2012 12th International Conference on Quality Software

A Methodological Framework for Evaluating Software Testing Techniques and Tools

Tanja E.J. Vos*, Beatriz Marín†, Maria Jose Escalona‡, Alessandro Marchetto§
*Centro de Métodos de Producción de Software (ProS), Universidad Politécnica de Valencia, Valencia, Spain
Email: tvos@pros.upv.es
† Universidad Diego Portales, Manuel Rodriguez Sur 415, Santiago, Chile
Email: bmarin@udp.cl
‡ Ingeniera Web y Testing Temprano (IWT2), Universidad de Sevilla, Sevilla, Spain
Email: mjescalona@us.es
§ FBK-irst, Trento, Italy
Email: marchetto@fbk.eu

***Abstract*—There exists a real need in industry to have guidelines on what testing techniques use for different testing objectives, and how usable (effective, efficient, satisfactory) these techniques are. Up to date, these guidelines do not exist. Such guidelines could be obtained by doing secondary studies on a body of evidence consisting of case studies evaluating and comparing testing techniques and tools. However, such a body of evidence is also lacking. In this paper, we will make a first step towards creating such body of evidence by defining a general methodological evaluation framework that can simplify the design of case studies for comparing software testing tools, and make the results more precise, reliable, and easy to compare. 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, (3) the gap in existing work on methodological evaluation frameworks will be narrowed, and (4) a body of evidence will be initiated. By means of validating the framework, we will present successful applications of this methodological framework to various case studies for evaluating testing tools in an industrial environment with real objects and real subjects.**

***Keywords*-Case study, Software testing techniques, Evaluation, Methodological framework.**

I. Introduction

Software testing practitioners need to make informed decisions about which techniques to use in a specific situation and estimate the time and effort that is needed to apply them. However, up to date, there do not exist clear guidelines for this. One of the principle reasons for this is that not enough empirical studies have been performed that can serve as documented experiences for secondary Evidence-Based Software Engineering (EBSE) [1] studies.

Thus more studies that evaluate and compare testing techniques and tools are needed [2]–[4]. However, to make sure that the resulting body of evidence can yield the right guidelines, the evaluative case studies should:

- involve realistic systems and subjects, and not toy-programs and students as is the case in most current work [2], [5].
- be done with thoroughness to ensure that any benefit identified during the evaluation study is clearly derived from the testing technique studied
- ensure that different studies can be compared

Although this type of research is time-consuming, expensive and difficult, it is fundamental since claims made by analytical advocacy are insupportable [6].

Unfortunately companies are often reluctant to participate in case studies. Some are unwilling to try new, perhaps unproven approaches. Others are concerned that they might reveal critical faults, poor metrics or performance. Yet others are unwilling to allow researchers in for fear of losing proprietary information, or that they may slow down the team, or simply do not know how to do it. Regardless the reasons, barriers must be overcome in order to advance [7] and create the needed body of evidence.

In this paper we propose a general methodological framework to reduce some of the entry barriers for conducting case studies. The framework will simplify the design of

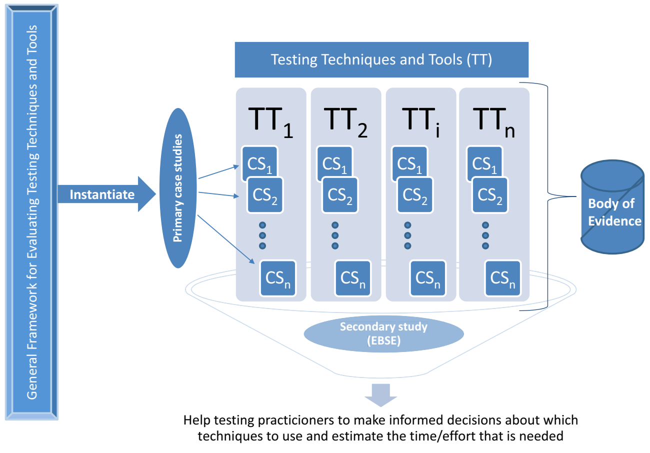
General Framework for Evaluating Testing Techniques and Tools
Instantiate
Primary case studies
Testing Techniques and Tools (TT)
TT1
TT2
TTi
TTn
CS1
CS2
CSn
Body of Evidence
Secondary study (EBSE)
Help testing practicioners to make informed decisions about which techniques to use and estimate the time/effort that is needed

Figure 1. Creating a body of evidence

case studies for comparing software testing techniques while ensuring that the many guidelines and check-list for doing empirical work have been met. Moreover, if case studies are all executed according to similar design (i.e. through instantiation of the framework), it will be possible to replicate studies and the created evidence will be easier compared and hence effectively aggregated in secondary studies (see Figure 1).

The framework we describe has evolved throughout the past years by doing case studies to evaluate testing techniques. The need to have a framework as described in this paper emerged some years ago during the execution of the EU funded project EvoTest (IST-33472, 2007-2009, [8]) and continued emerging during the EU funded project FITTEST (ICT-257574, 2010-2013, [9]). Both these are projects whose objectives are the development of testing tools that somewhere in the project need to be evaluated within industrial environments. Searching in the existing literature to find a framework that could be applied in our situation, did not result in anything that exactly fit our need: a *methodological* framework that is specific enough for the evaluation of software testing techniques and general enough and not make any assumptions about the testing technique that is being evaluated nor about the subjects and the pilot projects. We needed a framework that can be instantiated for any type of treatment, subject and object and simplifies the design of evaluative studies by suggesting relevant questions and measures. Since such a framework did not exist, we defined our own making sure that the guidelines and checklist that can be found in the literature are satisfied. Up to date we have successfully used the framework for various case studies during EvoTest and during FITTEST.

The remainder of the paper is organized as follows: Section II presents related work, Section III presents terminology, Section IV presents the general framework, Section V describes application of the framework and Section VI concludes.

II. Related work

Most existing work present *organizational* frameworks and guidelines, i.e. lists of the steps that must be carried, warnings that the studies should be carefully designed and that confounding factors should be minimized. With the exception of [10] which is restricted to fault injection, we do not know of any work that specifies *how* to evaluate software testing techniques, how the research questions can be defined, what variables could be measured, what the specific threats to validity can be, etc.

Lott and Rombach [11] describe a characterization schema for software testing experiments. The schema is similar to the general scheme in [12] 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. This schema helps users to organize and evaluate the design of an experiment. However, it offers no concrete help on how the various components of the experiment can be designed and what could be measured.

Do et al. [13], [14] define SIR, the Software Artifact Infrastructure Repository (sir.unl.edu), 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. No clear methodological guidelines are given.

The work from Eldh et. al. [10] describes a framework for the comparison of the efficiency, effectiveness and applicability of testing techniques based on fault injection. The steps of the framework are: prepare code samples with known faults through fault injection; select a testing technique; perform the experiment and collect data; analyze the data; and repeat the experiment if necessary. The paper describes industry challenges for every step. This work describes an interesting methodological framework, however, it is restricted to fault injection.

Most works describe general guidelines, roadmaps or *organizational* steps for software evaluations. Nevertheless, guidelines for the whole field of software engineering cannot be methodological since there are too many different techniques. Some relevant works for controlled experiments in software engineering are [15], [16]. For case studies in software engineering the reader is referred to [17]–[19].

We want to mention the DESMET organizational framework [16] separately, because, like our framework, it is especially developed for evaluating methods and tools within companies and is not so much directed to researchers but to tool vendors, software engineers wanting to assess a proposed change, etc. This means that the studies are context-dependent, and where we do not expect a specific method/tool to be the best in all circumstances. DESMET identifies nine methods for empirical evaluation and define a set of criteria to help evaluators to select an appropriate method. DESMET separates empirical method like experiments, case studies and surveys into quantitative and qualitative, resulting in the first 6 evaluation methods. Moreover they distinguish: qualitative screening, qualitative effect analysis and benchmarking. Our framework will only concentrate on qualitative and quantitative case studies.

III. A brief note on terminology

We will use the following terminology that is consistent with IEEE Standard Glossary of Software Engineering Terminology (Std 729-1983), the IEEE Standard Classification for Software Anomalies (Std 1044-2009) and the IFIP (International Federation for Information Processing): a Fault is the incorrect code that results from a human mistake. A Failure is the incorrect behaviour of the software that the user can observe.

Words like bug, issue, defect will be used interchangeably with fault. Words like anomaly, incident, problem will be used interchangeably for failure.

We consider software techniques or tools whose objectives are to find faults. That means for example test case generation techniques or tools that help define or automatically generate test cases that have to be executed on the SUT (System Under Test) in order to look for failures. But this also includes tools for noise making, like concurrency testing or load/stress testing tools that have to run in parallel with the system in order to provoke failures. This does not directly include regression testing or other minimization techniques whose objectives are to select test cases from a given suite using some criteria. For these techniques the framework might have to be adapted with additional measures.

IV. A Methodological Framework to Evaluate Testing Techniques and Tools

Imagine company C wants to evaluate T to see if it is usable and worthwhile to incorporate this technique or tool into its testing processes. The following sections help defining a case study.

A. Objective - What to achieve?

The general framework focuses on the measures of usability defined by ISO 9241-11: efficiency, effectiveness, and subjective satisfaction. Consequently, the research questions for each case study correspond to instantiations of:

RQ_1 How does T contribute to the effectiveness of testing (fault-finding capabilities) when it is used in real testing environments of C and compared to the current testing practices used at C?

RQ_2 How does T contribute to the efficiency of testing when it is used in real testing environments of C and compared to the current testing practices used at C?

RQ_3 How satisfied are testing practitioners of C during the learning, installing, configuring and usage of T when it is used in real testing environments?

B. Cases or Treatments - What is studied?

The case or treatment is the testing technique or tool T that is evaluated by means of the case study should be described. When designing and conducting an empirical study in software testing, its positioning in the field should be easily and clearly determined in order to be able to unify and combine results into families of experiments [20] or aggregate the results in secondary studies [1]. This approach demands for the usage of a taxonomy (as [21] calls it) or a hierarchy (as [20] calls it) or a characterisation schema (as [22] calls it) of the techniques, methods and tools under investigation. Given such taxonomy, it becomes possible to interpret the narrow results of a single study in the wider context. It is possible, for example, to determine the most closely related experiments or to design further experiments which cover neighbouring areas. It is also possible to understand the generality of the results in terms of its height in the taxonomy.

In our framework we have to decided to use the taxonomy from [21], that we have adapted to software testing and augmented with the results from [22]. The resulting schema is below. Note, not all of these items might be known for the treatment under consideration. Maybe the case study that is being defined has the objective to get some insights in to some of these characteristics.

Prerequisites

- *Software type*: type of software that can be tested with the technique
- *Development or life-cycle phase* to which it is linked.
- *Environment*: platform (hw and sw) and programming language with which it operates.
- *Scalability*: To what system size has it been applied?
- *Input*: What input, e.g., source code, executable program and execution scenarios, test cases, documentation, etc., does it require?
- *Knowledge*: required to be able to apply the technique
- *Experience*: required to be able to apply the technique

Results

- *Output*: What output, e.g., test cases, faults, anomalies, coverage data, etc..?
- *Completeness*: coverage provided by the test cases
- *Effectiveness*: Capability of finding faults
- *Defect types*: Type of faults that can be detected
- *Number of generated test cases*: per software size unit

Operation

- *Interaction*: What interaction modes, e.g., navigation, queries, successive refinement, etc., does it support?
- *User Guidance*: What guidance, e.g., none (i.e., it is completely automated),manual evaluation of output, selection of appropriate inputs, definition of patterns, filtering, etc., does it require from the user?
- *Sources of information*: where can you find information about how to use it.
- *Task Applicability*: To what tasks can it be applied? Is it general or special purpose?
- *Comprehensibility*: whether or not it is easy to understand.
- *Subjective satisfaction*
- *Effort*: How much effort it takes to apply it (effort in learning, installing, configuration and executing)
- *Maturity*: How mature is the treatment (e.g. [23] Still being developed; Not in use in commercial projects; Used in a few products produced by our own organization; Widespread use in own organization; Used in a few products outside of own organization; Widespread use outside of own organization.

Obtaining the tool

- *License*: Open source, Shareware of Commercial
- *Cost*: of purchase and maintenance
- *Support*: where can you turn to when help is needed.

C. Subjects - Who apply the techniques/tools?

Ideally, the subjects are workers of C. Subjects should be those people that normally use the techniques or tools that are being compared to T. If for some reason this is not possible (e.g. lack of time and resources, the tool is an academic prototype, etc.) then researchers or tool developers can evaluate the tools. However, this does mean that the subjective satisfaction cannot be measured and no results can be obtained of the capabilities of the tool within an industrial environment.

D. Objects - What are the pilot projects?

The **S**ystem **U**nder **T**est (SUT) should be a system that is typical of C (i.e. the way the software is developed, the way it is tested, languages used, etc.). Also, the available information about this system should be determined or measured in order to do the comparison with T. The following questions need to be answered:

S1 Will there be access to a system with known faults? What information is present about these faults?

S2 Can faults be injected into the system?

S3 Does C gather data from projects as standard practice? What data is this? Can this data be made available for comparison? Is there a company baseline?

S4 Does C have enough time and resources to execute various rounds of tests?, or more concrete:

- Is company C willing to make a new testsuite TS_{na} with some technique/tool T_a already used in the company C?
- Is company C is willing to make a new testsuite TS_{nn} with some technique/tool T_n that is also new to company C?
- Can we use an existing testsuite TS_e that we can use to compare? Do we know the techniques that were used to create that test suite, and how much time it took?

E. Variables - Which data to collect?

Independent and dependent variables are the attributes that define the study setting.

- Independent: Testing method T used; Complexity of the Industrial systems; Level of experience of testers of C that will do the testing.
- Dependent: Effectiveness, Efficiency, Satisfaction

The following is a list of metrics that could be measured. For a specific instantiation of this framework in a company, some variables might not be applicable.

1) Measuring effectiveness
   a) Number of test cases designed or generated.
   b) Number of invalid test cases generated.
   c) Number of repeated test cases generated.
   d) Number of failures observed.
   e) Number of faults found.
   f) Number of false positives (i.e. the test is marked as Failed, when the functionality is working).
   g) Number of false negatives (i.e. the test is marked as Passed, when the functionality is not working).
   h) Type and cause of the faults that were found.
   i) Coverage reached (estimated or measured).
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.).
   d) Time needed to test and observe failures (i.e. planning, implementation and execution).
   e) Time needed to identify fault types and causes for each observed failure.
3) Measuring subjective satisfaction
   a) **S**ystem **U**sability **S**core (SUS) questionnaire [24] consisting of 10 questions with 5 Likert-scale and a total score.
   b) 5 reactions (through reaction cards) that will be used to create a word cloud and Ven diagrams.
   c) Emotional face reactions during interviews (faces will be evaluated on a 5 Likert-scale from "not at all like this" to "very much like this").
   d) Subjective opinions about T.

We have decided to use the SUS questionnaire because this simple questionnaire gives most reliable results [25], [26]. We complement SUS with other measures since questionnaires alone have known limitations. In a review, Hornbaek [27] concludes that measures of satisfaction should be extended beyond questionnaires. We extend them with new methods for measuring satisfaction developed by Microsoft, i.e. reaction cards and faces questionnaires [28], [29].

F. Protocol - How to execute the study?

In Figure 2 the steps that have to be taken are depicted. If faults can be injected into the systems, care should be taken that [30]

- The artificially seeded faults are similar to faults that naturally occur in real programs due to mistakes made by developers. To identify realistic fault types, a history-based approach can be used, i.e. real faults can be fetched from the bug tracking system and made sure that these reported faults are an excellent representative of faults that are introduced by developers during

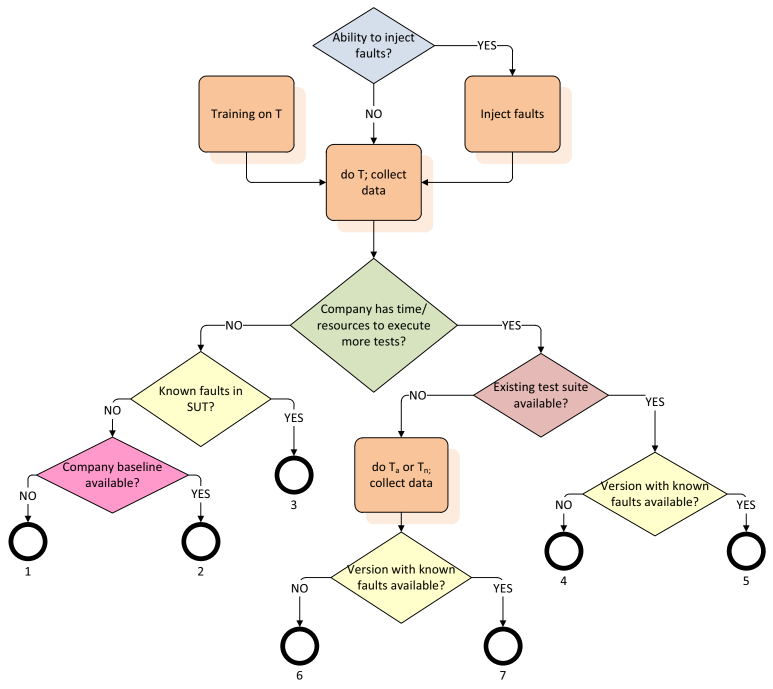
Ability to inject faults?
YES
NO
Training on T
Inject faults
do T; collect data
Company has time/ resources to execute more tests?
NO
YES
Known faults in SUT?
NO
YES
Company baseline available?
NO
YES
1
2
3
Existing test suite available?
NO
YES
do T_a or T_n; collect data
Version with known faults available?
NO
YES
6
7
Version with known faults available?
NO
YES
4
5

Figure 2. Possible scenarios fir the case study protocols

implementation. Also a faults taxonomy could be used like the one from [31].

- The faults should be injected fairly, i.e., an adequate number of instances of each fault type is seeded.

Then after the proper training about the treatment under study, the subjects should **do** the tests (i.e. learn, install and configure the tool; and design/generate and execute the test cases) and collect the data. The type and the procedure to execute each case study depends on the answers given to the questions in Section IV-D. This results in 7 possible case study scenarios (refer to Section IV-D (S4) to remember the meaning of the T_a, T_n, TS_e, etc.):

Scenario 1 consists of a qualitative assessment. Since we do not know how many errors there are, we cannot compare with other techniques, nor do we have a company baseline, we cannot do a quantitative evaluation. However, studying and reporting on the measurements found for effectiveness, efficiency and subjective satisfaction will be done during the semi-structured interviews with the testing practitioners.

Scenario 2 consist of Scenario 1 $\wedge$ quantitative analysis based on company baseline. The extent to which this is possible and how valid the conclusions are, depend on the data that is present in the company baseline.

Scenario 3 consists of (Scenario 1 $\vee$ Scenario 2) $\wedge$ quantitative analysis of **F**ault **D**etection **R**ate (FDR) w.r.t. the known set of faults.

Scenario 4 consists of (Scenario 1 $\vee$ Scenario 2) $\wedge$ quantitative comparison of T and TS_e. This scenario adds a quantitative comparison of T with TS_e. Since TS_e already exists, there are some measures that cannot be compared (e.g. related to the creation/design of the Test Suite, etc) these will be covered with scenario 1 analysis.

Scenario 5 consist of Scenario 4 $\wedge$ FDR of T and TS_e. This scenario adds a quantitative comparison of the fault detection ratio T with TS_e to scenario 4.

Scenario 6 consists of (Scenario 1 $\vee$ Scenario 2) $\wedge$ quantitative comparison of T and (T_a or T_n).

Scenario 7 consists of Scenario 6 $\wedge$ FDR of T and (T_a or T_n).

G. Threats to Validity of the Studies Performed

Threats to validity should be studies carefully for each instantiation of this framework. However, according to [15] we can distinguish: 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: random heterogeneity of subjects.

V. Applying the framework

To show the applicability of the framework, this section describes three instantiations to case studies whose objectives were to evaluate and compare testing techniques. The three studies are very different in nature, yet the framework could be easily instantiated. The first two studies have been done within an industrial environment where no information about existing bugs and existing testing techniques could be provided. The third study has been done within an academic environment where defects could be easily injected. The first study concerns a fully automated structural testing technique for which little previous knowledge and or experience was required. The second study concerns a sophisticated black-box testing technique that requires a significant amount of previous knowledge and experience in order to make it work. The third study compares four testing techniques, 3 of which are automated and one of which is manual. The next sections show that the framework can be effectively applied to describe all three studies.

A. *Search Based Structural Testing*

In [32] an instantiation of the described evaluation framework is used to execute a case study whose main goal is to research the scalability of the search based structural testing techniques developed within the EvoTest project [8] and automated within a tool called the ETF (Evolutionary Testing Framework). The description of the treatments according to the taxonomy from Section IV-B is in Table I.

The described tools are evaluated within two companies (Daimler[1] and Berner& Mattner[2]) that participated in the case study as part of their participation in EvoTest.

The instantiated research questions from Section IV-A related to effectiveness, efficiency and user satisfaction related to the testing technique are:

1) In comparison with random testing the ETF is more effective and more efficient in finding test cases for real-world systems.
2) The amount of time, effort and knowledge necessary to configure and use the ETF make it worthwhile to use it within an industrial setting.

The subjects (Section IV-C) in this case studies were testers employed by the two industrial companies. The objects (Section IV-D) were C functions selected from real-world automotive systems like an active brake assistant, rear window defroster, global powertrain engine controller, etc. Due to restrictions in the companies that could not share information about existing bugs, existing test suites, nor could inject faults, the answer to the questions S1 to S3 from section IV-F were al NO. The answer to question S4 was yes, but only if application of T_n could be done with a minimum of human effort by choosing an automated test generation method, like e.g. random testing. Consequently, the scenario for this case study (as explained in Section IV-F) corresponds to number 6, i.e. a qualitative assessment from scenario 1 and quantitative comparison of the ETF with random testing.

Variables from Section IV-E that were measured were the number of test cases (variable 1a), the degree of structural code coverage (variable 1i), the time needed to set up ETF (variable 2c), time to generate the test cases (variable 2b), time to test the system (variable 2d), and the general qualitative subjective satisfaction opinions within the industrial setting through an informal interview (variable 3d). Since no faults were expected to be found within this software that

Prerequisites	
Static or dynamic	Dynamic
Software Type	ISO C99 code
Lifecycle phase	Unit testing
Environment	Eclipse C
Scalability	Investigated by the mentioned case study.
Input	C code. Optionally: upper and lower bounds for variables; manual tuning of the parameters of the evolutionary engine.
Knowledge	If no optional parameters are provided: None. If optional parameters (see experience).
Experience	Some experience with coverage testing. If user wants to tune parameters then knowledge and experience is needed on evolutionary algorithms and tuning.
Results	
Output	Test cases; coverage information; faults.
Completeness	Investigated by the mentioned case study.
Effectiveness	Investigated by the mentioned case study.
Defect types	Investigated by the mentioned case study.
Test suite size	Investigated by the mentioned case study.
Operation	
Interaction	The users has to set up an Eclipse C-project and set up the compiler and linker preferences. Then just select the C function to test and evaluate the results.
User guidance	Eclipse menus can guide the user.
Source of information	Research papers; User Manual
Task applicability	Unit testing; Code coverage testing
Comprehensibility	Fully automatic if no optional parameters are set.
Subjective satisfaction	Investigated by the mentioned case study.
Effort	Investigated by the mentioned case study.
Maturity	Academic research tools under development
Obtaining the tool	Open source, support from researchers.

Table I
DECRIPTION OF THE ETF WHITE-BOX TESTING TECHNIQUES

[1]http://www.daimler.com
[2]http://www.berner-mattner.com

was already under production, the fitness values' progress was measured to get a qualitative measure for the quality of the test cases.

The instantiation of the **do** from section IV-F consisted of: (1) Install and configurate according to the ETF user manual. During these activities, work-diaries were maintained that contain a lists of the tasks (including their date, time and description) that are performed to set up the ETF according to the user manual (e.g. installation, configuration, find an appropriate set of parameters for the evolutionary engine, etc.); (2) Run each search 30 times for ensuring statistical meaning and collect the data related to the variables selected; (3) Have informal interviews about the general suitability and acceptability in the specific industrial setting.

B. Search Based Functional Testing

In [33] an instantiation of the described evaluation framework is used to execute a case study whose main goal is to research the applicability of the search based functional testing techniques automated within the ETF from the previous section. The description of the treatments according to the schema from Section IV-B is in Table II.

Since search based functional testing is not completely automated (as is structural testing from the previous section) and again the companies indicated that the case study systems were taken from serial production developments in which is would be very unlikely to find faults, and faults could not be injected, the instantiated research questions from section IV-A related to the three usability properties for this case study were:

1) Effectiveness
   - The ETF applied to real-world sized examples, in real-world test environments, is able to generate better test cases w.r.t. achieving the test goal than random testing.
   - The ETF is more effective in finding error revealing test cases when applied to real-world systems for black-box testing compared to random testing.
2) User satisfaction and efficiency
   - It is possible to use the ETF without detailed knowledge in evolutionary computation to search for interesting test data.
   - After installation of the ETF, the amount of time and effort it takes to configure the ETF in order to apply it to real-world systems for evolutionary functional testing is suitable within an industrial setting.

The subject (Section IV-C) in this case studies were embedded systems testers, three testers within each of the two companies. The testers already had some prior knowledge of evolutionary testing principles.

The objects (Section IV-D) used in the studies are real-world embedded control systems from the automotive domain. Both case study systems were taken from serial production developments, one for an adaptive cruise control systems and another for an anti-lock breaking system.

Again, the companies could not share information about

Prerequisites	
Static or dynamic	Dynamic
Software Type	Any type of system for which the input items (see below) can be developed.
Lifecycle phase	Requierements testing at the acceptance or system level test.
Environment	Eclipse IDE
Scalability	Investigated by the mentioned case study.
Input	Individual specification, test drivers, a validated objective function for breaking the selected requirement; Optionally: manual tuning of the parameters of the evolutionary engine.
Knowledge	Knowledge about the requirements to break; about the definition and validation of objective functions; knowledge to implement test drivers that provides the connection between the framework and the System Under Test
Experience	Experience with requirements testing and evolutionary testing.
Results	
Output	Test cases; faults.
Completeness	Investigated by the mentioned case study.
Effectiveness	Investigated by the mentioned case study.
Defect types	Investigated by the mentioned case study.
Test suite size	Investigated by the mentioned case study.
Operation	
Interaction	To customize the framework for a particular test aim, the user has to supply the following domain-specific components: 1) an individual specification, 2) a test driver, and 3) an objective function. The individual specification describes the structure of the individuals, that is the test data in an XML file. The test driver provides the connection between the framework and the SUT. It converts the individuals from the search process into test data. Subsequently, the test driver executes the SUT using the test data and monitors the output of the SUT. The monitoring results are passed back to the framework and are used by the objective function to calculate the adequacy of the test data.
User guidance	Eclipse menus can guide the user.
Source of information	Research papers; User Manual
Task applicability	Requirements testing.
Comprehensibility	Since the objective function has to be provided, the technique is not easy to understand nor apply.
Subjective satisfaction	Investigated by the mentioned case study.
Effort	Investigated by the mentioned case study.
Maturity	Academic research tools under development
Obtaining the tool	Open source, support from researchers.

Table II
DECRIPTION OF THE ETF BLACK-BOX TESTING TECHNIQUES

existing bugs, existing test suites, nor could inject faults, nor test with other techniques, so the answer to the questions S1 to S4 from section IV-F were al NO. Consequently, the scenario from Section IV-F for this case study corresponds to number 1, i.e. a qualitative assessment. The quality of the test cases was investigated through the resulting fitness values to verify that while fitness values are improving, the system is continuously being exercised closer and closer to a boundary condition (an optimal fitness value means breaking that boundary, i.e. the requirement). A quantitative comparison to random testing was added to have a baseline for comparing the underlying search algorithms.

Variables from Section IV-E that were measured were the number of test cases (variable 1a), the number of invalid test cases (variable 1b), number of faults found (variable 1e), the time needed to set up ETF (variable 2c) (including the time to define the fitness function), time to generate the test cases (variable 2b), time to test the system (variable 2d), and the general qualitative subjective satisfaction opinions within the industrial setting through an informal interview (variable 3d). Since no faults were expected to be find within this software that was already under production, the fitness values' progress was measured to get a qualitative measure of the quality of the test cases.

The instantiation of the **do** from section IV-F consisted of: (1) Install and configure according to the ETF user manual; Maintenance of work-diaries that should contain the tasks (including their date, time and description) that are performed to set up the ETF according to the user manual (i.e. tasks like to find an appropriate set of parameters for the evolutionary engine, etc.); (2) Implement the case study specific components (e.g. individual specification and test drivers). Work-diaries will be maintained in order to be able to estimate the necessary effort; (3) Define, refine and implement the fitness function and validate its suitability for breaking the requirement, working diaries will be maintained; (4) Run each search 30 times to give it statistical meaning and collect data listed below; (5) Have informal interviews about the general suitability and acceptability in the specific industrial setting.

C. Web testing of AJAX applications

In [34] an instantiation of the framework is presented that evaluates the capabilities of 4 testing techniques frequently used for web testing: model-based testing, coverage-based testing, black-box testing and state-based testing. The description of these four techniques according to the schema from Section IV-B are put together in Table III.

The techniques are evaluated by academics and the objects consisted of web applications are drawn from a student book instead of real world industrial applications, although care was taken that these applications were selected in a way such that the selected applications are typical in terms of technologies, frameworks, and languages used in industrial applications. Since the studies were executed by academics, no study could be done related to subjective satisfaction and hence the instantiated research questions were:

- What is the effectiveness in revealing faults of each of the Web testing techniques.
- What is the effort required to apply each Web testing technique?

Considering the fact that this was no industrial software,

Prerequisites	
Static or dynamic	Model/state/code coverage based testing require execution to generate test cases; Black-box testing requires a static analysis of the application requirements.
Software Type	Web and Web 2.0 applications.
Lifecycle phase	System level test.
Environment	Web environment: web server to install the application and web browser to run it.
Scalability	Small/medium size web applications.
Input	Model/state based testing: execution logs are required; Code coverage based testing: code to be instrumented for computing the coverage when running the test cases; Black-box testing: application requirements.
Knowledge	All testing methods require the knowledge of the application domain; State based testing requires also knowledge about how to identify the state of the system under test; Black-box testing: requires the ability of analyzing requirements, i.e., identifying scenarios;
Experience	Experience with web testing.
Results	
Output	Test cases; faults.
Completeness	Investigated by the mentioned case study.
Effectiveness	Investigated by the mentioned case study.
Defect types	Investigated by the mentioned case study.
Test suite size	Investigated by the mentioned case study.
Operation	
Interaction	Model/State/Code coverage testing: testers are asked to run the application to collect execution traces, then used to generate the test cases. Black-box: testers are asked to analyze the requirements to derive test cases; All methods: testers are asked to check the results of the test case execution.
Source of information	Research papers
Task applicability	System testing.
Comprehensibility	In state based testing could be difficult to understand what are possible application states must be tested, so that must be captured by models used to derive test cases.
Subjective satisfaction	No studies yet available.
Effort	Investigated by the mentioned case study.
Maturity	Academic research tools under development
Obtaining the tool	Open source, support from researchers.

Table III
DECRIPTION OF THE WEB TESTING TECHNIQUES COMPARED

faults could be injected and consequently the scenario from Section IV-F for this case study corresponds to number 3. Variables that were measured were: number of faults found (variable 1e), coverage of the use cases (variable 1i), type and severity of the faults that were found (variable 1h), test suites size (variable 1a), Time (in man-hours) needed to set up the testing infrastructure specific (variable 2c), Complexity of the test suites (variable 2d).

The instantiation of the **do** from section IV-F consisted of: (1) Inject faults (this is done by a person different from the tester) into the original Web applications, trying to simulate real programming errors using a taxonomy or defects from [31]. These changes do not break the execution of the target application, but lead to wrong or unexpected behaviours; (2) Apply the selected Web testing techniques (i.e., model-based, code-coverage, black-box, and state-based) to all the faulty Web applications with the aim of deriving suites of test cases for each of them; (3) Use each test suite to test the faulty Web applications.

VI. Conclusions

A methodological framework to evaluate software testing techniques has been presented. The objective of this framework is to 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 are to be executed according to a similar design, it will be more easy to compare the results obtained, and hence a body of evidence can be constructed that will enable researchers to investigate general statements about testing techniques and tools evaluated in different case studies might be specified.

We have presented three successful instantiations or applications of the framework to validate its applicability and effectiveness. However, the framework needs to be instantiated by many more case studies to validate the completeness of the identified variables in Section IV-E and the identified scenarios from Section IV-F. Our future work plans go in that direction, do many more case studies to validate the framework, refine it and start creating this so needed body of evidence that contain evaluations of software testing techniques.

Acknowledgment

This work was funded by the European project FITTEST (ICT257574, 2010-2013) and Spanish National project CaSA-Calidad (TIN2010-12312-E, Ministerio de Ciencia e Innovacin).

References

[1] B. Kitchenham, T. Dyba, and M. Jorgensen, "Evidence-based software engineering," in *Proc of ICSE*. IEEE, 2004, pp. 273–281.

[2] N. Juristo, A. Moreno, and S. Vegas, "Reviewing 25 years of testing technique experiments," *Empirical Softw. Engg.*, vol. 9, no. 1-2, pp. 7–44, 2004.

[3] P. Runeson, C. Andersson, T. Thelin, A. Andrews, and T. Berling, "What do we know about defect detection methods?" *IEEE Softw.*, vol. 23, no. 3, pp. 82–90, 2006.

[4] A. C. D. Neto, R. Subramanyan, M. Vieira, G. H. Travassos, and F. Shull, "Improving evidence about software technologies: A look at model-based testing," *IEEE Software*, vol. 25, no. 3, pp. 10–13, 2008.

[5] S. Hesari, H. Mashayekhi, and R. Ramsin, "Towards a general framework for evaluating software development methodologies," in *Proc of 34th IEEE COMPSAC*, 2010, pp. 208–217.

[6] N. Fenton, S. Pfleeger, and R. Glass, "Science and substance: a challenge to software engineers," *Software, IEEE*, vol. 11, no. 4, pp. 86 –95, Jul. 1994.

[7] D. Janzen, C. Turner, and H. Saiedian, "Empirical software engineering in industry short courses," in *Proc of the 20th Conf on Softw Eng Education & Training*, 2007, pp. 89–96.

[8] T. E. J. Vos, "Evolutionary testing for complex systems," *ERCIM News*, vol. 2009, no. 78, 2009.

[9] ——, "Continuous evolutionary automated testing for the future internet," *ERCIM News*, vol. 2010, no. 82, pp. 50–51, 2010.

[10] S. Eldh, H. Hansson, S. Punnekkat, A. Pettersson, and D. Sundmark, "A framework for comparing efficiency, effectiveness and applicability of software testing techniques," *TAIC Part*, pp. 159–170, 2006.

[11] C. Lott and H. Rombach, "Repeatable software engineering experiments for comparing defect-detection techniques," *Empirical Software Engineering*, vol. 1, pp. 241–277, 1996.

[12] V. Basili, R. Selby, and D. Hutchens, "Experimentation in software engineering," *IEEE TSE*, vol. 12, pp. 733–743, 1986.

[13] H. Do, G. Rothermel, and S. Elbaum, "Infrastructure support for controlled experimentation with software testing and regression testing techniques," in *Proc. Int. Symp. On Empirical Software Engineering*, 2004.

[14] H. Do, S. Elbaum, and G. Rothermel, "Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact," *Empirical Softw. Eng.*, vol. 10, no. 4, pp. 405–435, 2005.

[15] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering: an introduction*. Kluwer, 2000.

[16] B. Kitchenham, S. Linkman, and D. Law, "Desmet: a methodology for evaluating software engineering methods and tools," *Computing Control Engineering Journal*, vol. 8, no. 3, pp. 120 –126, Jun. 1997.

[17] B. Kitchenham, L. Pickard, and S. Pfleeger, "Case studies for method and tool evaluation," *Software, IEEE*, vol. 12, no. 4, pp. 52 –62, Jul. 1995.

[18] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Softw. Engg.*, vol. 14, no. 2, pp. 131–164, 2009.

[19] M. Host and P. Runeson, "Checklists for software engineering case study research," in *Proceedings of the First International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 479–481. [Online]. Available: http://dx.doi.org/10.1109/ESEM.2007.29

[20] V. R. Basili, F. Shull, and F. Lanubile, "Building knowledge through families of experiments," *IEEE Trans. Softw. Eng.*, vol. 25, no. 4, pp. 456–473, 1999.

[21] P. Tonella, M. Torchiano, B. Du Bois, and T. Systä, "Empirical studies in reverse engineering: state of the art and future trends," *Empirical Softw. Engg.*, vol. 12, no. 5, pp. 551–571, 2007.

[22] S. Vegas and V. Basili, "A characterisation schema for software testing techniques," *Empirical Softw. Engg.*, vol. 10, no. 4, pp. 437–466, 2005.

[23] B. A. Kitchenham, S. L. Pfleeger, L. M. Pickard, P. W. Jones, D. C. Hoaglin, K. E. Emam, and J. Rosenberg, "Preliminary guidelines for empirical research in software engineering," *IEEE Trans. Softw. Eng.*, vol. 28, no. 8, pp. 721–734, 2002.

[24] J. Brooke, "Sus: a 'quick and dirty' usability scale," in *Usability Evaluation in Industry*. Taylor and Francis, 1996.

[25] T. Tullis and J. N. Stetson, "A comparison of questionnaires for assessing website usability." in *Proc of the Usability Professionals Association Conf*, 2004.

[26] A. Bangor, P. T. Kortum, and J. T. Miller, "An empirical evaluation of the system usability scale," *International Journal of Human-computer Interaction*, vol. 24, pp. 574–594, 2008.

[27] K. Hornbaek, "Current practice in measuring usability: Challenges to usability studies and research," *Int. J. Hum.-Comput. Stud.*, vol. 64, pp. 79–102, February 2006. [Online]. Available: http://portal.acm.org/citation.cfm?id=1140933.1140935

[28] J. Benedek and T. Miner, "Measuring desirability: New methods for measuring desirability," in *Proc of the Usability Professionals Association Conf*, 2002.

[29] D. Williams, G. Kelly, and L. Anderson, "Msn 9: new user-centered desirability methods produce compelling visual design," in *Extended abstracts on Human factors in computing systems*, ser. CHI '04, 2004, pp. 959–974.

[30] A. Memon and Q. Xie, "Studying the fault-detection effectiveness of gui test cases for rapidly evolving software," *Software Engineering, IEEE Transactions on*, vol. 31, no. 10, pp. 884 – 896, oct. 2005.

[31] A. Marchetto, F. Ricca, and P. Tonella, "Empirical validation of a web fault taxonomy and its usage for fault seeding," in *Proceedings of the 2007 9th IEEE International Workshop on Web Site Evolution*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 31–38. [Online]. Available: http://portal.acm.org/citation.cfm?id=1524880.1525461

[32] T. E. J. Vos, A. I. Baars, F. F. Lindlar, P. M. Kruse, A. Windisch, and J. Wegener, "Industrial scaled automated structural testing with the evolutionary testing tool," in *ICST*, 2010, pp. 175–184.

[33] T. E. J. Vos, F. Lindlar, B. Wilmes, A. Windisch, A. Baars, P. Kruse, H. Gross, and J. Wegener, "Evolutionary functional black-box testing in an industrial setting," *Software Quality Journal*, pp. 1–30, 2012, 10.1007/s11219-012-9174-y. [Online]. Available: http://dx.doi.org/10.1007/s11219-012-9174-y

[34] A. Marchetto, F. Ricca, and P. Tonella, "A case study-based comparison of web testing techniques applied to ajax web applications," *International Journal on Software Tools for Technology Transfer (STTT)*, vol. 10, pp. 477–492, 2008, 10.1007/s10009-008-0086-x. [Online]. Available: http://dx.doi.org/10.1007/s10009-008-0086-x