Software testing education through a collaborative virtual approach

Conference Paper · January 2018

DOI: 10.1007/978-3-319-69341-5_21

3 authors:

Juan Pablo Ucán Pech
Universidad Autónoma de Yucatán
38 PUBLICATIONS 99 CITATIONS
SEE PROFILE

Raúl Antonio Aguilar Vera
Universidad Autónoma de Yucatán
78 PUBLICATIONS 209 CITATIONS
SEE PROFILE

Omar S. Gómez
Escuela Superior Politécnica de Chimborazo
99 PUBLICATIONS 654 CITATIONS
SEE PROFILE

Software testing education through a collaborative virtual approach

Juan P. Ucán Pech[1], Raúl A. Aguilar Vera[1], Omar S. Gómez[2],

[1]Universidad Autónoma de Yucatán, Mérida 97000, Yucatán, México,
[2]Escuela Superior Politécnica de Chimborazo, Riobamba 060106, Chimborazo, Ecuador,
{juan.ucan, avera}@correo.uady.mx, ogomez@espoch.edu.ec

Abstract. The use of collaborative virtual environments facilitates communication, coordination and cooperation within a group of people. In the field of Information and Communications Technology (ICT), collaborative virtual environments have been developed to support the learning of programming. With the aim of fostering software testing education by alternative means, we present findings from the use of a collaborative virtual environment used for software testing training. Particularly, we report an empirical study (controlled experiment) which assesses the effectiveness of detecting defects in instrumented programs with and without the support of an collaborative virtual environment (CVE). The CVE was used as part of a programming course at under-graduate level. The results suggest an equivalent effectiveness in defect detection for both types of participants, those who used the CVE and those who worked in a traditional manner. It was observed that with regard to effectiveness, for this type of task (the detection of defects), working virtually through a CVE versus working in a tradition manner (at the same time and in the same place) yields equivalent results. When collaborative work cannot be done in a traditional way, the use a CVE is an alternative approach equally effective for collaborating and learning on software testing.

Keywords: Software Testing Education, Computer-supported Collaborative Learning, Computer-supported Cooperative Work, Experimentation in Software Engineering.

1 Introduction

Education in software testing is still an open issue to address. Although most Computer Science programs offer at least one course on software engineering (SE), there is evidence to show that formal software testing training is deficient. For example, Chan et al. [1] found that less than 20% of software testing teams in Hong Kong had received formal software testing training at university. It is common practice for CS programs to briefly touch upon the topic of software testing as one of the issues addressed in SE courses [2]. According to Wong et al. [3], some US universities offer around 30 software testing courses as part of undergraduate CS programs but no one of the CS programs in the US has as prerequisite a software testing course. Recently Gómez et al. [4] found a relationship between the level of exposure to CS knowledge and the quality of test cases generated by different types of students, Gómez et al. [4] found that students who have had less exposure to CS knowledge generate poorer quality test cases.

With the aim of encourage by alternative means the adoption of software testing education in CS programs, we developed a collaborative virtual environment (CVE) which can be used as a tool support for training in this area.

The rest of this document is organized as following: Section 2 presents an overview of related work. Section 3 shows the main features of the CVE developed. Section 4 presents the experiment conducted. Section 5 presents de results. Section 6 exposes the findings. Finally the conclusions are presented in Section 7.

2 Related Work

Learning through Collaborative Virtual Environments (CVEs) has been an area of increasing interest within the community of Information and Communications Technology (ICT), particularly for those researchers interested in the development of educational software based on the paradigm of Computer-Supported Collaborative Learning (CSCL) [5]. The most important advantage of these applications lies in the communication, coordination and cooperation achieved through teamwork. The aforementioned type of interaction through virtual environments derives from an area of research known as computer-supported cooperative work (CSCW) [6].

CVEs can be understood as the result of the convergence of research in the fields of virtual environments (VEs) and computer-supported cooperative work (CSCW). Such software systems are developed with the purpose of promoting communication, coordination, cooperation and, in the ideal case, collaboration during teamwork [7]. Table 1 describes some characteristics of some existing CVEs.

Table 1. Characteristics of CVEs related to this study.

CVE	Application Environment	Characteristics
HabiPro [8]	Desktop	Text-based virtual environment, designed to solve programming problems in small work groups, developed in Java.
Jazz [9]	Desktop	Research project from IBM that focused on the incorporation of collaborative capabilities into an application development environment, developed in Java using a plugin that is incorporated into Eclipse.
SÁBATO [10]	Web	Tool used in teaching algorithms and programming in engineering, oriented more toward a teaching environment.
EclipseGavab [11]	Desktop	Tool with collaborative characteristics that enables the implementation of project-based learning, developed in Java using a plugin that is incorporated into Eclipse, with support for Pascal, C and Java.
VPL [12]	Web	Tool that is integrated with Moodle, it is not mentioned the programming language which was developed.

The CVEs shown in Table 1 were developed individually, they provide experience with and the use of a software tool, and most are desktop applications. With regard to their evaluation, HabiPro [8] included an experiment in which students were required to solve problems. With regard to Jazz [9], its various benefits were discussed from a contextual collaborative perspective. SÁBATO [10] was used as part of an algorithmic and programming undergraduate course where students solved algorithmic related problems through the use of this CVE. EclipseGavab [11] was compared with other tools, such as Turbo Pascal. Currently it is a project from the Universidad Rey Juan Carlos located in Spain. VPL [12] was used in an algorithms and programming course, and at the end of the course, the results of using and not using the tool were evaluated. In general, it is observed that these proposals are used for coding activities under a collaborative approach and not for testing purpose.

3 Characteristics of the developed CVE

A general structure of the CVE developed [13] and used in this study is described below. We incorporated expert system capabilities into the CVE to support students in detecting defects in a series of instrumented programs. This architecture can be understood as a layered architecture [14] for a system whose primary users are students. The students can use the CVE to view pieces of code from programs implemented under a selected programming paradigm (for example, the structured paradigm), and they are directed to identify defects in a series of instrumented programs. The students can perform this activity in working groups, in consultation with the professor or with an expert system (ES) integrated into the CVE that supports the identification of common defect types. At the moment, the developed CVE is intended for code inspection, we are planning to incorporate other kinds of testing methods such as black-box and white-box into the CVE.

Because of its nature and intended platform, the remainder of the design of the CVE was performed based on web engineering paradigm. For the modeling of the CVE, the UML-based Web Engineering (UWE) method was implemented; this method involves building separate models for the analysis of requirements, content, navigation, presentation and process [15]. An example of the user interface of this CVE is shown in Fig. 1.

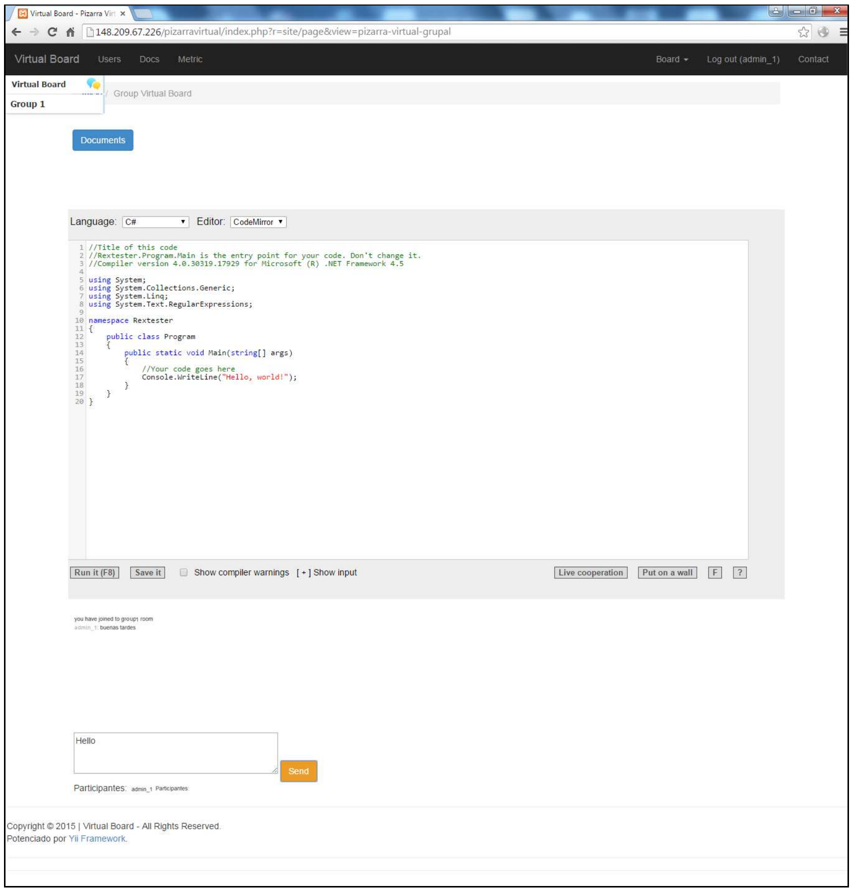
In general, the user (student) can view instrumented codes with which he or she wishes to work. In the central part of the interface, the user can access a virtual whiteboard where he or she can share pieces of code with other students. The interface also includes a chat area for collaboration among other colleagues.

4. Context of the experiment

To evaluate the effectiveness of defect detection using the developed CVE, a controlled experiment was conducted using undergraduate students as participants, who worked with CVE (assisted by CVE) and without the CVE (in the same place and at the same time, this being the traditional way of working in software testing).

for the task of detecting defects in a series of instrumented programs. For this evaluation, the following research question was considered.

RQ1: Is the effectiveness in the detection of defects affected by the use of a CVE?



```
//Title of this code
//Rextester.Program.Main is the entry point for your code. Don't change it.
//Compiler version 4.0.30319.17929 for Microsoft (R) .NET Framework 4.5

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text.RegularExpressions;

namespace Rextester
{
    public class Program
    {
        public static void Main(string[] args)
        {
            //Your code goes here
            Console.WriteLine("Hello, world!");
        }
    }
}
```

you have joined to group1 room
admin_1: buenas tardes

Hello

Participantes: admin_1 Participantes:

Fig. 1. Screenshot of the CVE user interface.

The above research question was translated into the following working hypotheses:

H_0. The effectiveness, measured as the percentage of observed defects, is equal for those participants who use the CVE and for those who work without the CVE (in a traditional manner).

H_1. The effectiveness, measured as the percentage of observed defects, is different for those participants who use the CVE and for those who work without the CVE (in a traditional manner).

The hypotheses presented above are tested against the measurements collected during the execution of the study. In essence, these measurements are associated with two treatment groups: the participants who worked with the CVE and those who worked without the CVE (in a traditional manner).

A crossover design was used in this study to ensure a greater number of measurements (observations) [16]. In a crossover design, the experimental units (in this case, the groups of participants, which comprised either two or three students) each receive two or more treatments, and the order of application of those treatments is determined by the design structure [16]. In this particular study, a 2 × 2 crossover design was used, involving two treatments (the use of the CVE and in a tradition manner) in two different sessions. With respect of this structure, in the first session, half of the groups of participants work with the CVE, while the other half of the groups work without the CVE (in a traditional manner); in the second session, each group of participants receives the treatment it did not receive in the first.

One shortcoming of this type of design is that the effects of certain treatments can extend beyond the period of application (carryover effect). A possible strategy for mitigating this drawback is to avoid applying the treatments in consecutive periods; in our case, a four-day break between the two sessions was planned. Table 2 summarizes the structure of the applied experimental design.

Table 2. 2 x 2 crossover design used in this study.

	Sequence 1	**Sequence 2**
Session 1 (Program 1)	With CVE	Traditional Manner
Session 2 (Program 2)	Traditional Manner	With CVE

4.1 Execution of the study

Before performing the study, the selected participants received a two-hour session of training on the use and operation of the CVE and on the types of defects that are common in programming. We also explained to participants a basic software inspection process. It was generally explained to the participants that the training session and the remaining sessions were part of a study on the CVE; verbal consent was received from the students to participate in this study.

The experiment here reported took place at the end of 2014 at the Faculty of Mathematics of the Autonomous University of Yucatan (UADY). A total of 46 students participated in this study. The participants were at the end of the first semester of their first year of the Software Engineering undergraduate program, all the participants have knowledge about key aspects of the C programming language.

Concerning the allocation of the experimental units (groups of participants) to the treatments, teams of two or three participants were formed, and these teams (18 total experimental units) were randomly allotted to one of the two sequences of treatments (with CVE - traditional manner or traditional manner - with CVE). For the recording of information regarding the observed defects, the participants used an automated coding instrument, that is, a form integrated into the CVE in which they registered the observed defects.

In two separate sessions of two hours each, the study was conducted in three rooms of the computer center of the Faculty of Mathematics at the UADY. Both the first and second sessions of the study began at the planned time. The students were given the printed specifications and the code of the program (instrumented with defects) to be inspected. Program codes were written in the C programming language. Moreover, the program source codes were available in the CVE. Each program was instrumented with seven defects. The knowledge source for the defects was obtained from [17], and the classification of defects was used from [18]. Each program had an approximate length of 150 LOC.

In the first session, one group of participant teams (teams with id 1, 2, 3, 8, 9, 13, 16 and 19) worked on the detection of the defects present in program 1 using the CVE, while the other group (teams with id 4, 5, 6, 7, 10, 12, 15 and 17) worked without using the CVE (traditional manner). In the second session, teams with id 1, 2, 3, 8, 9, 13, 16 and 19 worked on detecting the defects present in program 2 without using the CVE (traditional manner), while teams with id 4, 5, 6, 7, 10, 12, 15 and 17 worked using the CVE, that is, the treatments were allocated in the opposite manner. Participants assigned to the CVE worked each one in one computer, interacting through the CVE.

Regarding the specification of instrumented programs, program 1 (matrix.c) takes as input a square integer matrix of order *m* and performs four types of calculations on it; the following defects were introduced into this program according to the classification in [18]: two cosmetic, two control, one initialization, one omission and one computation. Program 2 (students.c) specification receives and performs basic functions over the scores of 10 students. Some of these functions are: Finding a student, update a score, compute the average of the scores, among other functions. This program contains the following defects: two computing, one cosmetic, one interface, one initialization, one data and one omission.

In each session, once the participants had completed their inspection activities of the assigned code, in addition to recording their results on printed sheets, they were also asked to register their defects observed in the CVE (for those who worked with the CVE).

5. Analysis and results

This section presents both the descriptive statistical analysis of the measurements collected and the inferential statistical analysis. Table 3 shows the effectiveness means (in terms of percentage of observed defects), standard deviations, minimum and maximum values for both treatments (with CVE and traditional manner). As shown in Table 3 the effectiveness in both treatments was similar.

Table 3. Means and standard deviations for the treatments.

Treatment	Effectiveness (%)	SD	Min.	Max
With CVE	43.45	20.03	14.28	95.28
Traditional Manner	45.33	17.78	14.28	75.71

Table 4 shows the effectiveness means, standard deviations, minimum and maximum values with respect to the instrumented programs used in the two sessions of the study. As shown in Table 4, on average, the participants seem to identify more defects in program 2.

Table 4. Means and standard deviations per session (program).

Period	Effectiveness (%)	SD	Min.	Max
Session 1 (Program 1)	37.88	16.55	14.28	75.71
Session 2 (Program 2)	50.90	18.84	14.28	95.28

Table 5 shows the effectiveness means, standard deviations, minimum and maximum values with respect to the two sequences used in this experimental design. Given the characteristics of this experimental design, there is a possibility that a carryover effect could arise and confound the real effects of the treatments. As observed in Table 5, however, the means for the two sequences appear to exhibit no substantial differences; thus, the absence of carryover effects in the treatments can be inferred.

Table 5. Means with respect to the sequences of application of the treatments.

Sequence	Effectiveness (%)	SD	Min.	Max.
Whit CVE-ad hoc	41.07	16.82	14.28	71.42
Ad hoc-CVE	47.71	20.32	14.28	95.28

Once the measurements are collected, it is possible to test the hypotheses posed above. The statistical model associated with the crossover design used is described in Equation (1).

$$y_{ijk}=\mu+\alpha_i+b_{ij}+\gamma_k+\tau_d+\lambda_c+\varepsilon_{ijk} \quad , \tag{1}$$

where μ is the general mean; α_i is the effect of the *i*th treatment sequence; b_{ij} is the random effect with variance σ_b^2 for the *j*th participant of the *i*th treatment sequence, γ_k is the period effect (session); τ_d is the direct effect of the treatment; λ_c is the carryover effect of the treatment administered in period *k*-1 of sequence group *i*; and ε_{ijk} is the independent random error, with an mean of zero and a variance of σ^2 for the participant in period *k*. In this model, the analysis of variance (ANOVA) is used to evaluate the various components of the model.

Table 6 presents the results of the ANOVA with respect to effectiveness. Effectiveness is measured as the percentage of observed defects for all the groups of participants. As observed in Table 6, only the program component (period or session effect) yielded a significant difference at an alpha level of 0.05.

The calculation of the separation measure between the treatment and the carryover effect produced a value close to 30% (29.2893%). This measure indicates the degree of orthogonality between the two effects. When a carryover effect is present in this type of design, this orthogonality is lost. A low percentage value of the separation measure suggests that there may be problems in the interpretation of the results [16]. For example, a significant difference between the treatments may be deduced when in reality, the difference is caused by the presence of a carryover effect. A 2 x 2 crossover design, such as the one used in this study, is prone to manifesting low percentages in the separation measurement. This design is recommended when the absence of carryover effects is inferred. Because no significant carryover effect was observed, the results of the ANOVA can be considered reliable. The lack of a carryover effect was reinforced by the similar effectiveness rates observed in both sequences (see Table 5).

Table 6. Analysis of variance with respect to effectiveness.

Source of Variation	Partial SS	df	MS	F	Prob > F
Treatment effect	290.22	1	290.22	0.9	0.3517
Session effect (program)	1546.17	1	1546.17	4.78	0.0373
Carryover effect	353.02	1	353.02	1.09	0.3051
Residuals	9058.20	28	323.51		

Finally, the ANOVA should satisfy the assumption of normality for reliable inferences to be obtained. We used the Shapiro-Wilk statistical test [19] in which the null hypothesis is that a given sample of data originates from a normally distributed population. Table 7 shows the results of this test with respect to the effectiveness.

Table 7. Results of the Shapiro-Wilk test.

Aspect	n	W	V	Z	Prob > z
Effectiveness	32	0.9652	1.160	0.307	0.37928

As observed in Table 7, the null hypothesis is accepted in favor of the normality of the data, that is, the measurements of each metric (aspect) are concluded to be drawn from a normal distribution; thus, our results can be considered reliable.

6. Discussion

According to the findings reported here, the results suggest an equivalence in the effectiveness of defect detection for both the groups of participants who worked in a virtual collaborative manner (with the CVE) and those who worked in a traditional manner (**RQ1**). For this type of activity (detection of defects), the obtained results suggest that the use of the CVE is equally as effective as working collaboratively following a traditional approach (ad hoc). However the kind of program used for the collaborative testing may yield different percentages of observed defects, each program to be tested may have different characteristics that as we observed may affect on the effectiveness.

Empirical studies may be exposed to threats to validity, next we discuss how we tackled them. With regard to threats to internal validity, the sessions were conducted as planned, without any disruption (history effect). Because a rest period was planned between the administration of the different treatments, a minimal effect of maturation can be assumed. The selection bias effect was reduced by randomization mechanism used in the experimental design. Although in both sessions only two experimental units (groups of participants) failed to complete the sequence of treatments (they only participated in one of the session), the presence of a low mortality effect can be considered as irrelevant. During the sessions of the study, interest in performing the activities was observed on the part of the participants; thus, a minimal effect of demoralization can be assumed. With regard to threats to the external validity of the study, the results presented here are partially generalizable to students with characteristics similar to those of the students who participated in the reported study. However, further replications of this study will alleviate this threat. We plan to conduct further replications with the aim of serving in the process of generation and consolidation of knowledge on this topic [20].

7. Conclusions

With the aim of foster software testing education by alternative means, a CVE was developed. In this work we presented the results of the assessment of such as CVE. Particularly the experiment focused on the assessment of the effectiveness of defect detection of two instrumented programs with and without the use of a CVE. Results suggest equivalent levels of effectiveness for those participants working with the CVE and for those working in a traditional way.

Some advantages of the use of the CVE are the following: 1) it is not necessary for the students to be located in the same physical space; 2) the CVE offers a virtual whiteboard feature, a component that includes an integrated IDE to write, inspect and compile pieces of code in a collaborative programming language, which is lacking in the traditional approach to learning programming and testing; and 3) the intelligent component (provided through an expert system in this proposal), which is a feature developed to facilitate the interaction between students during programming and testing training activities. Finally, the CVE assessed offers an effective means of assisting teams of students in the identification of defects in a series of instrumented programs.

As concluding remark, when collaborative work cannot be done in a traditional way, the use a CVE is an alternative approach equally effective, in this case for collaborative virtual working on software testing activities.

Acknowledgments

Authors from the Autonomous University of Yucatan (UADY) acknowledge the financial support received from the Mexican Secretary of Public Education (SEP) through project P/PFC-2016-31MSU0098J-13. We also acknowledge the support provided by UADY students participating in this study.

References

1. Chan, F., Tse, T.: Software testing education and training in hong kong. In Quality Software, 2005. (QSIC 2005). Fifth International Conference on, pp. 313-316 (2005)
2. Wong, W.E.: Improving the state of undergraduate software testing education. In 2012 ASEE Annual Conference, San Antonio, Texas (2012)
3. Wong, W., Bertolino A., Debroy V., Mathur A., Offutt J., Vouk, M.: Teaching software testing: Experiences, lessons learned and the path forward. In Software Engineering Education and Training (CSEET), 2011 24th IEEE-CS Conference on, pp. 530-534 (2011)
4. Gómez, O.S., Vegas S., Juristo, N.: Impact of CS programs on the quality of test cases generation: An empirical study. In 38th International Conference on Software Engineering ICSE’2016 Austin, Texas USA, pp. 374-383 (2016)
5. Koschmann, T.: Dewey's contribution to the foundations of CSCL research, Proc. of the Conference on Computer Support for Collaborative Learning: Foundations for a CSCL Community (CSCL '02), Gerry Stahl (Ed.). Int. Soc. of the Learning Sci., pp. 17-22 (2002)
6. Wilson, P.: Computer supported cooperative work: An introduction, Springer (1991)
7. Aguilar, R., Gómez, G., Moo, F., Uicab, R., Tuyub, R.: A Model Based on Intelligent Virtual Environments to Assist the Learning of Programming and Mathematics, International Conference on Education and New Learning Technologies (Edulearn 2010), Barcelona Spain, L. Gómez, D. Martí and I. Cande (Eds.), pp. 2564-2571 (2010)
8. Vizcaino, A.: Enhancing Collaborative Learning Using a Simulated Student Agent. Ph.D. Thesis, Universidad de Castilla-La Mancha, Spain (2002)
9. Hupfer, S., Cheng, L.T., Ross, S., Patterson, J.: Introducing Collaboration into an Application Development Environment, Proceedings of the 2004 ACM conference on Computer supported cooperative work, pp. 21-24 (2004)
10. Jiménez, J., Pavony, M., Álvares, A.: Entorno de integración de PBL y CSCL para la enseñanza de algoritmos y programación en ingeniería. Revista Avances en Sistemas e Informática de la Universidad Nacional de Colombia 5, pp. 189-194 (2008)
11. Gallego, M., Gortázar, F.: EclipseGavab, un entorno de desarrollo para la docencia online de la programación. XV Jornadas de Enseñanza Universitaria de la Informática (JENUI XV), Barcelona, pp. 501-508 (2009)
12. Rodríguez-del-Pino, J.C., Rubio, E., Hernández, Z.: A Virtual Programming Lab for Moodle with automatic assessment and anti-plagiarism features. Internacional Conference on e-Learning, e-Business, Entreprise Information Systems & e-Government (2012)
13. Ucán-Pech, J.P.: Aprendizaje de la Programación Asistido con Entornos Virtuales Colaborativos Inteligentes. Ph.D. Thesis. Dirección de Posgrado e Investigación de la Universidad del Sur, Campus Mérida, México (2015)
14. Bass. L., Clements, P., Kazman, R.: Software Architecture in Practice (3rd Ed.), Addison-Wesley Professional (2012)
15. Rossi, G., Pastor, O., Schwabe, D., Olsina L.: Web Engineering: Modelling and Implementing Web Applications, Springer London (2007)
16. Kuehl, R.O.: Design of experiments: statistical principles of research design and analysis, Duxbury/Thomson Learning (2000)
17. Ko, A.J.: Asking and answering questions about the causes of software behavior, Ph.D. Thesis, Carnegie Mellon University, USA (2008)
18. Basili, V.R., Selby, R.W.: Comparing the effectiveness of software testing strategies, IEEE Transactions on Software Engineering 12, pp. 1278-1296 (1987)
19. Shapiro, S.S., Wilk, M.B.: An analysis of Variance Test for Normality (Complete Samples), Biometrika 52, pp. 591-611 (1965)
20. Juristo, N., Gómez, O.S.: Replication of software engineering experiments. In Bertrand Meyer and Martin Nordio (Eds.), Empirical Software Engineering and Verification, 7007 of Lecture Notes in Computer Science, Springer-Verlag Berlin, pp. 60-88 (2011)