Reports with TDD and Mock Objects: an Improvement in Unit Tests

Alan S. C. Mazuco
Department of Computer Science
Masters in Applied Computing
University of Brasilia (UnB)
Campus Darcy Ribeiro
Brasilia, DF, Brazil
Email: alanmazuco@hotmail.com

Edna Dias Canedo
Faculdade UnB Gama - FGA
University of Brasilia (UnB)
Brasilia, DF, Brazil
Email: ednacanedo@unb.br

***Abstract*—The construction of reports in software engineering, although considered a simple task, is sometimes extremely difficult for the developer, especially if the report has a rich amount of detail and web software as a backdrop. This article will show how you can reduce the stress of developers using agile methodologies, such as Test Driven Development (TDD) associated with Mock Objects. Software testing is gaining the attention of software scholars because of the huge impact on the quality they produce and of the reduced delivery time. This study was driven by a shortage of literature and comes, as appropriate, to demonstrate how it is possible to reduce the drudgery of creating reports using open source tools like Jaspersoft and their implementers, such as IReport, along with the Eclipse IDE. The study was based on experiments carried out in the Brazilian Army's Performance Management Project, with a team of professionals who used mock objects to save time and improve performance gain speed and which also used performance in conducting their work and also the TDD Methodology as the main reference. From the results, empirical observation showed us the best and worst aspects encountered by participants during their work.**

***Keywords*–TDD, Test Driven Development; Mock Objects; Reports.**

I. INTRODUCTION

The Brazilian Army's Performance Management project, materialized in a corporate system of the same name, the "SGD", or "PMS" - Performance Management System, came into the world in order to carry out and follow the evaluations of its internal public, whose final destination is the subsidy decision-making of subsidies for several finalistic programs. The project has become a priority in the high command of the Brazilian Army, and there were several factors that led to such a distinction. However, what most drew attention was how quickly it got off the ground and won the web pages in the form of robust and efficient software. The system was completed in six months of development. In the seventh month, the system went into production as planned. The project's success was due to the fact that the project manager had decided to use agile methodologies, such as Test Driven Development (TDD), throughout the project development phase.

This case study will permeate some definitions, guiding and reinforcing the experiences developed along the Brazilian Army's Performance Management project, and showing why the TDD process has been tenaciously important in the software development process.

What caught the eye with the production of reports using mock objects was the increased pace of implementation, as well as the reduced fatigue of programmers. Such improvements led to an increase in the satisfaction of business owners due to the high demand for increasingly rich reports.

The results were obtained by performing an experiment conducted in the General Headquarters of the Brazilian Army, Such experiment followed empirical methodologies, allowing us to observe the best and worst aspects encountered by members of the participating teams.

For a better understanding, this paper is structured as follows: Section 2 presents several concepts on the subject at hand, as well as some major works reported in the community. This is very important because we found basis in the research literature that supports the experiments that were conducted. The Section 3 describes how the experiment was conducted, the subject of this study, the methodologies used and the composition of the teams that performed in it. The Section 4 presents the results, collected in the light of the experiment, using previously selected indicators.

II. RELATED WORKS

The utilization of Mock Object simulates the behavior of complex real objects and are therefore very useful when used in conjunction with TDD practices. This section explores some of the literature on the subject.

A. The Problem of Errors

A study published by the National Institute of Standards and Technology[1] and also by the United States Department of Commerce reveals that software errors cost around $60 billion to the US economy each year. Much has been said about techniques to minimize the catastrophic effect caused by software errors, as we see in Borges [1]. Such techniques include reusing code that has been widely tested and is trusted, as well as exhaustive verification techniques and validation tests performed by a team of testers.

As reported in Leon and Kochs [2], agile methodologies create ever-growing controversy, having their true effectiveness auestioned and putting their advocates in a heated battle of claims. However, the practice has shown that processes arising from the agile methodology bring many benefits for development, culminating in the satisfaction of clients.

Fig. 1 shows that the use of agile methodologies can systematically reduce the cost of making code changes. Regarding this, Beck [3] believes that the following are key aspects of agile methodologies:

- Effective (fast and adaptive) response to change;
- Effective communication among all stakeholders;
- Drawing the customer onto the team;
- Organizing a team so that it is in control of the work performed; Quick, incremental delivery of software.

According Baumeister and Wirsing [4], there are benefits to an evolutionary approach in which the developer writes the test before they write the functional code needed to satisfy that test.

Below, we can see a graphical representation of this study.

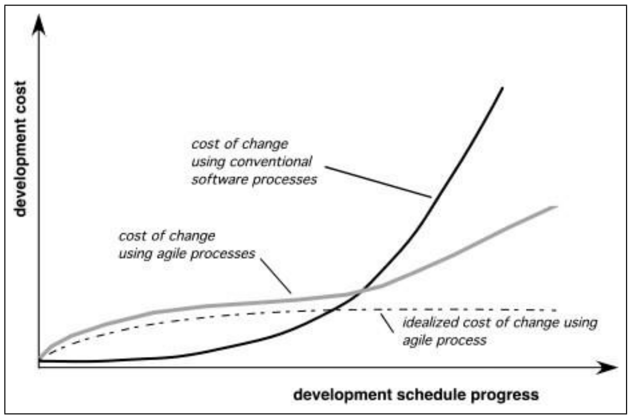


Figure 1. Agility and the Cost of Change [3]

B. The TDD as Tonic

TDD is defined as a set of techniques of Extreme Programming (XP) associated with agile methodologies.

According to Beck [5], an agile method could be compared to driving a car, where the driver has the task of driving the vehicle to his destination safely, without committing traffic offenses.

According Baumeister and Wirsing [6], there are benefits to an evolutionary approach in which the developer writes the test before he writes the functional code needed to satisfy that test.

According to Marrero and Settle [7], TDD is a way to reflect on modeling before writing the code itself. But as reported by Baumeister and Wirsing [6], the testdriven development is a programming technique where the main goal is to write clean functional code from a test that has failed.

According to a manifesto published in 2001 [3], we see that:

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- Individuals and interactions over processes and tools;
- Working software over comprehensive documentation;
- Customer collaboration over contract negotiation;
- Responding to change over following a plan That is, while there is value in the items on the right, we value the items on the left more.

As Fowler has shown in [8] developers should worry about performing a refactoring of the code in order to optimize it more and more, and TDD processes are perfectly consonant with this approach.

Fig. 2 illustrates the TDD methodology. Note that, while the traditional approach first encodes the main business rule for later testing, the general idea of the TDD is to mitigate the code through unit testing until it passes the test [9], where frameworks like JUnit (Java) are often used by more savvy developers.

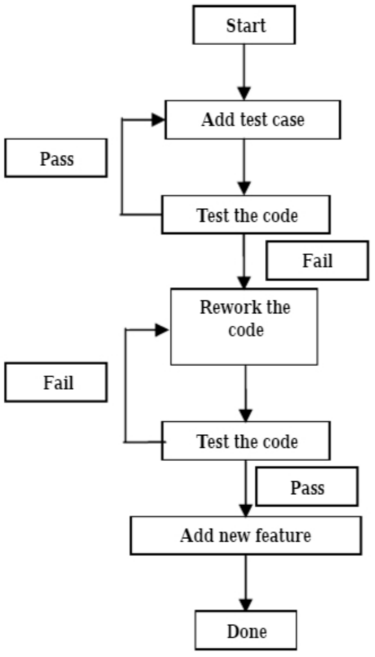


Figure 2. Concept of the TDD, in general lines [10]

C. Mock Objects par Excellence

The main ground of TDD is to use intensive testing, even before coding the main object. Therefore, the concept of Mock Objects fits like a glove, from which false objects could be created and tested with the main code, to obtain the desired result.

A mock object, according Mackinnon et al [11], is a substitute implementation to emulate another domain code. It has to be simpler than the actual code, not a duplicate implementation, and allow you to set your status to help the test.

As seen in Stroustrup and Lenkov [12], it can be difficult to conceive detailed unit testing in scoped languages such as Java, without breaking the scope. To remedy that, the unit testing technique for field packs was created.

Fowler is emphatic when he says [13]:

“The term *Mock Objects* has become a popular one to describe special case objects that mimic real objects for testing. Most language environments now have frameworks that make it easy to create mock objects. What’s often not realized, however, is that mock objects are but one form of special case test object, one that **enables a different style of testing**”.

From what we can presume, and as we pointed out in the text above, the use of Mock Objects is not limited to performing unit testing using JUnit or similar frameworks. Rather, Mock Objects are flexible enough to perform tests for various purposes. In the case of our experiment, it was used to build reports. The results were measured and scientifically proven, and have brought many benefits to our team of developers, allowing greater flexibility in the process.

Testing with Mock Objects has been the key to solving problems, as it transfers the actual behavior of the object to a close-to-real fictional situation, being in perfect conformity with the principle of Demeter Law [14]:

“Code with the encapsulation of ideas and modularity, easily following the object-oriented technique to the programmer... while minimizing code duplication, the number of method arguments, and the number of methods per class.”

Nevertheless, Freeman et al. [15] state the following:

“Mock Objects is an **extension to Test-Driven Development** that supports good Object-Oriented design by guiding the discovery of a coherent system of types within a code base. It turns out to be less interesting as a technique for isolating tests from third-party libraries than is widely thought.”

According to Brown and Tapolcsanyi [16], Mock Objects are divided into patterns, as we can see in Table 1:

TABLE I. PATTERNS CATALOG FOR MOCK OBJECTS [16].

Pattern Name	Synopsis
MockObject	Basic mock object pattern that allows for testing a unit in isolation by faking communication with collaborating objects.
Test Stubs	
MockObject via Factory	A way of generating mock objects, utilizing existing factory methods.
Self Shunt	Unit Test code serves as the mock object by passing an instance of itself.
Pass in Mock Collaborator	Pass in a mock object in place of the actual collaborating object.
Mock Object via Delegator	Creates a mock implementation of a collaborating interface in the Test class or mock object.

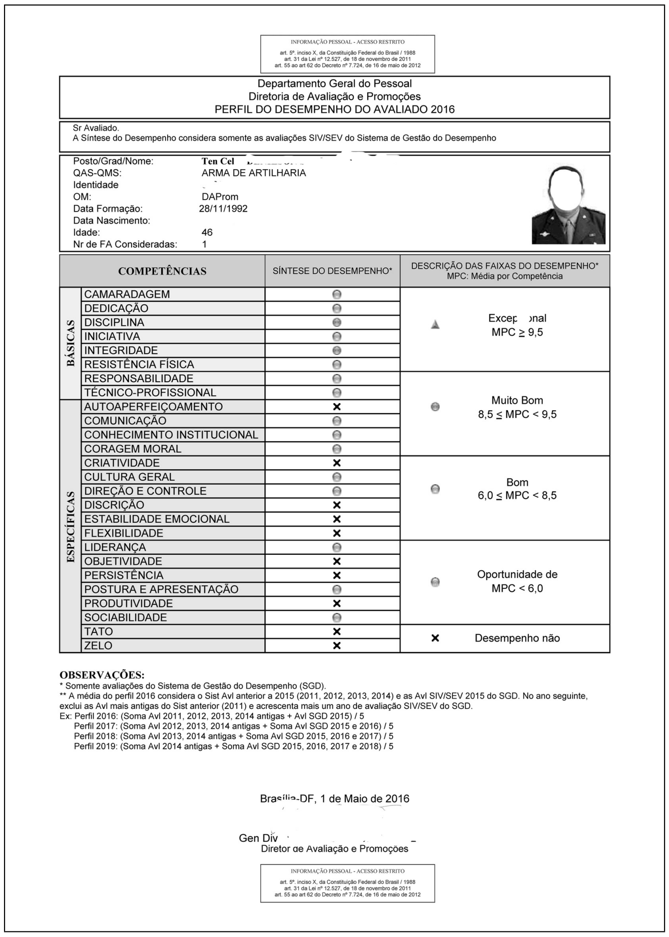
The following patterns will be added next year for 2004 PLOP:
-Mock Objects via CrossPoints;
-Write Testable Code; and
-Mock Object with Guard.

During the experiments conducted in this study, as seen in the table, we used the Mock Object described below.

III. THE EXPERIENCE

1) The Mock Object used: According to Brown and Tapolcsanyi [16] and Fowler [13] Mock Objects can be used to build repeatable, automated, and highly leveraged Unit Tests. In many cases, setting up Mock Object frameworks that “emulate” the real world objects is necessary. Thus, the pattern used in the experiment was the Self Shunt. This pattern fit like a glove, as the Java report-creation operations are extremely repetitive, bringing some fatigue obstacles and construction time.

Fig. 3 shows the report to be created in the experiment, just to get an idea of the complexity of development. From there, a mock object containing all the data would be used to create form this report using fictitious data.



INFORMAÇÃO PESSOAL - ACESSO RESTRITO
art. 5º. inciso X, da Constituição Federal do Brasil / 1988
art. 31 da Lei nº 12.527, de 18 de novembro de 2011
art. 55 ao art 62 do Decreto nº 7.724, de 16 de maio de 2012

Departamento Geral do Pessoal
Diretoria de Avaliação e Promoções
PERFIL DO DESEMPENHO DO AVALIADO 2016

Sr Avaliado.
A Síntese do Desempenho considera somente as avaliações SIV/SEV do Sistema de Gestão do Desempenho

Posto/Grad/Nome:	Ten Cel
QAS-QMS:	ARMA DE ARTILHARIA
Identidade	
OM:	DAProm
Data Formação:	28/11/1992
Data Nascimento:	
Idade:	46
Nr de FA Consideradas:	1

COMPETÊNCIAS		SÍNTESE DO DESEMPENHO*	DESCRIÇÃO DAS FAIXAS DO DESEMPENHO* MPC: Média por Competência	
BÁSICAS	CAMARADAGEM	●	▲	Excep onal MPC ≥ 9,5
	DEDICAÇÃO	●		
	DISCIPLINA	●		
	INICIATIVA	●		
	INTEGRIDADE	●		
	RESISTÊNCIA FÍSICA	●		
	RESPONSABILIDADE	●	●	Muito Bom 8,5 ≤ MPC < 9,5
	TÉCNICO-PROFISSIONAL	●		
ESPECÍFICAS	AUTOAPERFEIÇOAMENTO	✗		
	COMUNICAÇÃO	●		
	CONHECIMENTO INSTITUCIONAL	●		
	CORAGEM MORAL	●		
	CRIATIVIDADE	✗	●	Bom 6,0 ≤ MPC < 8,5
	CULTURA GERAL	●		
	DIREÇÃO E CONTROLE	●		
	DISCRIÇÃO	✗		
	ESTABILIDADE EMOCIONAL	✗		
	FLEXIBILIDADE	✗		
	LIDERANÇA	●	●	Oportunidade de MPC < 6,0
	OBJETIVIDADE	✗		
	PERSISTÊNCIA	✗		
	POSTURA E APRESENTAÇÃO	●		
	PRODUTIVIDADE	✗		
	SOCIABILIDADE	●		
	TATO	✗	✗	Desempenho não
	ZELO	✗		

OBSERVAÇÕES:
* Somente avaliações do Sistema de Gestão do Desempenho (SGD).
** A média do perfil 2016 considera o Sist Avl anterior a 2015 (2011, 2012, 2013, 2014) e as Avl SIV/SEV 2015 do SGD. No ano seguinte, exclui as Avl mais antigas do Sist anterior (2011) e acrescenta mais um ano de avaliação SIV/SEV do SGD.
Ex: Perfil 2016: (Soma Avl 2011, 2012, 2013, 2014 antigas + Avl SGD 2015) / 5
Perfil 2017: (Soma Avl 2012, 2013, 2014 antigas + Soma Avl SGD 2015 e 2016) / 5
Perfil 2018: (Soma Avl 2013, 2014 antigas + Soma Avl SGD 2015, 2016 e 2017) / 5
Perfil 2019: (Soma Avl 2014 antigas + Soma Avl SGD 2015, 2016, 2017 e 2018) / 5

Brasília-DF, 1 de Maio de 2016

Gen Div
Diretor de Avaliação e Promoções

INFORMAÇÃO PESSOAL - ACESSO RESTRITO
art. 5º. inciso X, da Constituição Federal do Brasil / 1988
art. 31 da Lei nº 12.527, de 18 de novembro de 2011
art. 55 ao art 62 do Decreto nº 7.724, de 16 de maio de 2012

Figure 3. Report template that served as “guide” to build the report.

Professionals in JasperReports know the difficulty in formatting a report as complex as this, so much so that some prefer not to venture. However, reports provide a lot of function points as a measure to estimate the system size, ensuring larger, more rewarding results from a financial point of view [17]. Therefore, no software factory will ignore this practice.

To aid in the experiment, the stakeholders have provided a psychologist, whose main function would be to analyze the developers over of the more cognitive aspects, such as fatigue, while also monitoring of the interviews.

2) Description of Experiment: The team of the Brazilian Armys Performance Management Project performed the experiment. It was conducted in an isolated room, and carried out by three pairs of certified developers, all of which were timed, in accordance to the following:

TABLE II. DESCRIPTION OF EXPERIENCE.

OBJECT OF THE EXPERIMENT	Create a complex report of one of the system's activities, SGD, containing a relatively heavy image and 45 attributes.
USED TOOLS	Programming language: Java other tools: IDE Eclipse, JUnit and IReport.
METHODOLOGY	Alpha team: Development using TDD only. Beta team: Development using TDD with Mock Object;
TIMING	The stopwatches were linked at the beginning of implementation experiment, but had no finishing time preset.

A. *Sequence of Activities*

First, we conceived the construction of a report, where the sequence of the activities of developers should rotate around through the steps shown in Fig. 4:

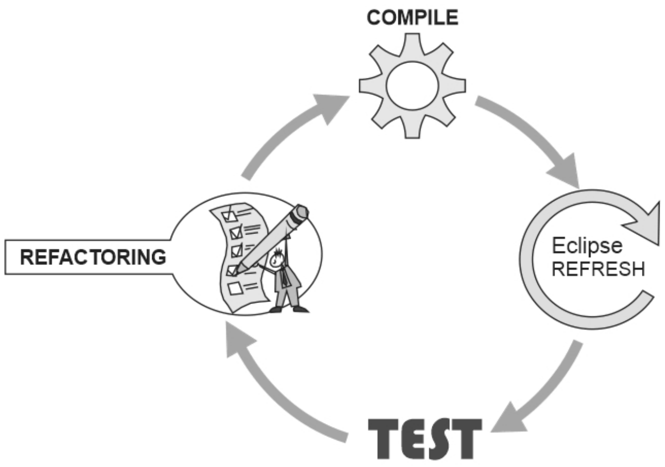


Figure 4. Report construction with TDD

Performing that activity to build reports with the aforementioned tools is extremely strenuous, since the developer uses a lot of manual effort to draw pictures, frames, lines and put texts in places previously defined by the template that serves as a guide, see Fig. 3. The time spent on the activity could harm the progress of timelines, creating frustrations and stress. Because of this, the developer should focus their tasks exactly in this manual effort, not worrying about the collection and processing of data, and thus gaining additional time.

When using Eclipse IDE, developer reports required the execution of a "refresh" in preparation for the report, after refactoring and recompiling the report. Only after that could they call the run-time report. Such process is tiresome and time-consuming. When running a report he needed to go to the database and bring the data to fill the report. This does not seem very productive.

Before connecting the chronometers, each team received the complete description for making the Mock Object and the corresponding business rules, consisting of:

1) **Alpha Team**: The report should present the data from a database, consisting basically of an object with 45 attributes and one more consisting of type byte array - an image. It was delivered to staff together with the business rules for the connection and information, as well as a report template as a guide, Fig. 3.

2) **Beta Team**: The report should present the data from a Mock Object, consisting basically of an object with 45 attributes and one more consisting of type byte array - an image. It was explained also that this object should be an identical copy of the original object. The names of the attributes for making the Mock Object were delivered along with the business rules as well as the report template as a guide, Fig. 3.

IV. RESULTS

For a better understanding, we have grouped the results into two subsections, with the metrics in the first subsection and the analysis in the second.

A. *Presentation*

Prior to the experiment, we created an index to measure the work of developers around seven items we deem relevant for the study. This index was scaled from 2 to 10 points. To understand it better, Table 3 shows this index in more detail. Fig. 5, along with Table 4, present the final result of the experiment.

TABLE III. DESCRIPTION AND CONTENT OF THE INDEXES.

Index	*Orientation index*
10	Higher than expected.
8	Sometimes higher than expected.
6	Expected.
4	Sometimes lower than expected.
2	Below expected.

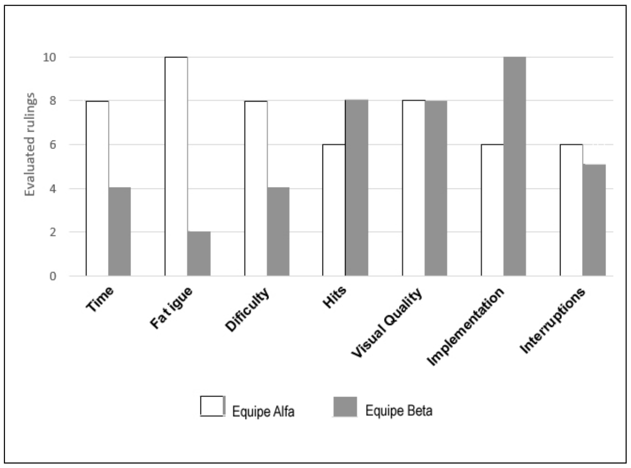


Figure 5. Results with seven variables.

Where:

1) **Time**: Average Results of chronometers after delivery of work, marked to the accuracy of seconds;
2) **Fatigue**: The developer reported extreme tiredness, often interpreted as a painful sensation, result of physical and mental effort;
3) **Difficulty**: The developer reported feeling some difficulty performing the work required;
4) **Hits**: By examining the code, the developer has submitted correct answers;
5) **Visual quality**: On visual inspection, the work presented refinements;
6) **Implementation**: correct implementation of the business rules and alignment with the template which was the implementation guide;
7) **Interruptions**: The developer interrupted the proceedings to ask questions.

Table 4 shows the final result of the experiment, the time spent per each participant to carry out the work, and the number of interruptions of each to the removal of doubts.

TABLE IV. TIMEKEEPING TEAMS.

Team	*Developer*	*Time*	*Interruptions*
A	1	3h 35m 37s	6
	2	3h 15m 08s	5
	3	3h 25m 15s	7
B	4	1h 35m 47s	4
	5	1h 15m 22s	6
	6	1h 40m 57s	4

The Time column displays the time, we measurements for all developers and the Interruptions column shows the stops made to clear doubts from developers in relation to business rules considered by them as confusing. Each participant possessed their corresponding timer, in order not to invalidate the experiment.

B. Analysis

Analyzing the results, we confirmed our suspicions and were not surprised that the Beta Team presented the best performance, both from a qualitative point of view, as well as quantitative. The disparities are more relevant in the following items: **Fatigue**, **Time** and **Difficulty**. Notably, the Fatigue presented by Alpha Team was most notorious.

TIME

In relation to the measured time, we see that no individual participant fulfilled the expectations. However, the Beta Team, which used Mock Objects, spent nearly half the time of the Alpha Team to solve the problem. Fig. 6 shows the data of Table 4 in a chart showing the real-time taken from their stopwatches on the y-axis, where we can see more clearly the disparity. The developer 6 from Beta Team has most experience among the other.

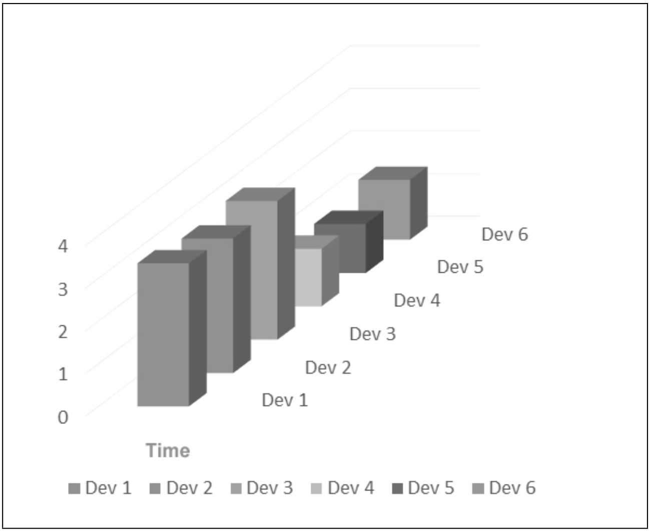


Figure 6. Analysis of time developers.

FATIGUE

Fig. 7 shows the measurement of fatigue. Such measurement was done empirically, by conducting interviews with developers. The score was measured based on reports and on a psychological evaluation. To this regard, a second interview was necessary, this time with a psychologist, to evaluate the general conditions of the participants and greater accuracy of the calculation. The graph below is materializing these indexes.

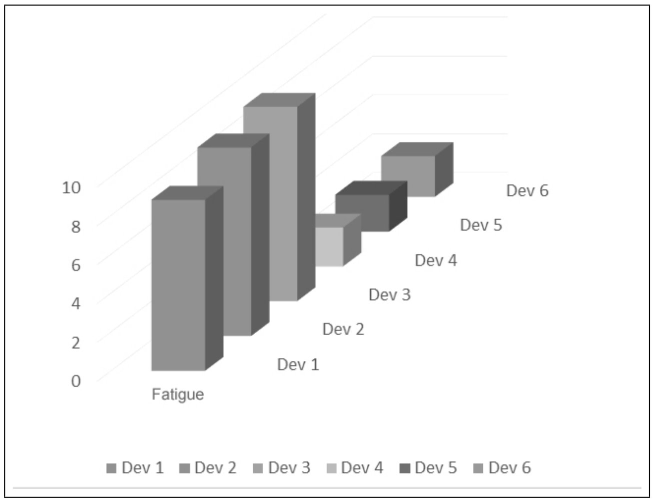


Figure 7. Analysis of fatigue developers.

TIME x FATIGUE

Analyzing fatigue and time, side by side, we can see a huge disparity between the teams. It can be seen quite clearly that the Beta Team, represented by the participants 4, 5 and 6, showed less wear than the Alfa team. It was a result we expected, since the Beta team was the one who was using Mock Object in the experiment. This analysis can be seen in detail in Fig. 8.

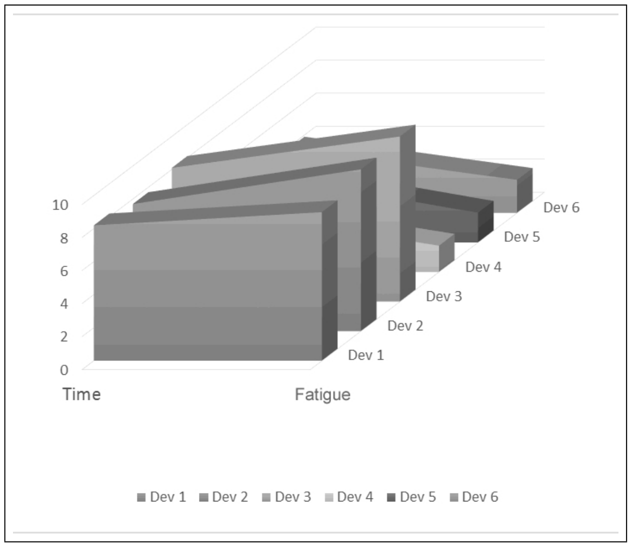


Figure 8. Relationship between time and fatigue.

V. Conclusion

In this paper, we present the use of Mock Objects with TDD, a few quotes on the subject and an experiment in the laboratory which demonstrates the practice of this activity. The experiment had two multidisciplinary teams that conducted a study, consisting of the preparation of a report considered quite complex, using the IReport and Eclipse IDE tools, the use of timing and direct observation of the leader.

For the experiment, seven items were submitted for evaluation: elaboration time, fatigue presented by the developers, perception of difficulty reported by the participants, number of hits, visual quality of the work, correct implementation of business rules and the number of interruptions per participant. Each of the items were measured empirically by using a scale of 2 to 10. For the analysis of results, we infer that the adoption of Mock Objects can be a good strategy when work requires great effort from developers. However, one more refined analysis of the situation may provide better subsidies for decision-making.

There is strong evidence for the growth of TDD practice in future. In recent years, the academic community has been conducting various experiments to show empirically that TDD helps the software development process. Some of these studies are done by professors well known in the community, such as prof. Laurie Williams (North Carolina State University) [18] and Prof. David Janzen (California Polytechnic State University) [19].

The Brazilian Army collaborated with researchers providing the means to carry out this work, bringing an important contribution to the science of Software Engineering. Based on the studies in this field, we believe that the TDD process will continue to be of interest to researchers.

References

[1] E. N. C. Borges, “Benefits of test driven development,” Universit of Rio Grande do Sul/Computer Institute, 2006.

[2] A. Leon and A. S. Koch, Agile software development evaluating the methods for your organization. Artech House, Inc., 2004.

[3] K. Beck et al., “Manifesto for agile software development,” 2001.

[4] K. Beck, “Embracing change with extreme programming,” Computer, vol. 32, no. 10, 1999, pp. 70–77.

[5] ——, “Extreme programming explained: embrace change,” 2000.

[6] H. Baumeister and M. Wirsing, “Applying test-first programming and iterative development in building an e-business application,” in International Conference on Advances in Infrastructure for e-Business, e-Education, e-Science, and e-Medicine on the Internet, SSGRR 2002, LAquila, Italy, 2002.

[7] W. Marrero and A. Settle, “Testing first: emphasizing testing in early programming courses,” in ACM SIGCSE Bulletin, vol. 37, no. 3. ACM, 2005, pp. 4–8.

[8] M. Fowler, “Refactoring: Improving the design of existing code,” in 11th European Conference. Jyväskylä, Finland, 1997.

[9] K. Beck, Test-driven development: by example. Addison-Wesley Professional, 2003.

[10] S. Yenduri and L. A. Perkins, “Impact of using test-driven development: A case study.” in Software Engineering Research and Practice, 2006, pp. 126–129.

[11] T. Mackinnon, S. Freeman, and P. Craig, “Endo-testing: unit testing with mock objects,” Extreme programming examined, 2001, pp. 287–301.

[12] B. Stroustrup and D. Lenkov, “Run-time type identification for c++(revised yet again),” document X3J16/92-0121, American National Standards Institute Accredited Standards Committee, Tech. Rep., 1992.

[13] M. Fowler, “Mocks arent stubs,” 2007.

[14] K. Lieberherr, I. Holland, and A. Riel, “Object-oriented programming: An objective sense of style,” in ACM SIGPLAN Notices, vol. 23, no. 11. ACM, 1988, pp. 323–334.

[15] S. Freeman, T. Mackinnon, N. Pryce et al., “Mock roles, objects,” in Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications. ACM, 2004, pp. 236–246.

[16] M. Brown and E. Tapolcsanyi, “Mock object patterns,” in The 10th Conference on Pattern Languages of Programs, Monticello, USA, 2003.

[17] G. C. Low and D. R. Jeffery, “Function points in the estimation and evaluation of the software process,” Software Engineering, IEEE Transactions on, vol. 16, no. 1, 1990, pp. 64–71.

[18] W. Laurie, “Laurie Williams - profile,” http://collaboration.csc.ncsu.edu/laurie/, 2016, [Online; accessed 08-August-2016].

[19] D. Janzen, “David Janzen - profile,” http://users.csc.calpoly.edu/~djanzen/, 2016, [Online; accessed 08-August-2016].