Evaluating Software Testing Techniques and Tools

Tanja Vos, Beatriz Marín, Ignacio Panach, Arthur Baars, Claudia Ayala, and Xavier Franch

Centro de Mtodos de Produccin de Software (ProS)
Universidad Politécnica de Valencia
{tvos,bmarin,jpanach,abaars}@pros.upv.es
http://www.pros.upv.es

Universitat Politécnica de Catalunya
{cayala,franch}@essi.upc.edu
http://www.essi.upv.edu/~gessi/

Abstract. Case studies can help companies to evaluate the benefits of testing techniques and tools before their possible incorporation into the testing processes. Although general guidelines and organizational frameworks exist describing what a case study should consist of, no general methodological framework exists that can be instantiated to easily design case studies to evaluate different testing techniques. In this paper we define a first version of a general methodological framework for evaluating software testing techniques, that focusses on the evaluation of effectiveness and efficiency. Using this framework, (1) software testing practitioners can more easily define case studies through an instantiation of the framework, (2) results can be better compared since they are all executed according to a similar design, and (3) the gap in existing work on methodological evaluation frameworks will be narrowed.

Keywords: Case study, Software Testing Techniques, Methodological Framework, Evaluation

1 Introduction

There exists a real need in industry to have guidelines on what testing techniques to use for different testing objectives, and how usable these techniques are. In order to obtain such guidelines, more case studies need to be performed. However, there is a lack of a general methodological evaluation framework that can simplify the design of case studies for comparing software testing techniques and make the results more precise, reliable, and easy to compare.

In this paper, we address this issue by presenting a first version of a general methodological framework for evaluating software testing techniques and tools[1],

[1] We refer to techniques and tools to take into account the evaluation of those techniques that are applied semi-automatically.

which can be instantiated to define case studies that evaluate the effectiveness and efficiency of specific techniques and tools. The framework is general since it does not make any assumptions about the testing technique that is being evaluated nor about the subjects and the pilot projects. This framework has been systematically defined following the general guidelines to perform case studies defined by [11].

The remainder of the paper is organized as follows: Section 2 presents some relevant related work, Section 3 presents the general framework for the evaluation of testing techniques, and Section 4 presents our conclusions and future work.

2 Related work

Lots of work is available with guidelines, roadmaps or *organizational* steps for doing empirical research in software engineering. Some relevant works for controlled experiments are [13, 6], and for case studies are [7, 9, 8, 11, 5]. An overview of empirical research issues for software engineering has been presented in [12].

Specific for software testing, [10] describe a characterization scheme for experiments. The schema is similar to [1] but adapted to deal with evaluating testing techniques. This schema is divided in four parts: the goals and hypotheses that motivate the experiment; the plan for conducting the experiment; the procedures used during the experiment; and the results.

Do et al. [3, 2] define the SIR (Software Artifact Infrastructure Repository) (sir.unl.edu) infrastructure to support controlled experiments with software testing techniques. The main contribution of their work is a set of benchmark programs that can be used to evaluate testing techniques.

The work from Eldh et. al. [4] is most close to our work. They describe a straightforward framework for the comparison of the efficiency, effectiveness and applicability of different testing techniques based on fault injection or seeding. The steps of the framework are: prepare code samples with known faults (i.e. through fault injection or seeding); select a testing technique; perform the experiment and collect data; analyze the data; and repeat the experiment if necessary.

All work cited above present *organizational* frameworks and guidelines, i.e. the steps that must be carried out to desing and conduct the studies and confounding factors that should be minimized. However, with the exception of [4] that is restricted to fault injection, we did not find any work that specifies *how* to evaluate software testing techniques, how the research questions can be defined, what variables could be measured, what can be threats to validity, etc. This is precisely what our proposed methodological framework do.

3 A Methodological Evaluation Framework

Imagine company C wants to evaluate the testing technique or tool T to see if it is usable and worthwhile to incorporate this technique or tool into its testing processes. In order to design a case study for this company, an instantiation of the following sections should be defined.

3.1 Objective - What to achieve?

The general framework focuses on the measures of usability defined by ISO 9241-11 (ISO 1998): efficiency, effectiveness, and subjective satisfaction. In this paper, we only present the efficiency and effectiveness. Consequently, the research questions for each case study correspond to instantiations of:

RQ_1 What is the effectiveness (fault[2]-finding capabilities) of T when it is used in real testing environments?
RQ_2 What is the efficiency of T when it is used in real testing environments?

3.2 Cases or Treatments - What is studied/evaluated?

The case, treatment, or unit of analysis is the testing technique or tool T that is evaluated by means of the case study.

3.3 Subjects - Who apply the techniques/tools?

The subjects are workers of C. Subjects should be those people that normally use the testing techniques or tools that are being compared to T.

3.4 Objects - What are the pilot projects?

The System Under Test (SUT) used in the case study should be a system that is normally developed within C. This means that it should be a project that is typical of the organization (i.e. the way the software is developed, the way it is tested, languages used, etc.). Also, the available information about the selected system should be determined in order to perform the comparison with T. We distinguish 3 possibilities for comparing:

S_1 T and the traditional testing techniques can be executed during the case studies, s.t. the same variables/measures can be collected for both testing approaches in order to compare them.
S_2 We have access to previous versions, a sister project, or a company baseline that has the right information to compare with T.
S_3 If we cannot do S_1 nor S_2, but the SUT is open, faults can be injected (this possibility can be more difficult to compare efficiency).

3.5 Propositions or Concrete Hypothesis - What to know?

The research questions has been refined into propositions that can be evaluated through the variables that will be measured during the studies.

P_1 T is more effective in finding faults when it is applied to real-world systems compared to current practice in C.
P_2 T is more efficient in finding faults when it is applied to real-world systems compared to current practice in C.
P_3 The amount of resources needed to install, configure and learn T in order to use it for testing real-world systems is worthwhile within C.

[2] Fault: Incorrect code that results from a human mistake

3.6 Variables and Measures - Which data to collect?

Independent and dependent variables are the attributes that define the study setting. Manipulating the independent variables can spawn significant amounts of data, which can help the researchers to determine whether the propositions are verified or not. Dependent variables are defined as the results whose values vary predictably by manipulating the independent variables.

- Independent variables: Testing method T used, Industrial systems used in the case study, Software Engineers/Testers of C that will do the testing, Level of experience of testers.
- Dependent Variables: Effectiveness, Efficiency.

For a specific instantiation of this framework in a company, some variables might not be applicable. In this case, these variables must not be included in the study and the section *how to interpret the data* should be revised. The following measures must be collected to evaluate the dependent variables:

1. Measuring effectiveness
   (a) Number of test cases designed or generated.
   (b) Number of invalid test cases are generated.
   (c) Number of repeated test cases are generated.
   (d) Number of failures[3] observed.
   (e) Number of faults found.
   (f) Number of false positives
   (g) Number of false negatives
   (h) Type and cause of the faults that were found.
   (i) Estimation (or measurement if it is possible) of coverage reached.
2. Measuring efficiency
   (a) Time needed to learn the testing method T.
   (b) Time needed to design or generate the test cases.
   (c) Time needed to set up the testing infrastructure specific to T (install, configure, develop test drivers, etc.). If software is to be developed or other mayor configuration/installation efforts are needed, it might be a good idea to maintain working diaries.
   (d) Time needed to test the system and observe failures (i.e. planning, implementation and execution) in hours.
   (e) Time needed to identify fault types and causes for each observed failure.

3.7 Protocol - How to execute the study and collect the data?

Company C should do the tests and measure the variables in a real environment. The procedure to execute each case study consist of the following steps:

1. If (S_1) then execute tests how you would do normally and collect the data.
2. If (S_2) then collect the data from sister project or company base line.

[3] Failure: Incorrect behaviour of software that the user can observe.

3. If (S_3) then inject faults into the system.
4. Receive instructions or a short training course on how to use T.
5. If T is a tool then install, configure, and collect data.
6. If T is a test case design/generation technique or tool then design or generate the test cases and collect data.
7. Execute tests with/from T and collect data.

For each failure found during steps 1 and 7: fill a bug report (be as detailed as possible), keep track of the time spent for finding the failure, and keep track of time for filling the bug report and identifying each fault.

3.8 Data Analysis - How to interpret the findings?

In order to evaluate P1, the measured variables are used to find out whether:

- More faults have been found with T than with the techniques used currently in the industrial setting (variables 1d, 1e)
- More critical faults are found with T than with the techniques used currently in the industrial setting (variable 1h)
- More coverage is reached using T. (variable 1i)
- Less false positives are detected (variable 1f)
- Less false negatives are detected (variable 1g)

In order to evaluate P2, the measured variables are used to find out whether:

- The same faults are found in the same time (variables 1e, 2b, 2d, 2e)
- More faults are found in the same time or less time (variables 1e, 2b, 2d, 2e)
- If finding faults using T takes more time, then the extra time it takes to use T is worth it considering the criticality of the faults found (variable 1h)

In order to evaluate P3, variables 2a and 2c are used.

3.9 Threats to Validity of the Studies Performed

Using [13], the following threats may jeopardize the validity of the case studies. Construct Validity threats might be: hypothesis guessing, evaluation apprehension and experiment expectancies. Internal Validity threats could be maturation, history related, instrumentation, and the observer effect. External Validity threats could be related to interaction of selection and treatment. Conclusion Validity threats might be random heterogeneity of subjects.

4 Conclusions

In this paper, a first version of a methodological framework to evaluate software testing techniques has been presented. This framework will enable software testing practitioners to more easily define case studies by instantiating the framework, while ensuring that the many guidelines and checklists for doing empirical

work have been met. In addition, since case studies will be executed according to a similar design, it will be more easy to compare the results obtained, and hence, specify general statements about testing techniques and tools evaluated.

As future work, we plan to improve the framework incorporating the evaluation of the subjective satisfaction of the users of testing techniques, and the feedback obtained by other researchers in national and international forums.

Acknowledgments. This work was funded by the MICINN project PROS-REQ (TIN2010-19130-C02-01, TIN2010-19130-C02-02) and the EU funded project FITTEST (ICT257574, 2010-2013).

References

1. Basili, V.R., Selby, R.W., Hutchens, D.H.: Experimentation in software engineering. IEEE Trans. Softw. Eng. 12, 733–743 (July 1986), http://portal.acm.org/citation.cfm?id=9775.9777
2. Do, H., Elbaum, S., Rothermel, G.: Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact. Empirical Softw. Engg. 10(4), 405–435 (2005)
3. Do, H., Rothermel, G., Elbaum, S.: Infrastructure support for controlled experimentation with software testing and regression testing techniques. In: Proc. Int. Symp. On Empirical Software Engineering, ISESE '04 (2004)
4. Eldh, S., Hansson, H., Punnekkat, S., Pettersson, A., Sundmark, D.: A framework for comparing efficiency, effectiveness and applicability of software testing techniques. TAIC part pp. 159–170 (2006)
5. Host, M., Runeson, P.: Checklists for software engineering case study research. In: Proceedings of the First International Symposium on Empirical Software Engineering and Measurement. pp. 479–481. ESEM '07, IEEE, Washington (2007)
6. Kitchenham, B., Linkman, S., Law, D.: Desmet: a methodology for evaluating software engineering methods and tools. Computing Control Engineering Journal 8(3), 120 –126 (Jun 1997)
7. Kitchenham, B., Pickard, L., Pfleeger, S.: Case studies for method and tool evaluation. Software, IEEE 12(4), 52 –62 (Jul 1995)
8. Kitchenham, B.A., Pickard, L.M.: Evaluating software eng. methods and tools part 10: designing and running a quantitative case study. SIGSOFT Softw. Eng. Notes 23, 20–22 (May 1998), http://doi.acm.org/10.1145/279437.279445
9. Kitchenham, B.A., Pickard, L.M.: Evaluating software engineering methods and tools: part 9: quantitative case study methodology. SIGSOFT Softw. Eng. Notes 23, 24–26 (January 1998), http://doi.acm.org/10.1145/272263.272268
10. Lott, C.M., Rombach, H.D.: Repeatable software engineering experiments for comparing defect-detection techniques. Empirical Software Engineering 1, 241–277 (1996), http://dx.doi.org/10.1007/BF00127447, 10.1007/BF00127447
11. Runeson, P., Höst, M.: Guidelines for conducting and reporting case study research in software engineering. Empirical Softw. Engg. 14(2), 131–164 (2009)
12. Shull, F., Singer, J., Sjøberg, D.I.: Guide to Advanced Empirical Software Engineering. Springer-Verlag New York, Inc., Secaucus, NJ, USA (2007)
13. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: Experimentation in software engineering: an introduction. Kluwer Academic Publishers, Norwell, MA, USA (2000)