different conclusions reached with the different experiments, and by studying in depth other aspects of TDD.

So far, there is no work dealing with the TDD, if considered as well as a process which merges together testing and code in a unique practice. This work aims at providing an analysis on the productivity and on the effectiveness of TDD. Controlled experiments carried out in industrial settings, as well as the one

presented here, might be useful to reinforce the validity of the findings.

## 3. THE EXPERIMENT

The experiment aimed at testing the following null hypotheses.

**$H_{01}$**: there is no difference in the productivity between TDD and TAC.

**$H_{02}$**: there is no difference in quality of unit tests between TDD and TAC.

$H_{01}$ helps to answer the research question R.1, whereas $H_{02}$ the research question R.2.

**Subjects**

The experiment was carried out in the facilities of the Soluziona Software Factory, a software house located in Ciudad Real, Spain. The professional business of Soluziona Software consists of software development and maintenance in the following areas: gas, water and electricity management systems, economic management of quality and environment, market simulators, economic-financial management, corporative systems, public health system, e-commerce, telecommunications, etc. 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.).

**Assignments**

The subjects were required to realize a system, named "TextAnalyzer", in order to satisfy two different requirements in two different runs, and precisely, one requirement per run. The programming language was java, while ECLIPSE [14] and JUnit [15] were chosen as development environments. For precision's sake, the subjects were required to write the code and the test suites for the requirements.

Subjects received two forms that they had to fulfill, one for run, in the following way:

- to indicate the requirement realized in each run together with the practice performed (TDD or TAC).
- for each requirement, subjects had to list the correspondent assertions they wrote in order to test the methods of the JAVA classes, which satisfied such requirement.
- for each assertion subjects had to write down
  - the start time; and
  - end time, which takes into account when the test is overcome, i.e. when every bug is detected and the test is passed without any failure.

Exemplar forms are showed in the appendix together with the two assignments. All the experimental material was translated in Spanish by the Spanish authors.

**Rationale for sampling Population**

The 28 subjects were selected among a set of professionals with comparable skills: they had 5 years of experience in using java and in computer programming. All the subjects had previously participated in several software engineering projects and had at least one year of experience as employee of the company. As the subjects had no previous experience on TDD, we performed training sessions before the experiment, as discussed in the '*process*' sub-section.

**Variables**

The variables are described in Table 1.

**Table 1. Variables used in the experiment**

| Variable | Description | Meaning |
|---|---|---|
| **Hypothesis $H_{01}$** | | |
| **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 changing the code for fixing the bugs emerged from the test. | It is assumed as an indicator of the *productivity.* The product is considered as the test cases and the corrected code. |
| **MeanTime** | It is the mean time for writing and executing a test suite. | They are indicators of the *effort* spent by subjects when performing the practices |
| **TotalTime** | It is the amount of time spent by the subject for realizing the overall assignment. | |
| **Hypothesis $H_{02}$** | | |
| **MeanAPM** | Mean Assertion per Method. It is the mean number of assertions written for a class' method. | It is assumed as an indicator of test cases' accuracy. The more are the assertions dedicate to a method the more is complete the test case for that method. |
| **AssertTot** | Total Number of Assertions. It is the total amount of assertions in the project. | It is assumed as an indicator of the precision 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 covered by the test. |

**The process**

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

For instance, among the n subjects, the subject Sj performed TDD at the first run for implementing the A2's requirements, and, the A1's requirements were developed at the second run, with the TAC practice. Before the experiment, the subjects took part to a training session, which included a seminary about test driven development, and lab exercises in order to increase the familiarity with the practice.

**Table 1:The experimental design**

| | RUN I | | RUN II | |
|---|---|---|---|---|
| **Subjects** | **Treatment** | **Assignment** | **Treatment** | **Assignment** |
| $S_1$ | TDD | A1 | TAC | A2 |
| $S_2$ | TAC | A1 | TDD | A2 |
| $S_j$ | TDD | A2 | TAC | A1 |
| $S_n$ | TAC | A2 | TDD | A1 |

# 4. ANALYSIS OF DATA

Data presented in this section have been cleaned by outliers with the aid of a tool, which performs statistical analyses. In order to get suddenly an idea of the overall experiment's results, histograms were used. In the appendix, the data set is reported in greater detail.

## 4.1 Descriptive Statistics

Figure 1 compares the mean values of the evaluated metrics, whereas box plots of data set can be found in the appendix.

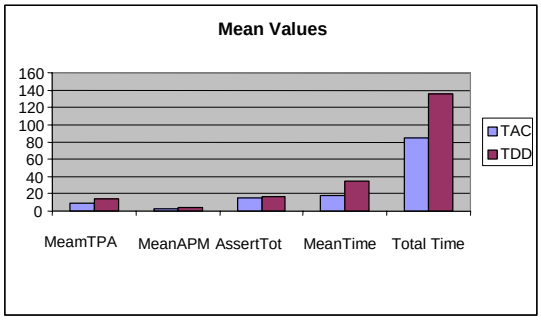


**Figure 1: Comparing mean values of data sets.**

It emerges that:

- TDD requires more time than TAC for the execution of the tasks: TDD slows down the overall rhythm of the work (see TotalTime metric) and also the mean throughput of developers (see MeanTPA). This might be due to the iterative nature of the TDD's process. This finding has not a negative connotation at all, as we believe that the exceeding time is used to increase the quality of code. Unfortunately, we cannot demonstrate it here: quality's aspects of the software are not observed, since it is not the focus of the paper.

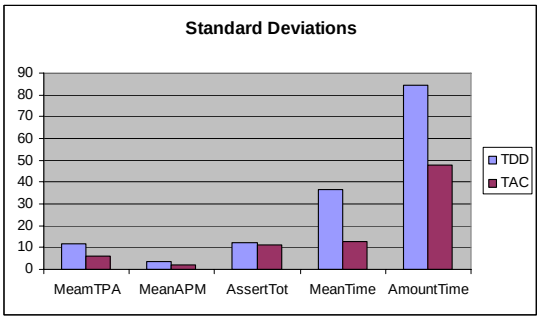


**Figure 2: Comparing standard deviations of data sets.**

- TDD fosters a greater accuracy (MeanAPM) and precision (AssertTot) of testing. TDD leads the subjects to analyze in depth the test cases for all the methods, obtaining an overall improvement of the unit tests. We observed that subjects designed accurately the use scenarios of different methods, identifying completely equivalence classes, selecting thoroughly the input in order to detect the bugs. Conversely, when they apply TAC, the test is faced with a kind of monolithic approach, where the tests for all the methods are grouped together in larger test cases; this drives the subjects to be less precise when defining the assertions, and instead of dividing the problem in many sub-problems and deal with them separately, they tend to face the problem at one time. As a consequence, developers are more prone to neglect some aspects in the test cases: the quality of unit testing is, sometime, seriously affected.

Figure 2 shows the standard deviation values of the data sets. Standard deviation computes the dispersion around the central value and it is an indicator of the variability of the sample's data set.

TDD is more predictable than TAC for the data sets of all the metrics. Predictability is related to the opportunity to make good estimations during the planning phase of software projects. This could be a point in favor of TDD, as it could be a motivation to adopt the practice in real contexts.

Such a property might be explained with the fact that TDD puts a great emphasis on the unit testing: as each method must have the correspondent unit test, all the subjects must produce roughly the same number of test cases with a similar precision.

Conversely, in TAC the responsibility of the test cases quality is left to the individual developer, since it is realized after that the code is written. This entails that some developers spend more time testing because they analyze the scenarios in greater detail, whereas others prefer to stop at a certain point; hence the greater repeatability of the TDD's performances.

## 4.2 Testing of Hypotheses

Table 3 shows the results of hypotheses testing; Mann-Whitney tests were used because the sample data set was not normally distributed, and the p-level was fixed at 0.05.

**Table 3. Testing of hypotheses**

| Testing | Rank Sum (a) | Rank Sum (b) | p-level | Comment |
|---|---|---|---|---|
| Hypothesis $H_{01}$ | | | | |
| MeanTPA TDD(a)-TAC(b) | 846.00 | 585.00 | **0.037374** | There is **evidence** that TDD requires more time per assertions than TAC |
| TotalTime TDD(a)-TAC(b) | 885.00 | 546.000 | **0.005501** | There is **evidence** that TDD requires an overall amount of time longer than TAC |
| MeanTime TDD(a)-TAC(b) | 861.00 | 570.00 | **0.018412** | There is **evidence** that TDD requires more time in average than TAC |
| Hypothesis $H_{02}$ | | | | |
| AssertTot TDD(a)-TAC(b) | 763.00 | 668.00 | 0.544772 | There is no evidence that TDD produces more assertions than TAC |
| MeanAPM TDD(a)-TAC(b) | 755.50 | 675.50 | 0.633341 | There is no evidence that TDD produces more assertions per method than TAC |

The statistical test of hypotheses produced the following results:

- there is evidence that TDD requires more time than TAC.
- there is no evidence that TDD lets developers realize more accurate and precise test cases than TAC.

## 4.3 Lessons Learned to improve the experimental design

No particular matters occurred during the experiment nor subjects complained for anything, thus we suppose that the experimental design was good enough. However, two considerations must be highlighted:

- it might be useful to enlarge the time window; TDD is very time consuming, and hurry might drive the subjects to do less than they wish. We believe that this explains why we did not obtain empirical evidence on the quality data sets.
- our impression is that the strongest difference between the two practices may be perceived on the code: analysis of the quality obtained is worth investigating.

# 5. THREATS OF VALIDITY

**Threats to construct validity**

The dependent variables aimed at capturing the productivity of the evaluated practices. Since they were obtained from the data collected by forms filled in by subjects, the measurement was objective. In order to facilitate the accuracy of data, the subjects were provided with an example of a fulfilled form and the process was carefully explained during the prior training session. Besides, in order to indicate correct times, all subjects used their own computer system clocks.

**Threats to Internal Validity**

The following issues have been dealt with:

- Differences among subjects. Using a within-subjects design, error variance due to differences among subjects was reduced. The subjects were professionals with experience in JAVA programming, familiar with the ECLIPSE environment, and with the assignments. Moreover, the subjects learnt TDD, TAC, and JUnit, during the same (introductory) seminar before the experiment.
- Learning effects. The subjects were required to deal with only one assignment for each run and the assignments were the as independent as possible, in order to cancel the learning effects. There is no evidence that learning effects occurred between the two runs, as Mann Whitney tests show in Table 4.
- Fatigue effects. On average, the experiment lasted 13 hours (three hours for training session and five hours for each run). However, this time was distributed into three consecutive days: the first was dedicated to train the subjects, while the second and the third ones were dedicated to the first and second run, respectively. This arrangement was chosen in order to reduce as much as possible the fatigue effects. As a result, fatigue effects did not appear. As a confirmation, some subjects asked for a longer time to accomplish better the assignments.
- Persistence effects. In order to avoid persistence effects, the experiment was run with subjects who had never done a similar experiment.
- Subject motivation. Professionals showed a great interest in taking part to a scientific experiment. They were pleased to exercise TAC, TDD and JUnit which could bring benefits to their daily work.
- Other factors. Plagiarism and influence among subjects were controlled by supervising the runs. Regarding the experimental package, each subject performed both the practices, but in different runs; each assignment was solved by the same number of subjects, and the two practices were used equally to implement the

assignments. This reduced the possible threats related with likely differences in the assignments. As Table 5 shows, there are no statistically significant differences among the assignments. The Mann- Whitney test was used and the p-level fixed at 0.05.

**Table 4 . Comparison between the first and second run**

| Testing | Rank Sum (a) | Rank Sum (b) | p-level |
|---|---|---|---|
| MeanTPA I Run (a) - II Run (b) | 691.000 | 740.000 | 0.775584 |
| TotalTime I Run (a) - II Run (b) | 638.000 | 793.000 | 0.509 |
| MeanTime I Run (a) - II Run (b) | 673.000 | 758.000 | 0.9715 |
| AssertTot I Run (a) - II Run (b) | 663.500 | 767.500 | 0.8376 |
| MeanAPM I Run (a) - II Run (b) | 691.000 | 740.000 | 0.7755 |

**Table 5. Comparison between the first and the second assignment**

| Testing | Rank Sum (a) | Rank Sum (b) | p-level |
|---|---|---|---|
| MeanTPA I Asgmt (a) - II Asgmt (b) | 755.000 | 676.000 | 0.154045 |
| TotalTime I Asgmt (a) - II Asgmt (b) | 758.000 | 672.500 | 0.136817 |
| MeanTime I Asgmt (a) - II Asgmt (b) | 721.500 | 709.500 | 0.407382 |
| AssertTot I Asgmt (a) - II Asgmt (b) | 659.500 | 771.500 | 0.622097 |
| MeanAPM I Asgmt (a) - II Asgmt (b) | 660.000 | 771.000 | 0.97124 |

**Threats to External Validity**

Three threats to external validity have been identified which could limit the ability to generalize the research results to the population under study [12].

- Material and tasks. In the experiment, the scope of the assignments was not actually comparable to real projects of the company, since the material and assignments were designed considering the restrictions of time. Therefore assignments more similar to industrial projects shall be considered in future studies.
- Subjects. As the experiment has been performed by professionals, generalization of the results was facilitated.
- Environment. The experiment was performed in one of the work rooms of the company and the tasks had to be solved with ECLIPSE, JUnit and the computers used by the professionals in their daily work. The overall settings provided the subjects with a very realistic environment.

# 6. CONCLUSIONS

TDD is a practice which prescribes to write and change the code of a class' method only on the basis of the correspondent unit test' results.

Although TDD is considered a 'development practice' rather than a testing practice, it is actually twofold, because it includes both coding and testing aspects in a tightly interleaved process. Since TDD is a practice per se and it might be used also independently from the other agile practices and in other kinds of software processes, we wondered if and when TDD can be preferred to the traditional TAC.

We believe that TDD is more time consuming than TAC, but leads developer to design more precise and more accurate test cases.

We carried out an experiment in order to verify our thesis, and obtained the following results:

- there is statistical evidence that TDD requires more time than TAC: this does not necessarily entail that TDD deteriorates the productivity, as the quality of code could be improved. As discussed previously, it is probably due to the iterative process of TDD; the process of TAC is more linear and requires a smaller number of feedbacks and reworks on code.
- there is not statistical evidence that TDD brings about more accurate and precise unit tests than TAC, even if subjects who used TDD outperformed those who use TAC, during all the experimental runs. We are convinced that TDD increases such quality aspects and that evidence might be obtained in a longer experiment, where differences between the two practices could be more evident.
- TDD lets a greater predictability of performances than TAC: such a result might be helpful when estimating project's costs. This is due to the fact that the reworks on code in TAC depends on the willing and care of the developer, thus the time are more varying. In TDD the load of testing depends on the class design, thus more predictable.

The most relevant limit of our experiment stands in its nature of controlled experiment: the available time window and the tasks were exemplar ones. In the real scenarios, tasks are more complex. However, experiments in vitro are necessary for exploring the research field before executing experiments on the field, that is for:

- understanding the most relevant issues which deserve to be investigated and which do not: we believe that code's quality might provide helpful insight.
- adjusting experimental design on the basis of the feedback from the subjects and the matters arisen during the runs; by enlarging the time window, we can obtain a greater evidence of the difference between the practices.

A strength point is the collaboration of professionals, as it helped to enforce the external validity.

As future steps, we are planning:

- to replicate the experiment in other environments, such as universities and companies, in order to enforce the validity of the results; our aim is to enlarge the observation time up to six or 12 months, and
- to analyze the relationship with the intrinsic quality of the code delivered, especially with regard to software maintainability.

## 7. ACKNOWLEDGEMENTS

We would like to thank you managers of Soluziona for allowing us to carry out the experiment and the engineers who took part to it. This research has been partially supported by the projects: FAMOSO, partially funded by Ministerio de Industria, Turismo y Comercio, FIT-340000-2005-161 Plan Nacional de Investigacion Cientifica, Desarrollo e Innovacion Tecnologica 2004-2007 and "Fondo Europeo de Desarrollo Regional (FEDER)", European Union, and MECENAS (Junta de Comunidades de Castilla-La-Mancha, Consejeria de Educacion y ciencia, PBI06-0024).

## 8. REFERENCES

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

[2] Darcy, D.P., and Kemerer, C.F. OO Metrics in Practice. *IEEE Software 22*, (November-December 2005), pp-17-19.

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

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

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

[6] Geras, A., Smith, M. and Miller, J. A Prototype Empirical Evaluation of Test Driven Development. In *Proc. of the 10th Inter'l Symposium on Software Metrics (METRICS'04)*, (Sidney, Australia, 2004), IEEE CS Press, pp. 405-416.

[7] Grable, R., Jernigan, J., Pogue, C., and Divis, D. Metrics for Small Projects: Experiences at the SED. *IEEE Software 16*, (March-April 1999), pp. 21-29.

[8] Kitchenham, B., and Mendes, E. Software Productivity Measurement Using Multiple Size Measurement. *IEEE Transaction on Software Engineering 30*, (December 2004), pp.1023-1035.

[9] Muller, M., and Hagner, O. Experiment about Test-first programming. In *Proc. of Empirical Assessment in Software Engineering (EASE'02)*, (Keele, UK, 2002).

[10] Pankur M., Ciglaric M., Trampus M. and Vidmar T. Towards empirical evaluation of test-driven development in a university environment. In *EUROCON 2003. Computer as a Tool. The IEEE Region 8 , Volume: 2*, (Ljublijana, Slovenia, 2003), IEEE CS Press, pp.83, 86.

[11] Premraj, R., Kitchenham, B., Shepperd, M., and Forselius, P. An Empirical Analysis of Software Productivity over Time. In *Proc. of the 11th IEEE Int'l Software Metrics Symposium (METRICS '05)*, (Como, Italy, 2005), IEEE CS Press, pp.37.

[12] Sjoberg, D., Anda, B., Arisholm, E., Dyba, T., Jorgensen, M., Karahasanovic, A., Koren, E. and Vokác, M. Conducting Realistic Experiments in Software Engineering. In *Proc. of the 2002 Int'l Symposium on Empirical Software Engineering (ISESE'02)*, (Nara, Japan, 2002), IEEE CS Press, pp.17.

[13] Williams L., Maximilien, E., and Vouk, M. Test-driven development as a defect-reduction practice. In *Proc. of the 14th IEEE Int'l Symposium on Software Reliability Engineering (ISSRE'03)*, (Denver, Colorado, USA, 2003), IEEE CS Press, pp.34-48.

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

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

## 9. APPENDIX

**Table 6. Mean values**

| Indicator | TAC | TDD |
|---|---|---|
| MeanTPA | 8.4895 | 14.18117 |
| MeanAPM | 2.768151 | 3.62927 |
| AssertTot | 14.76923 | 16.51852 |
| MeanTime | 17.45015 | 35.07586 |
| TotalTime | 85.03846 | 135.9231 |

**Table 7. Standard deviations**

| Indicator | TAC | TDD |
|---|---|---|
| MeanTPA | 6.32758 | 11.76298 |
| MeanAPM | 2.10803 | 3.393464 |
| AssertTot | 11.41012 | 11.99025 |
| MeanTime | 12.58014 | 36.6277 |
| TotalTime | 47.73886 | 84.73886 |

**Assignation**

To develop a system named "TextAnalyzer", which must operate with the following text:

"Todo pasa y todo queda, pero lo nuestro es pasar, pasar haciendo caminos, caminos sobre el mar. Nunca perseguí la gloria, ni dejar en la memoria de los hombres mi canción; yo amo los mundos sutiles, ingrávidos y gentiles, como pompas de jabón. Me gusta verlos pintarse de sol y grana, volar bajo el cielo azul, temblar súbitamente y quebrarse... Nunca perseguí la gloria. Caminante, son tus huellas el camino y nada más; caminante, no hay camino, se hace camino al andar."

**Assignment 1**

The program must calculate the frequency of the words in the text (expressed in percentage), and the position of their first occurrences. As a result, the program must display a list with the four most frequent words and their first occurrence. An example of the output of the program could be the following:

- The word "aldea" firstly appears in the 7th position and its frequency is 40%.

- The word "camino" firstly appears in the 50th position and its frequency is 25%.

- The word "todo" firstly appears in the 10th position and its frequency is 5.4%.

- The word "llegada" firstly appears in the 20th position and its frequency is 5%.

Once the former list has been displayed, the program must offer the user the option of displaying the rest of words (ordered by frequency), by showing the following message: "Do you want to obtain the frequency (%) and the first occurrence of the rest of words in the text (Y/N)?

**Assignment 2**

The program must calculate the maximum and minimum distance (expressed in number of words) between two words indicated by the user. For example, regarding the provided text, if the user types the words: "caminante" and "camino", the program must display: "the minimum distance is 2 and the maximum distance is 14". If one or both the words do not appear in the text the program must return -2. If the two words indicated by the user are the same word, the program must return -1.

**Forms**

Exemplar forms filled in follow.

| # Req | Kind of Practice | Unit Test Class | Number of Assertions | Start Time | End Time |
|---|---|---|---|---|---|
| 1 | TDD | FinderTest | 6 | 12.10 | 12.40 |
| | | Occurrence Calculator Test | 5 | 12.45 | 13.50 |

| # Req | Kind of Practice | Unit Test Class | Number of Assertions | Start Time | End Time |
|---|---|---|---|---|---|
| 2 | TAC | MaxOccur Calculator Test | 7 | 14.00 | 14.15 |
| | | ListPublish erTest | 2 | 14.20 | 16.00 |

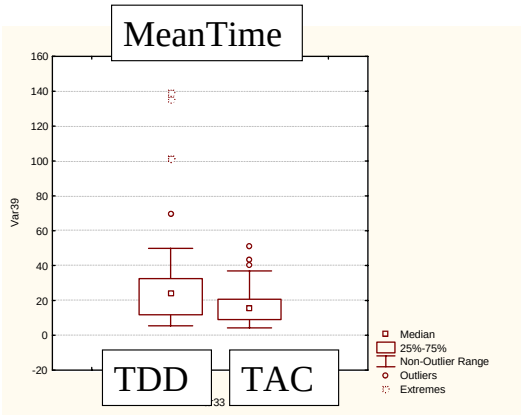


**Figure 3. Box plots of meantime.**

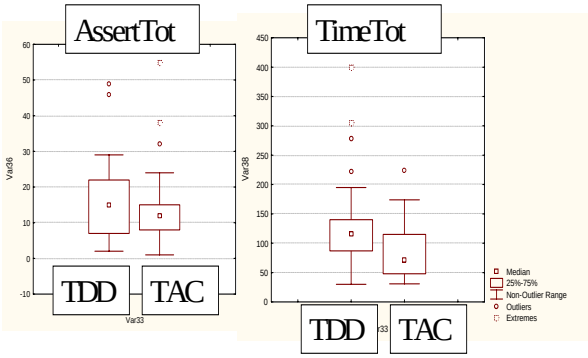


**Figure 4. Box plots of asserttot and timetot.**

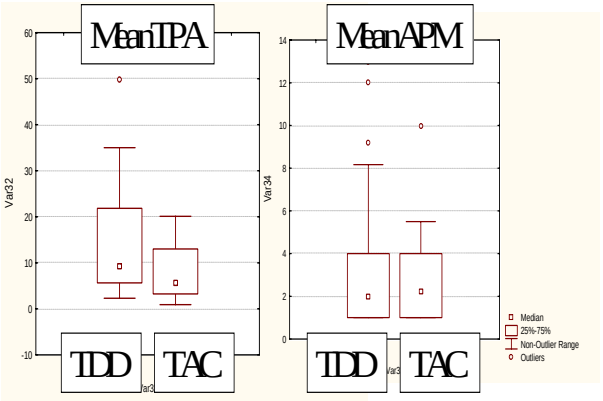


**Figure 5. Box plots of meantpa and meanapm**