Do Automatically Generated Test Cases Make Debugging Easier? An Experimental Assessment of Debugging Effectiveness and Efficiency

MARIANO CECCATO and ALESSANDRO MARCHETTO, Fondazione Bruno Kessler
LEONARDO MARIANI, University of Milano Bicocca
CU D. NGUYEN, Fondazione Bruno Kessler and SnT Centre, University of Luxembourg
PAOLO TONELLA, Fondazione Bruno Kessler

Several techniques and tools have been proposed for the automatic generation of test cases. Usually, these tools are evaluated in terms of fault-revealing or coverage capability, but their impact on the manual debugging activity is not considered. The question is whether automatically generated test cases are equally effective in supporting debugging as manually written tests.

We conducted a family of three experiments (five replications) with humans (in total, 55 subjects) to assess whether the features of automatically generated test cases, which make them less readable and understandable (e.g., unclear test scenarios, meaningless identifiers), have an impact on the effectiveness and efficiency of debugging. The first two experiments compare different test case generation tools (Randoop vs. EvoSuite). The third experiment investigates the role of code identifiers in test cases (obfuscated vs. original identifiers), since a major difference between manual and automatically generated test cases is that the latter contain meaningless (obfuscated) identifiers.

We show that automatically generated test cases are as useful for debugging as manual test cases. Furthermore, we find that, for less experienced developers, automatic tests are more useful on average due to their lower static and dynamic complexity.

5

Categories and Subject Descriptors: D.2.5 [**Software Engineering**]: Testing and Debugging—*Testing tools*

General Terms: Experimentation, Verification

Additional Key Words and Phrases: Empirical software engineering, debugging, automatic test case generation

ACM Reference Format:
Mariano Ceccato, Alessandro Marchetto, Leonardo Mariani, Cu D. Nguyen, and Paolo Tonella. 2015. Do automatically generated test cases make debugging easier? An experimental assessment of debugging effectiveness and efficiency. ACM Trans. Softw. Eng. Methodol. 25, 1, Article 5 (November 2015), 38 pages.
DOI: http://dx.doi.org/10.1145/2768829

1. INTRODUCTION

Automated test case generation has been a major area of investigation in the last decade [Pacheco and Ernst 2007; Fraser and Arcuri 2011; Tillmann and Halleux 2008; Fraser and Zeller 2010]. Tools for automated test case generation typically aim at maximizing branch or path coverage. Randoop [Pacheco and Ernst 2007] produces Java test cases automatically as random sequences of method invocations with random

Authors' addresses: M. Ceccato (corresponding author), A. Marchetto, Fondazione Bruno Kessler, Trento, Italy; email: ceccato@fbk.eu; L. Mariani, University of Milano Bicocca, Milano, Italy; C. D. Nguyen, Fondazione Bruno Kessler, Trento, Italy and SnT Centre, University of Luxembourg, Luxembourg; P. Tonella, Fondazione Bruno Kessler, Trento, Italy.

DOI: http://dx.doi.org/10.1145/2768829

parameter values. EvoSuite [Fraser and Arcuri 2011] and eToc [Tonella 2004] use a genetic algorithm to maximize branch coverage for Java classes under test. Other tools target path coverage by means of dynamic symbolic execution [Godefroid et al. 2005; Sen et al. 2005], including DART [Godefroid et al. 2005], CUTE [Sen et al. 2005], EXE [Cadar et al. 2008], and Crest [Burnim and Sen 2008] for C code, PEX [Tillmann and Halleux 2008] for C# code, and SAGE [Godefroid et al. 2008] for binary code.

While substantial effort has been devoted to empirically study the effectiveness of these solutions in revealing faults [Pacheco and Ernst 2007; Beckman et al. 2010; Tillmann and Halleux 2008; Godefroid et al. 2005; Fraser and Zeller 2010], the problem of the actual usability of the automatically generated test cases for the developers who rely on them for debugging is a neglected topic. Automatically generated test cases (autogen, for brevity) are in fact less understandable and meaningful than manually designed test cases. Indeed, autogen tests do not exercise any meaningful test scenario from the end-user's viewpoint; they do not address explicitly any high-level requirement or functionality; they include meaningless identifiers and data values. Hence, interpretation of a failing execution might be substantially more difficult with autogen test cases than with manual tests. This might impact negatively the debugging activities in comparison to debugging when supported by manually written tests.

The problem of the actual usability and effectiveness of autogen test cases is a key problem in the area of automated software testing. So far, no empirical evidence has been collected to show whether there is any negative impact on debugging and the extent of such an impact. When test case generation technologies are considered for industrial adoption, estimating all the associated costs, including indirect costs such as an additional cost during debugging, is critical for informed decision-making.

This article presents the first family of experiments that investigates the impact of automated test case generation on debugging. We have considered two test case generators (Randoop [Pacheco and Ernst 2007] and EvoSuite [Fraser and Arcuri 2011]) in two separate experiments, each replicated twice, to measure empirically the effectiveness and efficiency of debugging when autogen test cases are used as compared to manual test cases available for the applications under test. Since autogen test cases include meaningless identifiers (e.g., `x0`, `x1`) that are generated automatically by tools, we also investigated, in a separate experiment, whether the obfuscation of identifiers in the test cases has any negative impact on debugging. This third experiment aims at assessing the role of meaningful identifiers in test cases during debugging, regardless of any other feature of autogen tests, to understand whether the generation of meaningless identifiers alone has a negative impact on debugging. In total, we involved 55 human subjects in all replications of our three experiments, with a wide range of expertise and capabilities (from BSc and MSc students to PhD/Post-docs and researchers/professors).

This article extends our previous work [Ceccato et al. 2012] with the following key contributions.

—In addition to the Randoop random test case generator, we consider test cases generated by another tool, EvoSuite, which generates test cases using genetic algorithms. A new experiment, replicated twice, was designed and conducted to assess the impact of EvoSuite test cases on debugging.
—We investigate the role of meaningful and understandable code identifiers used in test cases in a new experiment where identifiers of manual test cases are deliberately obfuscated.
—We extend the discussion of our findings to account for the results obtained from the new studies, which allows us to provide a detailed and wide-spectrum analysis of the impact of autogen tests on debugging.

—We analyze the representativeness of the two applications under test chosen for the empirical studies, JTopas and XML-Security, by comparing their features with those of other open-source Java projects.

The article is organized as follows. After discussing related work in Section 2, Section 3 describes the design underlying the family of three experiments reported in this article. Sections 4, 5, and 6 report the results of the three experiments. A discussion of the experimental results is provided in Section 7. Section 8 discusses the threats to the validity of our experiments. We draw our final conclusions in Section 9.

2. RELATED WORK

Test cases are one of the most relevant sources of information to locate faults. Several papers [Artzi et al. 2010; Jones et al. 2002; Zeller and Hildebrandt 2002] propose approaches and techniques to analyze test cases and their execution, with the aim of detecting and locating faults. The underlying idea of test-based debugging is that the information collected during test case execution (e.g., code coverage and application state) can be useful to identify the set of "suspicious" code statements (i.e., those where the fault may be located). Automatically generated test cases have been found to be quite effective in detecting faults [Andrews et al. 2008; Ciupa et al. 2007; Duran 1984]. Several automated test case generation tools have been proposed. Concrete and symbolic execution are mixed in tools such as PEX [Tillmann and Halleux 2008], CUTE [Sen et al. 2005], and DART [Godefroid et al. 2005]. Randoop [Pacheco and Ernst 2007] creates test cases based on randomly generated sequences of method invocations enriched with assertions automatically inferred through dynamic analysis. EvoSuite [Fraser and Arcuri 2011] takes advantage of genetic algorithms to generate test cases for Java classes, with the goal of maximizing the level of branch coverage. Fraser and Zeller [2010] apply mutation analysis to automatically generate test cases. In our experiment, we compared manually defined test cases to the ones obtained with Randoop and EvoSuite. This does not cover the full range of possible test case generators, but we think that Randoop and EvoSuite are representative of two important families of tools (relying on random testing and search-based testing). Moreover, the experiment with obfuscated identifiers provides key insights that hold for all test case generators, since they all generate meaningless (obfuscated) identifiers.

Empirical studies on automated test case generators [Andrews et al. 2008; Ciupa et al. 2007; Duran 1984] are focused on the fault-finding capabilities of tools. However, no study is available about the impact of automatically generated test cases on debugging. In this article, we report a family of experiments that, for the first time, measures the effectiveness of automatically generated test cases when they are used to carry out debugging tasks.

Most of the literature containing empirical studies related to testing and debugging describes experiments that do not involve human subjects. For instance, Frankl and Weiss [1991] compared the capability of revealing faults when test cases satisfy different coverage criteria. However, their work did not consider the intensive manual activity necessary to locate a fault after a test case has revealed it. A few empirical studies on software testing and debugging involved human subjects, but they considered directions different from the one investigated in this article. For instance, the studies by Ricca et al. [2009] and Huang and Holcombe [2009] focus on the testing process and strategy, evaluating the impact of the "test first" strategy either on the accuracy of change tasks or on the quality of the final code.

A few attempts have been made to investigate the relationship between testing and debugging. Fry and Weimer [2010] presented an observational study on the accuracy of human subjects in locating faults. They discovered that certain types of faults are difficult to locate for humans. For instance, “extra statement” faults seem easier to detect than “missing statement” faults. However, the authors did not investigate the role of test cases. They also observed that, independently from the faults, certain code contexts are difficult to debug for humans. For instance, the array abstraction is easier than the tree abstraction. Weiser and Lyle [1986] empirically evaluated the impact of a slicing tool on debugging. They did not observe any improvement when developers use a slicing tool to debug small, faulty programs. Parnin and Orso [2011] performed an experiment to investigate how developers use debugging tools and whether these improve performance. Tools are shown to help complete debugging tasks faster. Still, Parnin and Orso’s [2011] study does not consider the role of test cases.

Other empirical studies on software testing, which are to some extent related to the present article, include: (1) Itkonen et al. [2009], where the authors present an observational study in which they report the manual (functional) testing practices in four software companies. They noticed that many of the applied techniques are based on similar ideas as traditional test case design techniques, even if they rely on experience rather than a formalized technique, as available in scientific literature. (2) Ricca et al. [2009] report a series of studies that investigate the role of acceptance tests when facing requirement changes. (3) Yu et al. [2008] performed an experiment to evaluate the impact of test suite reduction on the effectiveness of fault localization techniques. They also show that fault localization effectiveness strongly varies depending on the test suite reduction strategy considered. (4) Huang and Holcombe [2009] report an experiment in which they compared the effectiveness of test-first and test-last strategies. Although their results are not confirmed statistically, they found that the quality of the produced software increases with an increased time spent on application testing. (5) Ruthruff et al. [2005] report an experiment in which they studied the impact of two factors, that is, information base and mapping, in fault localization. They showed that such factors impact to a major extent the effectiveness of fault localization activities.

To the best of our knowledge, our work is the first empirical study with human subjects that investigates the effectiveness and efficiency of debugging when autogen and manually written test cases are used.

3. EXPERIMENT DEFINITION AND PLANNING

This section describes the definition, design, and settings of the experiments in a structured way, following the template and guidelines by Wohlin et al. [2012].

The goal of the study is to investigate the differences between manually written and autogen test cases, with the purpose of evaluating how well they support debugging tasks. The quality focus regards how manually written and autogen test cases affect the capability of developers to correctly and efficiently debug the code. The results of the experiments are interpreted regarding two perspectives: (1) a researcher interested in empirically validating autogen test cases and (2) a quality manager who wants to understand whether the time spent in writing test cases pays off when facing actual debugging tasks.

The context of the studies is defined as follows: the subjects of the study are developers facing debugging tasks, while the objects are applications that contain the faults to be fixed. We collected empirical data from three experiments.

(1) *Manual vs. Randoop [MvR]*. This experiment compares manual test cases to those produced automatically by a random test case generator, Randoop [Pacheco and Ernst 2007].
(2) *Manual vs. EvoSuite [MvE]*. This experiment compares manual test cases to those produced automatically by an evolutionary test case generator, EvoSuite [Fraser and Arcuri 2011].

Table I. Overview of the Family of Experiments

Experiment	Treatments		Replication	University	Subjects
MvR	Manual tests	Randoop tests	I	Univestity of Trento	7 MSc
			II	Univestity of Milano Bicocca	14 BSc + 8 MSc
MvE	Manual tests	EvoSuite tests	I	Univestity of Milano Bicocca	6 Researchers
			II	Fondazione Bruno Kessler	9 Researchers
MvO	Original identifiers	Obfuscated identifiers	I	Univestity of Milano Bicocca	11 MSc studentes

(3) *Manual vs. Obfuscated [MvO].* This experiment investigates the impact on debugging of a specific aspect of autogen test cases: the artificial identifiers that appear in their implementation. To evaluate the impact of artificial identifiers, we compare manual test cases with original identifiers to manual test cases with identifiers obfuscated, as in the tests produced by test generation tools.

While several tools are available for automated test case generation [Pacheco and Ernst 2007; Fraser and Arcuri 2011; Tillmann and Halleux 2008; Tonella 2004; Godefroid et al. 2005; Sen et al. 2005; Cadar et al. 2008; Burnim and Sen 2008], we restricted our choice to those that support the Java programming language because the students involved in the experiment are mostly familiar with Java. Moreover, we wanted to choose tools that have a reasonable usability and support from their developers and that have reached a good maturity level. Under such constraints, the only two tools that we could find as freely available are Randoop and EvoSuite.

Experiments MvR and MvE do not distinguish two different aspects that affect autogen test cases: (1) their structure and (2) the identifiers they use. Usually, autogen test cases have a simpler structure than manual test cases and, differently from manual test cases, they use meaningless, automatically generated identifiers. The aim of experiment MvO is to factor out one of these two aspects from the other. Namely, MvO investigates the extent to which the presence of meaningless identifiers in autogen test cases is detrimental to debugging activity. Since the simpler structure of autogen tests is potentially beneficial to debugging, by combining the outcome of MvO with that of MvR and MvE, we can understand whether one of the conflicting aspects, simple structure versus meaningless identifiers, compensates for or prevails over the other.

An overview of the experiments is summarized in Table I. Experiment MvR was conducted in two replications: the first involved 7 MSc students of the University of Trento attending the Software Analysis and Testing course. The second replication involved 14 BSc students of the University of Milano-Bicocca attending the Software Analysis and Testing course, and 8 MSc students of the University of Milano-Bicocca attending the Software Quality Control course. Experiment MvE involved 6 professors/Post-docs from University of Milano-Bicocca and 9 researchers/Post-docs from Fondazione Bruno Kessler. MvE differs from MvR on both the type of autogen test cases used in the experiment and the skills of the subjects (mostly PhD students and researchers instead of MSc and BSc students). The involvement of skilled subjects allows assessment of the role of experience and ability on debugging. Experiment MvO was conducted at the University of Milano-Bicocca and involved 11 MSc students attending the Software Development Process course. MSc students from Trento and from Milan share a similar background in computer programming and software engineering, having attended courses with similar content (e.g., Java programming, fundamentals of software engineering, etc.) in previous years at the two universities. The initial results obtained from MvR have been used to tune the design of the experiments MvE and MvO. We report the differences of the empirical setup when applicable. All subjects involved in the studies have basic skills in Java programming, debugging, and use of the Eclipse IDE[1].

The applications used in the experiment are JTopas and XML-Security, further characterized in Appendix A. Both applications are available with manually written tests. Moreover, some of their faults are documented in the SIR[2] repository [Do et al. 2005].

JTopas is a customizable tokenizer that tokenizes input text files. It allows users to customize the grammar of the input files by specifying the structure of keywords, compounds, and comments, and the case sensitivity. JTopas consists of 15 classes and 4,482 NCLoCs (*non-comment lines of code*). XML-Security is a library that provides functionalities to sign and verify signatures in XML documents. It supports many mature digital signature and encryption algorithms on standard XML formats, such as XHTML and SOAP. It consists of 228 classes, for a total of 29,255 NCLoCs.

3.1. Hypotheses Formulation

Based on the study definition reported before, we formulate the following null hypotheses to be tested.

H_{01}. There is no difference in the effectiveness of debugging when debugging is supported by different kinds of test cases.

H_{02}. There is no difference in the efficiency of debugging when debugging is supported by different kinds of test cases.

However, since different kinds of test cases are considered in different experiments (see Table I), these hypotheses can be broken down as follows.

—Experiment MvR

H_{01R}. There is no difference in the effectiveness of debugging when debugging is supported either by manually written or Randoop test cases.

H_{02R}. There is no difference in the efficiency of debugging when debugging is supported either by manually written or Randoop test cases.

—Experiment MvE

H_{01E}. There is no difference in the effectiveness of debugging when debugging is supported either by manually written or EvoSuite test cases.

H_{02E}. There is no difference in the efficiency of debugging when debugging is supported either by manually written or EvoSuite test cases.

—Experiment MvO

H_{01O}. There is no difference in the effectiveness of debugging when debugging is supported either by manually written or Obfuscated test cases.

H_{02O}. There is no difference in the efficiency of debugging when debugging is supported either by manually written or Obfuscated test cases.

These null hypotheses are *two-tailed* because there is no a-priori knowledge on the expected trend that should favor either manually written or autogen test cases. On the one hand, manual test cases are meaningful for a developer who is determining the position of a fault, while automatic tests may contain meaningless statements and code identifiers that may confuse developers. On the other hand, manual tests could be difficult to understand because they may require understanding of complex parts

[1] `http://www.eclipse.org`

[2] `http://sir.unl.edu/portal/index.php.`

of the application logic, while automatically generated tests may be simpler since they are generated without a clear knowledge of the application business logic.

The null hypotheses indicate that we have two dependent variables: *debugging effectiveness* and *debugging efficiency*. In experiments MvR, MvE, and MvO, we asked subjects to fix eight faults in total (four faults for each subject application).

During their fault-fixing activities, developers typically resort to a regression test suite to check whether the code changes introduced to fix the bug have broken any pre-existing functionality. In MvE and MvO, we have provided developers with the regression test suites available for the object applications, XML-Security and JTopas. To avoid any interference between regression testing and bug localization and correction, we made sure that the regression test suite cannot reveal the faults to be fixed in the experiments. To determine whether manual and autogen test cases differ according to the identifier understandability and code complexity metrics, we introduce two additional derived null hypotheses:

DH_{03}. There is no difference in the number of *artificial / user-defined identifiers* of the manually written and autogen test cases used in the experiment.
DH_{04}. There is no difference in the *static / dynamic complexity* of the manually written and autogen test cases used in the experiment.

Experimental support for the alternative hypotheses associated with DH_{03} and DH_{04} provides useful information for the interpretation of the results about the two dependent variables considered in H_{01} and H_{02}.

We have collected the participants' opinions on the performed debugging tasks through survey questionnaires. Answers are on a Likert scale whose extremes are *strongly agree / disagree* and whose middle point is *uncertain*. We analyze the answers to the survey questionnaires by formulating the following null hypotheses.

HQx_{05}. Participants are *uncertain* about what is stated in question Qx.
HQx_{06}. There is no difference in the answer to question Qx between participants who were supported by different kinds of test cases during debugging.

3.2. Variable Selection

In our experiments, the effectiveness of debugging is measured as the number of correctly fixed faults. We evaluated correctness of the fixes by running a predefined set of test cases that cover the faults in the subject programs (these test cases have not been provided to subjects). If all the test cases passed, we further manually inspected the fixed code to verify whether the fix was correct. The efficiency of debugging is evaluated as the number of correct tasks (i.e., the number of correctly fixed faults) divided by the total amount of time spent for these tasks (measured in minutes): $eff = \frac{\sum Corr_i}{\sum Time_i}$, where $Corr_i$ is equal to 1 if the i-th task is performed correctly, 0 otherwise, while $Time_i$ is the time spent to perform the i-th task. In other words, efficiency is measured as the number of correctly performed tasks per minute.

The independent variables (the main factors of the experiments) are the treatments applied during execution of the debugging tasks. The two alternative treatments in the three experiments are (1) manually written test cases, that is, those distributed as unit tests for the object applications (obtained from the SIR repository); and (2) either test cases automatically generated by Randoop [Pacheco and Ernst 2007] (MvR experiment) or test cases automatically generated by EvoSuite [Fraser and Arcuri 2011] (MvE experiment), or obfuscated test cases (MvO experiment).

The understandability of the test cases that reveal the faults might affect debugging performance and may vary substantially between manual and autogen test cases. Since we cannot control this factor in the experiments because it depends on how manual test cases have been defined and how the test case generation algorithms work, we measure this factor in our experimental setting. The test case understandability might, in fact, represent one of the key features in which manual and autogen test cases differ, which could possibly explain some of the observed performance differences.

Unfortunately, there is no easy, widely accepted way of measuring the understandability of test cases. We approximate such measurement by considering two specific factors of understandability, namely identifier meaningfulness and complexity of the test code. For the former we manually classify each identifier in a test case as either *Artificial* (automatically generated) or *UserDef* (user-defined) and count the respective numbers. In order to measure the test case complexity, we consider both static metrics (*MeLoC* and *McCabe*) and dynamic metrics (*Exec. methods* and *Exec. LoCs*), which provide the test case complexity from the developer's perspective[3]. Metrics are computed using the Eclipse Metrics plugin (`http://metrics.sourceforge.net`). As static metrics we measure:

—*MeLoC*, number of nonblank and non-comment lines of code inside each method body; and
—*McCabe*, cyclomatic complexity of each test method.

As dynamic metrics, we consider the amount of application code exercised by each test case. We count it at two granularity levels:

—*Exec. methods*, the number of methods executed by a test case; and
—*Exec. LoCs*, the number of statements executed by a test case.

3.3. Other Factors

We measured the following other factors that could influence the dependent variables:

(1) the subjects' ability;
(2) the subjects' experience;
(3) the object system;
(4) the experiment session; and
(5) the fault to be fixed.

(1) *Subjects' ability.* The ability of subjects in performing debugging tasks was measured using a pretest questionnaire with questions about programming and debugging ability, experience with the development of large applications, and scores in academic courses related to development and testing. In the case of MvR where subjects include BSc students, we also exploited the results of a training session where subjects were asked to answer some code comprehension questions and to fix faults in each of the two object applications. According to the answers given to the pre-questionnaire and the results of the training lab, we classified the subjects who participated in MvR into three categories. *High-ability* subjects are those who had experience with the development of large applications, have an academic score of at least 27/30[4], and completed correctly at least 50% of the tasks in the training lab. *Medium-ability* subjects are those who either had experience with the development of large applications or have an academic score of at least 27/30, and correctly completed at least 25% of the tasks in the training lab. The rest of the subjects are classified as *low-ability* subjects. In MvO, since subjects did not participate in a training session, we use these same definitions of ability levels, except for the part

[3]Although some of the used metrics are actually size metrics, we regard them as test case complexity indicators since they reflect the perceived complexity associated with usage of the test cases during debugging.
[4]In the Italian academic grade system, a score of 27/30 corresponds to a B in the ECTS grade system and to an A- in the U.S. system.

Table II. Experimental Design

	Group1	Group2	Group3	Group4
Lab 1	JTopas M	XML-Security M	JTopas A	XML-Security A
Lab 2	XML-Security A	JTopas A	XML-Security M	JTopas M

A = autogen test cases, M = manually written test cases.

about the number of correctly completed tasks. Finally, since all participants in MvE have both experience with large applications and valuable CVs, we do not consider ability levels in MvE (i.e., all subjects involved in MvE are high-ability subjects).

(2) *Subjects' experience.* We classified subjects according to four levels of experience: BSc students, MSc students, PhD/Post-docs, and researchers/professors. In MvR we used BSc and MSc students, in MvE we used PhD/Post-docs and researchers/professors, and in MvO we used MSc students only. Thus, this factor is investigated in MvR and MvE, but not in MvO. In addition to the subjects' experience, for PhD/Post-docs and researchers/professors we also considered the subjects' research *field* as a factor.
(3) *Object system (i.e., application).* Since we adopted a balanced design with two systems (see Section 3.4), subjects could show different performance on different systems. Hence the system is also a factor.
(4) *Experiment session (i.e., Lab).* We measured whether any learning effect occurred between the two labs (see Section 3.4 for a description of the counter-balanced design that we adopted).
(5) *Fault to be fixed.* Since faults are all different (a detailed description of the faults used in our experiments is provided in Appendix A), the specific features of the fault to be fixed may interact with the main factor.

For each factor, we test whether there is any effect on debugging effectiveness and debugging efficiency and check its interaction with the main factor. We formulate the following null hypotheses on the other factors.

H_{0c_i}. The factor i, $i = \overline{1..5}$, does not significantly influence effectiveness and efficiency in performing debugging tasks.

These null hypotheses are also two-tailed because we do not have any a-priori knowledge about the direction in which a factor could influence effectiveness and efficiency. Whenever a statistically significant influence exists, we also test the interaction of the factor with the main factor.

3.4. Experimental Design

We adopted a counter-balanced design: each replication of the experiments consists of two experimental sessions (*Lab 1* and *Lab 2*), with 2 hours allocated for each lab. Subjects have been split into four groups, balancing the level of ability and experience in each group. This design ensures that each subject works on the two applications (JTopas and XML-Security) and with the two different treatments (manual vs autogen test cases), as shown in Table II. Moreover, this design allows to study the effect of all the factors using statistical tests.

3.5. Experimental Procedure and Material

Before each experiment, we asked the subjects to fill a pre-questionnaire in which we collected information about their ability and experience in programming and testing. BSc students have also been trained with lectures on testing and debugging, and participated in a training laboratory where they were asked to cope with debugging tasks very similar to the experimental tasks. This made us confident that all the subjects, including BSc students, were quite familiar with the development environment and the debugging process. In the case of BSc students, their effectiveness in the training tasks has been used to assess the subjects' level of ability.

To perform the experiment, subjects used a personal computer with the Eclipse development environment equipped with a standard Java debugger. We distributed the following material.

—*The application code.* Depending on the group, either JTopas or XML-Security is used. The code contains four faults in every repetition of MvR, MvE, and MvO; subjects are told there are four faults to be debugged and that one fault-revealing test case is available per fault.
—*Fault-revealing test cases.* Either manually written or autogen fault-revealing test cases are used, depending on the group to which, the subjects belong, as shown in Table II. Each test case reveals exactly one fault. Faults are supposed to be addressed in order, and they are sorted according to difficulty, from easier to harder to fix. The difficulty of the tasks has been established in a testing session by the authors of this article. A regression test suite is provided in MvE and MvO to allow subjects to check for regressions once they have completed the fault-fixing task.
—Printed instructions describing the experimental procedure are given.

Before the experiment, we gave subjects a description of the experimental procedure, but no reference was made to the study hypotheses. Each laboratory session was carried out according to the following instructions.

(1) Import the application code into Eclipse.
(2) For each fault-revealing test case, (i) mark the start time; (ii) run the test case and use it to debug the application and fix the fault; and (iii) mark the stop time.
(3) Create an archive containing the modified source code and send it to the experimenter by email.
(4) Fill a post-experiment survey questionnaire.

During the experiment, teaching assistants were present in the laboratory to prevent collaboration among subjects and to verify that the experimental procedure was respected—in particular, that faults were addressed in the right order and that time was correctly marked.

After the experiment, subjects have been asked to fill a post-experiment survey questionnaire on the subjects' behavior during the experiment so as to find justification for the quantitative observations. The questionnaire we used for MvR consists of 17 questions related to:

Q1. adequacy of the time given to complete the tasks;
Q2. clarity of tasks;
Q3–4. difficulties experienced in understanding the source code of the application and the source code of the test cases;
Q5. difficulties in understanding the features under test;
Q6. difficulties in identifying the portion of code to change;
Q7–8. use and usefulness of the Eclipse debugging environment;
Q9. number of executions of the test case;
Q10–11. percentage of total time spent looking at the code of the test cases and of the application;
Q12. difficulties in using the test cases for debugging;
Q13. fixes of bugs achieved without fully understanding the bugs, relying just on test cases;
Q14. need for inspecting the application code to understand bugs;

Q15. perceived level of redundancy in test cases;
Q16. usefulness of local variables in test cases to understand the test;
Q17. test cases being misleading (they initially drove the subject to wrong paths in locating faults).

Answers are given on the following 5-level Likert scale [Oppenheim 1992]: *strongly agree, agree, uncertain, disagree, strongly disagree*.

After the experiment MvR, we recognized that several questions have not been useful, especially considering that the answers were formulated in a Likert scale. For the studies MvE and MvO, we thus formulated a new post-questionnaire that includes open questions to address the issues with the Likert scale. The new questionnaire still includes questions Q1, Q2, Q3, Q4, Q7, and Q8 from the previous questionnaire, but also includes two open questions:

OQ1. requesting a description of the main challenges faced during debugging;
OQ2. requesting a description of the debugging process that was followed.

In the case of MvE, since subjects consist of PhD students, Post-docs, researchers, and professors, we also added two questions aimed at determining whether the subjects already had some experience with the object applications (Question *SQ1*) and with the SIR faults (Question *SQ2*).

3.6. Seeded Faults

We seeded faults that satisfy the following requirements. First, faults are located in different parts of the object applications. Second, each fault is revealed by a manual and an autogen test case. Third, faults do not interact with each other, that is, each test case reveals a single fault. Finally, faults are based on the bugs available in the *Software-artifact Infrastructure Repository* (SIR)[5].

In MvR, eight faults have been seeded into the two applications for debugging. In order to meet the four requirements, since the autogen test cases are not able to reveal some of the SIR faults, we slightly changed these faults so that they can be detected. However, such changes do not modify the nature of the faults. An example of such changes, together with a thorough description and characterisation of the faults used in the empirical study, can be found in Appendix A.2.

In MvE, among the eight seeded faults, six have been generated by the mutation tool Jester[6], while two are in common with MvR (the other six faults used in MvR could not be revealed by any EvoSuite test, so we have replaced them). A characterisation and detailed description of the six mutants used in experiment MvE can be found in Appendix A.2.

3.7. Analysis Method

A *general linear model* (GLM) incorporates a number of different statistical models: ANOVA, ANCOVA, MANOVA, MANCOVA, ordinary linear regression, t-test, and F-test. To test the effectiveness and efficiency of subjects in performing debugging tasks (H_{01} and H_{02}) we used a general linear model. This consists of fitting a model of the dependent output variables (effectiveness and efficiency of debugging) as a function of the independent input variables (all the factors including the main factor, i.e., the kind of test cases). A general linear model allows to test the statistical significance of the influence of all factors on the effectiveness and efficiency of debugging (H_{01}, H_{02}, H_{0c_1}, H_{0c_2}, H_{0c_3}). We assume significance at 95% confidence level ($\alpha = 0.05$), so we reject the

[5] `http://sir.unl.edu`.
[6] `http://jester.sourceforge.net`.

null hypotheses having p-value<0.05. In case of relevant factors, the interpretations are formulated by visualizing interaction plots.

In case we cannot reject the null hypothesis, we risk committing a Type-II error, that is, accepting a null hypothesis that is actually false. We can estimate the probability of committing a Type-II error as $1 - Power$, where $Power$ is the statistical power of the adopted general linear model. We used a nonparametric statistical test, the Wilcoxon two-tailed paired test [Wohlin et al. 2012], to address the derived null hypotheses DH_{03} and DH_{04}. The use of nonparametric tests does not require any assumption on the normal distribution of the population. Such a test checks whether differences between different types of test cases—in terms of (i) the number of artificial/user-defined identifiers and (ii) difference in the static/dynamic complexity are statistically significant.

In order to understand whether the test case complexity is a property that characterizes the main treatments (manual vs. autogen test cases), we measured the goodness of the test case complexity metrics as predictors of the treatment. Specifically, we computed the confusion matrix where each test case complexity metric is a possible predictor and the binary classification between manual and autogen test cases is the predicted factor. Standard classification metrics (number of true/false positives/negatives) and derived metrics (precision, recall, accuracy, and F-measure) are then used to assess the degree to which manual and autogen test cases can be separated using the test case complexity as the distinguishing feature. Specifically, correct classifications are indicated as TP (true positives, i.e., correctly classified as autogen) and TN (true negatives, i.e., correctly classified as non-autogen), while errors are of two types: FP (false positives, i.e., manual test cases classified as autogen) and FN (false negatives, i.e., autogen test cases classified as non-autogen). The four derived metrics [van Rijsbergen 1979] are defined as follows: *precision* = TP / (TP + FP); *recall* = TP / (TP + FN); *accuracy* = (TP + TN) / (TP + FP + TN + FN); *F-measure* = 2 *precision* * *recall* / (*precision* + *recall*).

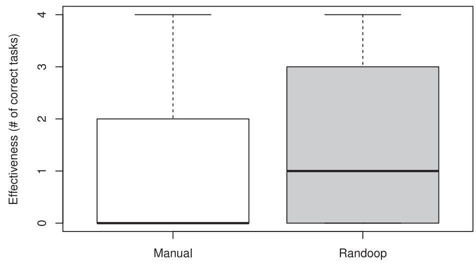
Regarding the analysis of the survey questionnaires, we evaluate the questions related to time availability, general difficulties found by subjects, and use of the debugging environment (Q1–Q8) by verifying whether the answers are either *strongly agree* (2) or *agree* (1). We test medians using a two-tailed Mann-Whitney test for the null hypothesis HQx_{05}, that is, $\widetilde{Qx} = 0$, where zero corresponds to *uncertain* and $\widetilde{Qx}$ is the median for Question Qx. The same test is also performed for Questions SQ1 and SQ2.

Among these questions, for those specific to test cases (Q4–Q8), the answers of the subjects using manually written tests are compared to those of the subjects using autogen tests. In this case a two-tailed Mann-Whitney test is used for the null hypothesis HQx_{06}, that is, $\widetilde{Q}_{autogen} = \widetilde{Q}_{Manual}$. The same comparison is also performed for the Questions Q9–Q17.

To analyze the answers to the open questions, we adopted the following process. We summarized the answers given by the subjects who worked with different treatments, and compared them to look for commonalities and differences. We grouped similar answers into common concepts and discarded unconfirmed observations, that is, those reported by just one subject. Eventually we compared the concepts emerging from answers formulated by subjects who worked with different treatments.

4. RESULTS OF MANUAL VS. RANDOOP [MVR]

This experiment compares manual and Randoop test cases [Pacheco and Ernst 2007]. It was replicated twice, the first time with 7 Masters students from University of Trento, and the second with 8 Masters and 14 Bachelors students from University of Milano-Bicocca. The data used in this section have already been reported in our



	Mean	Median	Sd
Manual tests	0.89	0.00	1.31
Randoop tests	1.65	1.00	1.50

Fig. 1. Boxplots and descriptive statistics for effectiveness (manual vs. Randoop): debugging was significantly more effective when Randoop test cases are used.

Table III. GLM Analysis of Effectiveness (Manual vs. Randoop)

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	−0.8222	0.2448	−3.36	0.0016
Treatment	0.1679	0.0795	2.11	**0.0405**
System	−0.0325	0.0799	−0.41	0.6860
Lab	0.1075	0.0819	1.31	0.1962
Experience	0.2246	0.0837	2.68	**0.0102**
Ability	0.2258	0.0676	3.34	**0.0017**

p-values in boldface indicate a statistically significant influence on effectiveness.

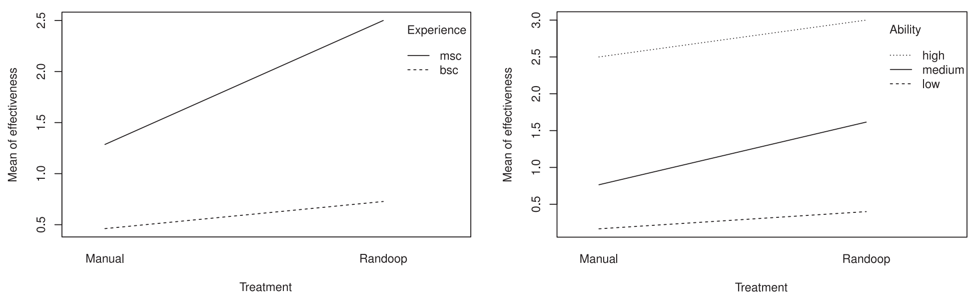
previous conference paper [Ceccato et al. 2012]. They are provided also in this article for completeness and to facilitate the comparison with the results of the experiments presented in the next two sections. Moreover, in this section we carry out a different and more detailed analysis as compared to the conference paper.

4.1. Debugging Effectiveness

Figure 1 (top) shows boxplots of the effectiveness in fault fixing. The figure compares the number of correct answers given by the subjects when the faults are debugged using either manually written or randomly generated test cases. Figure 1 (bottom) shows descriptive statistics (mean, median, and standard deviation) of effectiveness for the two distinct factors.

Table III reports the analysis of effectiveness of debugging with GLM. The model takes into account not only the effect of the main treatment (manual or autogen test cases) but all the factors that we considered in our experimental design, that is, the system, the lab, and the experience and ability of participants. Statistically significant cases are in boldface. We can observe that subjects who used autogen tests showed better effectiveness (i.e., correctly fixed more faults) than those who used manually written tests. Data confirm the trend with significance at 95% confidence level ($\alpha = 0.05$). Thus we reject H_{01R} and conclude that the debugging effectiveness is higher when debugging is supported by random tests than by manually written tests.

From Table III we can also understand the role of the other factors in influencing the main factor. Let us first consider *Ability* (*high*, *medium*, or *low*) and *Experience* (*BSc* or *MSc* student). We can notice that both Ability and Experience have a significant effect. From the interaction plots reported in Figure 2, we can notice that the high-ability and -experience subjects are associated with a line substantially higher than the line



(a) experience (b) ability

Fig. 2. Interaction plot of effectiveness between treatment (manual vs. Randoop) and experience/ability; diverging/converging lines indicate potential interactions. High-experience subjects exhibit higher effectiveness increase when using Randoop test cases as compared to low-experience subjects.

Table IV. GLM Analysis of Correctness by Treatment and Fault (Manual vs. Randoop)

(a) Jtopas

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	0.4699	0.1549	3.03	0.0029
Treatment	0.0675	0.0830	0.81	0.4172
Fault	−0.0617	0.0383	−1.61	0.1089

(b) Xml-Security

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	−0.0247	0.1450	−0.17	0.8650
Treatment	0.2710	0.0812	3.34	**0.0011**
Fault	0.0052	0.0379	0.14	0.8915

p-values in boldface indicate a statistically significant influence on correctness.

for the low-ability/-experience subjects. For what concerns experience (see Figure 2(a)), we can notice that lines are not parallel and tend to diverge instead. This indicates some interaction between experience and main treatments. Indeed, high-experience subjects improve their performance when using autogen tests much more than do lower-experience subjects. In other words, subjects with high experience are better at taking advantage of the higher effectiveness provided by autogen tests.

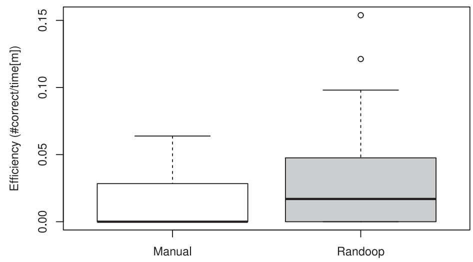
In Table III, we can notice that System and Lab are not significant factors, thus there is no effect of the system and no learning effect between the two experimental sessions.

Finally, we analyze the role of Fault as a factor to see whether the faults influenced the result or interacted with the main factor (manual versus autogen tests) to influence the result. We cannot study the impact of this factor on effectiveness, as the latter is a metrics over all faults while we are interested in each fault individually. So we resort to $Corr_i$, which measures the correctness of the fix for each i-th fault.

The analysis is performed separately for the two systems (JTopas and XML-Security) because faults are different. Results of the analysis with GLM by Treatment and Fault on Correctness do not reveal any statistically significant influence of the faults on the effectiveness of debugging (see Table IV).

4.2. Debugging Efficiency

The same procedure used with effectiveness was also applied to efficiency. Figure 3 (top) shows boxplots for efficiency with the two treatments. It compares efficiency of the subjects when the faults are debugged using either manually written or randomly



	Mean	Median	Sd
Manual tests	0.01	0.00	0.02
Randoop tests	0.03	0.02	0.04

Fig. 3. Boxplots and descriptive statistics for efficiency (manual vs. Randoop): debugging was significantly more efficient when Randoop test cases are used.

Table V. GLM Analysis of Efficiency (Manual vs. Randoop)

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	−0.0924	0.0240	−3.85	0.0004
Treatment	0.0200	0.0078	2.57	**0.0135**
System	0.0046	0.0078	0.59	0.5587
Lab	0.0100	0.0080	1.24	0.2200
Experience	0.0195	0.0082	2.38	**0.0219**
Ability	0.0182	0.0066	2.74	**0.0087**

p-values in boldface indicate a statistically significant influence on efficiency.

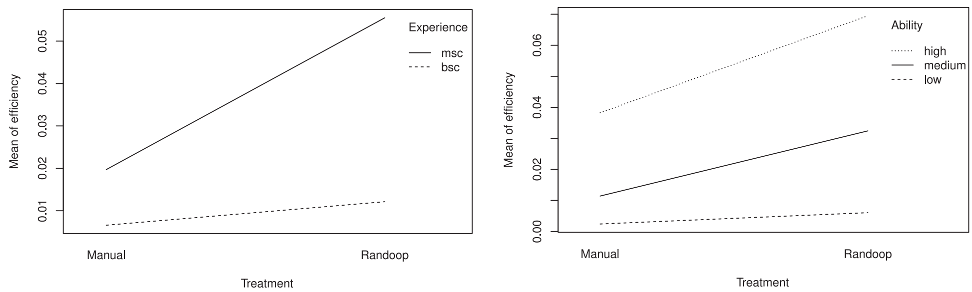
generated test cases. The corresponding descriptive statistics are reported in Figure 3 (bottom).

Table V reports the analysis with GLM. The trend shown for effectiveness is confirmed here: the efficiency of subjects working with autogen tests is higher than when working with manually written tests. Thus we reject H_{02R} and conclude that the efficiency of debugging is higher when debugging is supported by random tests than by manually written tests.

From Table V, we can also understand the role of the other factors. Both Ability and Experience have a significant effect on and interact with the main treatment. The interaction plots in Figure 4 indicate a similar effect as for the effectiveness: higher-ability/-experience subjects are particularly good at taking advantage of the higher efficiency associated with the use of autogen tests.

The factors System and Lab do not have a significant influence on efficiency of debugging.

Since we cannot study the impact of the Fault factor on efficiency as the latter is a metrics over all faults while we are interested in each fault individually, we resort to $Time_i$, which is the time taken to produce the fix for the i-th fault. The analysis is performed separately for the two systems (JTopas and XML-Security) because faults are different. Results of the analysis with GLM by Treatment and Fault on Time are significant for XML-Security, while they are close to significance but not significant (at level 0.05) for JTopas (see Table VI). Faults influence efficiency and, as apparent from the interaction plot in Figure 5, also interact with the main factor to influence Time. In fact, Fault 1 is quite different from Faults 2–4, since with Fault 1 manual and random tests are equally (in-)efficient, resulting in the highest mean fixing time, while for the other faults random tests support a more efficient debugging activity. We conclude



(a) experience (b) ability

Fig. 4. Interaction plot of efficiency between treatment (manual vs. Randoop) and experience/ability; diverging/converging lines indicate potential interactions. High-experience subjects exhibit higher efficiency increase when using Randoop test cases as compared to low-experience subjects.

Table VI. GLM Analysis of Time by Treatment (Manual vs. Randoop) and Fault

(a) Jtopas				
	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	36.8888	6.6480	5.55	0.0000
Treatment	−5.3017	3.6000	−1.47	0.1431
Fault	−3.0500	1.6777	−1.82	0.0712
(b) Xml-Security				
	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	50.6453	6.3140	8.02	0.0000
Treatment	−6.3437	3.4764	−1.82	0.0704
Fault	−9.1720	1.6638	−5.51	**0.0000**

p-values in boldface indicate a statistically significant influence on time.

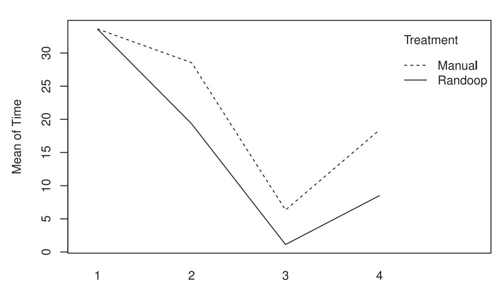


Fig. 5. Interaction plot of time between treatment (manual vs. Randoop) and fault for XML-Security. Fault 1 requires a similar amount of fixing time for both treatments, while for the other faults manual tests require more time than Randoop.

that, for faults (such as Fault 1) that are particularly difficult to debug, the choice between manual and random tests does not affect efficiency, while for normal faults (the majority of XML-Security faults, i.e., Faults 2–4) random tests are preferable.

4.3. Test Case Understandability

Table VII shows the occurrence of artificially generated and user-defined identifiers in autogen and in manual test cases. User-defined identifiers are, of course, present also

Table VII. Occurrences of Artificial/User-Defined Identifiers in the Test Cases (Manual vs. Randoop)

	Random tests		Manual tests	
	Artificial IDs	UserDef IDs	Artificial IDs	UserDef IDs
JTopas				
T1	20	4	0	36
T2	18	9	0	59
T3	19	8	0	26
T4	61	22	0	16
XML-Security				
T1	7	3	0	9
T2	63	27	0	18
T3	13	5	0	20
T4	23	7	0	21

in autogen test cases, for instance, due to names of classes instantiated or methods called in the test cases. Artificial identifiers may be present in manual test cases as well, for instance, when code generation tools (e.g., tools for parser generation from grammars) are used. This never happens in our two case studies.

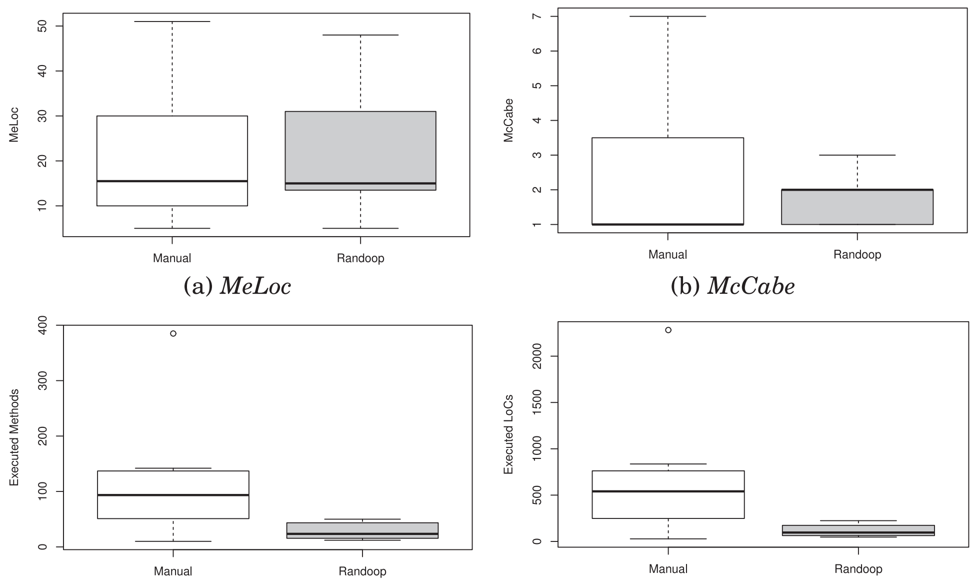
Each random test case has on average 28 artificial identifiers and the difference with manual test cases (having no artificial identifiers) is statistically significant according to the Wilcoxon two-tailed paired test. Random tests have on average 15 user-defined identifiers less than the corresponding manual tests. This difference is not statistically significant (it would be significant at level 0.10). In summary, the number of artificial (meaningless) identifiers in random test cases is substantially higher than in manual ones and the number of user-defined (meaningful) identifiers substantially smaller. The difference in identifiers does not explain the observed difference in effectiveness and efficiency, which goes in the opposite direction: random tests yield superior performance.

Figure 6 shows boxplots of the four complexity metrics for manual and autogen test cases. While the two types of tests are quite similar with respect to *MeLoc* complexity, manual tests have slightly higher *McCabe* complexity than random tests. However, none of these two metrics differ, by a statistically significant amount. The difference between test cases is more pronounced when considering dynamic complexity metrics. Manual test cases are more complex than random tests, both in *executed methods* and *executed LoCs*.

The mean number of methods (119.75) and LoCs (676.38) executed by manual test cases (see Table VIII) is substantially higher than the number of methods (28.38) and LoCs (117.00) executed by random tests. The ratio is order of two for JTopas, while even higher for XML-Security (reaching an order of magnitude when LoCs are considered). The difference between manual and random test cases is statistically significant at 95% confidence level, so we can reject the null hypothesis DH_{04} (with respect to dynamic metrics).

We also computed the confusion matrix (see Table IX) associated with a nearest-neighbor classifier that predicts the test case type based on one of the two dynamic complexity metrics (either the executed methods or LoCs). The predictor classifies a new test case by determining the available test case having the closest dynamic complexity metrics value and assigning it to the class (autogen or manual) of such closest test case [Cover and Hart 1967]. The confusion matrix was obtained by applying onefold cross-validation.

Executed LoCs is a better predictor than executed methods. The values reported in Table X (bottom) for this predictor are quite close to 1, showing that in our experiment it is possible to predict the type of a test case from its dynamic complexity metrics (specifically, executed LoCs) with high accuracy. This means that the autogen and



(a) *MeLoc* (b) *McCabe*

(c) executed methods (d) executed LoCs

Fig. 6. Boxplots of test case size and complexity (manual vs. Randoop). The figures suggest that manual and Randoop are comparable in terms of *MeLoc* and *McCabe*. With respect to executed methods and LoCs, manual is more complex.

Table VIII. Descriptive Statistics and Paired Analysis (Wilcoxon's Test) of Static (top) and Dynamic (bottom) Test Case Metrics: Manual (M) vs. Randoop (R)

Metric	N	R.mean	R.sd	M.mean	M.sd	diff.mean	diff.median	diff.sd	p.value
MeLoc	8	21.50	14.98	20.88	15.42	0.62	0.50	21.12	1.00
McCabe	8	1.75	0.71	2.38	2.20	−0.62	0.00	2.07	0.59
Methods	8	28.38	15.50	119.75	116.97	−91.38	−65.00	120.77	**0.04**
LOCs	8	117.00	65.39	676.38	707.53	−559.38	−416.50	725.33	**0.04**

Manual tests are substantially more complex than Random tests according to the dynamic metrics Methods and LoCs.

Table IX. Nearest Neighbor Classifier Prediction of the Test Case Type (Manual vs. Randoop) based on Executed Methods or LoCs

	Randoop		Manual	
Metric	TP	FN	FP	TN
Exec. Methods	6	2	2	6
Exec. LOCs	8	0	2	6

manual test cases used in the experiment can be characterized with a high accuracy, respectively, as *low dynamic complexity* and *high dynamic complexity* test cases.

In summary, manual test cases are dynamically more complex than random test cases. This might explain the observed performance degradation exhibited by subjects working with manual test cases, despite the presence of more meaningful identifiers in these test cases.

Table X. Prediction Performance Metrics for the Nearest-Neighbor Classifier Using Executed Methods or LoCs

Metric	Precision	Recall	Accuracy	F.measure
Exec. Methods	0.75	0.75	0.75	0.75
Exec. LOCs	0.80	1.00	0.88	0.89

The test case type is Manual or Randoop.

Table XI. Analysis of Post-Questions Q1–Q8

	Low ability		High ability		All	
Question	median	p.value	median	p.value	median	p.value
Q1: Enough time	not certain	0.59	agree	**0.01**	not certain	0.10
Q2: Tasks clear	not certain	0.05	strongly agree	**0.01**	agree	<**0.01**
Q3: No difficulty in understand application code	not certain	0.37	not certain	0.23	not certain	0.98
Q4: No difficulty in understand test code	not certain	1.00	not certain	1.00	agree	**0.03**
Q5: Easily understand the feature under test	not certain	0.85	not certain	0.37	not certain	0.30
Q6: No difficulty in identifying where to fix	not certain	0.19	not certain	0.07	not certain	0.32
Q7: Used the Eclipse debugger	not certain	0.30	strongly agree	**0.01**	agree	**0.01**
Q8: Debugger was useful	agree	0.41	agree	**0.01**	agree	<**0.01**

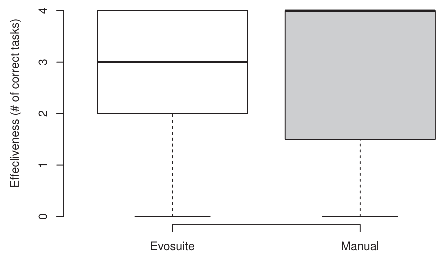
Mann-Whitney test for the null hypothesis $median(Qx) = 0$ (manual vs. Randoop); p-values in boldface indicate that the result is statistically significant at level 0.05.

4.4. Analysis of Post-Questionnaire

We used the answers to the questions from Q1–Q8 to gain insights into subjects' activity. Results are summarized in Table XI. Considering data over all the replications, answers to Questions Q2, Q4, Q7, and Q8 produced statistically significant results (p-value < 0.05), while answers to the other questions are not statistically significant. Subjects found the tasks to be clear ($\widetilde{Q2} = 1$, i.e., *agree*, with p-value < 0.01), and overall, they had no difficulty in understanding the source code of the test cases (Q4). The debugger was used (Q7) only by high-ability subjects, although it was judged useful (Q8) by all.

Then, we compared answers for the questions specific to test cases Q9–Q17 to understand whether any statistical difference can be observed between subjects who worked with manually written test cases and those who used autogen ones. The unpaired Mann-Whitney test never reported statistical significance, so we omit the table (it can be found in the technical report [Ceccato et al. 2013]).

Let us now analyze the differences between answers given by the low- and by the high-ability subjects (see Table XI, columns 2–3, 4–5). According to the post-questionnaire, there is a remarkable difference in the use of the Eclipse debugger (Q7) between the low- and high-ability subjects in that only the latter declare to have used it extensively. This might be part of the explanation for the gap between manual and random test cases observed in both effectiveness and efficiency (see Figures 1 and 3). Without the debugger, low-ability subjects could take advantage only of simple test cases (i.e., those generated by Randoop) while they could not manage the complexity of most manual test cases, resulting in lower performance in the latter case. On the contrary, high-ability subjects, who used the debugger more extensively, were able to take advantage also of the complex test scenarios. Of course, they also had better performance with the simpler, random tests.



	Mean	Median	Sd
Manual tests	2.73	4.00	1.58
EvoSuite tests	2.73	3.00	1.28

Fig. 7. Boxplots and descriptive statistics for effectiveness (manual vs. EvoSuite): debugging was equally effective with manual and EvoSuite test cases.

Differently from the low-ability subjects, high-ability subjects considered the debugger useful (Q8), which is consistent with the extensive use of the Eclipse debugger (Q7) reported only by the high-ability subjects. Another difference is that time was regarded as sufficient to complete the debugging task (Q1) and tasks were regarded as clear (Q2) by high-ability subjects, while this was not the case for low-ability subjects.

5. RESULTS OF MANUAL VS. EVOSUITE [MVE]

This experiment compares manual and autogen test cases, but considers a different test case generation algorithm (i.e., EvoSuite [Fraser and Arcuri 2011]) and subjects with higher experience. It was also replicated twice, the first time with six professors/Post-docs from University of Milano-Bicocca and the second time with nine researchers/Post-docs from Fondazione Bruno Kessler. Participants were asked to locate and fix faults, supported by either: (i) manually written test cases or (ii) test cases generated by EvoSuite.

5.1. Debugging Effectiveness

Figure 7 shows boxplots of the effectiveness in fault fixing. The median of effectiveness when debugging with EvoSuite tests is lower than when manual tests are used, but the overall distribution of effectiveness is very similar. Figure 7 (bottom) reports the corresponding descriptive statistics.

Table XII reports the analysis with GLM to study the influence of the main factor (i.e., the treatment) and of other factors on the effectiveness of debugging. The statistical test reports a p-value>0.05. Thus we cannot reject the null hypothesis H_{01E}, stating that there is no difference in effectiveness of debugging when debugging is supported either by manually written or EvoSuite test cases. The probability of a Type-II error (accepting a false null hypothesis), obtained from GLM power analysis, is 1%.

Table XII also reveals the role of the factors in influencing the dependent variable, that is, the debugging effectiveness.

As first factor we studied whether the Experience of participants (PhD student/Post-doc or researchers/professors) influenced the results. Differently from the MvR experiment, we can notice that experience did not influence the effectiveness of debugging. Then, we considered whether the research Field of participants (software testing or a different field) influenced the results. Also in this case, large p-values bring us to accept the null hypothesis (non-influence of the research field).

We considered whether the particular System used in the experimental sessions (JTopas or XML-Security) influenced the result. Differently from the MvR experiment,

Table XII. GLM Analysis of Effectiveness (Manual vs. EvoSuite)

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	1.4103	0.3845	3.67	0.0012
Treatment	−0.0264	0.1128	−0.23	0.8167
Experience	−0.1538	0.1279	−1.20	0.2408
Field	0.0962	0.1279	0.75	0.4595
System	−0.4014	0.1128	−3.56	**0.0016**
Lab	0.0048	0.1128	0.04	0.9664

p-values in boldface indicate a statistically significant influence on effectiveness.

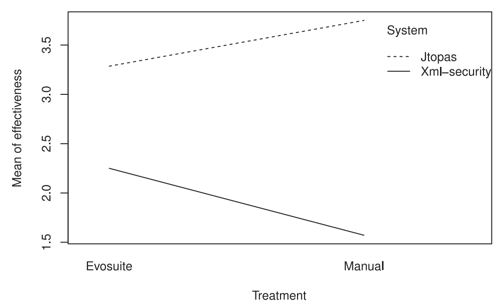


Fig. 8. Interaction plot of effectiveness between treatment (manual vs. EvoSuite) and system; diverging/converging lines indicate potential interactions. XML-Security is more difficult to debug than JTOpas and even more so when manual test cases are used.

we can notice that the subject system had a significant effect in influencing the effectiveness (p-value<0.05). Figure 8 shows an interaction plot of Effectiveness between Treatment and System. We can notice that, when working on XML-Security, participants had on average a lower effectiveness than when working on JTopas, both with manual and with EvoSuite tests. Moreover, such gap is amplified when manual tests are used, hence indicating a level of interaction with the main factor. XML-Security was more difficult to debug and the associated difficulty was further increased when manual tests were available for the debugging task.

We analyze the learning effect by studying the Lab factor. We can notice (see Table XII) that the lab did not influence the effectiveness. The last factor that we consider is the Fault. We analyze whether the faults influenced the result and whether they interacted with the main factor to influence the result. We adopt the same analysis procedure that was applied for the MvR experiment (i.e., GLM for $Corr_i$ for each i-th fault).

Table XIII reports the results of GLM for Correctness by Treatment and Fault. There is a statistically significant influence of Fault only for XML-Security.

By looking at the interaction plot shown in Figure 9, we can notice that, overall, use of autogen tests increases the proportion of correctly executed debugging tasks as compared to the use of manual ones, but such improvement is fault specific. For Fault 1 the improvement is marginal; for Faults 2–3 there is a remarkable improvement; Fault 4 is a case where the difference between autogen and manual tests is substantial. On Fault 4 the mean correctness of debugging is more than doubled when autogen tests are used.

5.2. Debugging Efficiency

A similar procedure was used to study the efficiency of debugging. Figure 10 shows boxplots of the efficiency with the two alternative treatments. The picture reveals no

Table XIII. GLM Analysis of Correctness by Treatment and Fault (Manual vs. EvoSuite)

(a) Jtopas

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	0.7887	0.1632	4.83	0.0000
Treatment	0.1161	0.0832	1.39	0.1686
Fault	−0.0333	0.0371	−0.90	0.3733

(b) Xml-Security

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	1.0155	0.2412	4.21	0.0001
Treatment	−0.1696	0.1264	−1.34	0.1847
Fault	−0.1133	0.0564	−2.01	**0.0492**

p-values in boldface indicate a statistically significant influence on correctness.

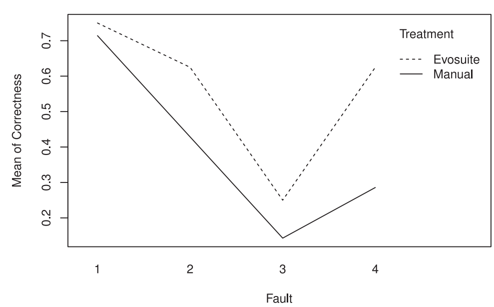
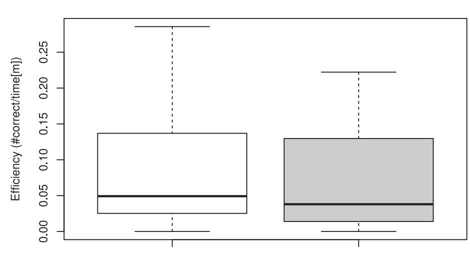


Fig. 9. Interaction plot of correctness between treatment (manual vs. EvoSuite) and fault on XML-Security; diverging/converging lines indicate potential interactions.



	Mean	Median	Sd
Manual tests	0.07	0.04	0.07
EvoSuite tests	0.10	0.05	0.10

Fig. 10. Boxplots and descriptive statistics for efficiency (manual vs. EvoSuite): debugging was equally efficient with manual and EvoSuite test cases.

clear trend. The efficiency with EvoSuite is, on average, higher than with manual tests, but the distribution is similar.

Table XIV shows the result of analysis with GLM. Statistical significance is not reached (p-value>0.05), so we cannot reject the null hypothesis H_{02E}, stating that there is no difference in the efficiency of debugging when debugging is supported either by manual or by EvoSuite test cases. The probability of a Type-II error (accepting a false null hypothesis), obtained from GLM power analysis, is 1%.

Table XIV. GLM Analysis of Efficiency (Manual vs. EvoSuite)

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	0.2978	0.0810	3.68	0.0012
Treatment	−0.0294	0.0238	−1.24	0.2273
Experience	−0.0025	0.0269	−0.09	0.9269
Field	−0.0096	0.0269	−0.36	0.7252
System	−0.1208	0.0238	−5.08	**0.0000**
Lab	0.0213	0.0238	0.90	0.3789

p-values in boldface indicate a statistically significant influence on efficiency.

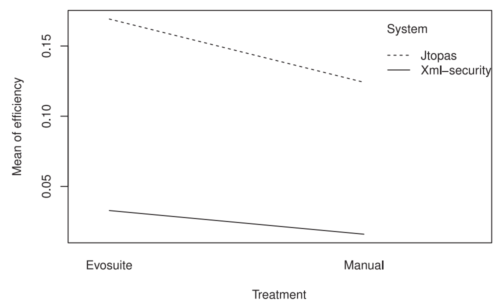


Fig. 11. Interaction plot of efficiency between treatment (manual vs. EvoSuite) and system; diverging/converging lines indicate potential interactions. XML-Security is more difficult to debug than JTOpas; both become more difficult to debug when manual test cases are used.

In summary, EvoSuite test cases are as good as manual in supporting debugging in terms of both effectiveness and efficiency.

From Table XIV, we can see that factors Experience, Field, and Lab did not influence the results. As with effectiveness, we can notice that the factor System has a significant effect on efficiency (p-value<0.05). Figure 11 shows an interaction plot of Efficiency between Treatment and System. Similarly to the interaction plot for the effectiveness (see Figure 8), when working on XML-Security participants had, on average, a lower efficiency than when working on JTopas, both with manual and with EvoSuite tests. Moreover, the availability of manual tests further reduces debugging efficiency on both JTopas and XML-Security, hence revealing some interaction with the main treatment.

For the factor Fault, we apply GLM to estimate $Time_i$ for each i-th fault. Table XV reports the results of GLM for Time by Treatment and Fault. No statistically significant influence of Fault on Efficiency was reported by the statistical test.

5.3. Test Case Understandability

Table XVI reports the number of artificial/user-defined identifiers in EvoSuite test cases (second and third columns) and in manual test cases (fourth and fifth columns). This table differs from Table VII because Randoop and EvoSuite reveal different faults and, correspondingly, different manual test cases are used in experiments MvR and MvE. No artificial identifiers are present in manual test cases, while EvoSuite tests contain both artificial and user-defined identifiers.

Table XVII reports results of the Wilcoxon test on the identifiers. EvoSuite test cases have on average 3 artificial identifiers and manual tests have on average 12 user-defined identifiers more than EvoSuite tests. This difference is statistically significant according to the results of the Wilcoxon test (p-value<0.05). So we can reject the null hypothesis DH_{03E} and can formulate the alternative hypothesis that the number of

Table XV. GLM Analysis of Time by Treatment (Manual vs. EvoSuite) and Fault

(a) Jtopas

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	6.7574	5.5761	1.21	0.2306
Treatment	1.8973	2.8439	0.67	0.5074
Fault	−0.3333	1.2690	−0.26	0.7937

(b) Xml-Security

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	17.3869	6.4545	2.69	0.0093
Treatment	7.2589	3.3810	2.15	**0.0361**
Fault	−2.3333	1.5087	−1.55	0.1275

p-values in boldface indicate a statistically significant influence on time.

Table XVI. Occurrences of Artificial/User-Defined Identifiers in the Test Cases (Manual vs. EvoSuite)

	EvoSuite tests		Manual tests	
	Artificial ID	UserDef ID	Artificial ID	UserDef ID
		JTopas		
T1	2	5	0	27
T2	3	7	0	18
T3	3	6	0	15
T4	2	5	0	18
		XML-Security		
T1	4	10	0	26
T2	3	8	0	27
T3	3	9	0	9
T4	1	6	0	12

Table XVII. Paired Analysis of Artificial/User-Defined Identifiers (Wilcoxon's test)

ID type	N	M.mean	M.sd	E.mean	E.sd	diff.mean	diff.median	diff.sd	p.value
Artificial ID	8	0.00	0.00	2.62	0.92	−2.62	−3.00	0.92	**0.01**
UserDef ID	8	19.00	7.01	7.00	1.85	12.00	12.00	7.13	**0.02**

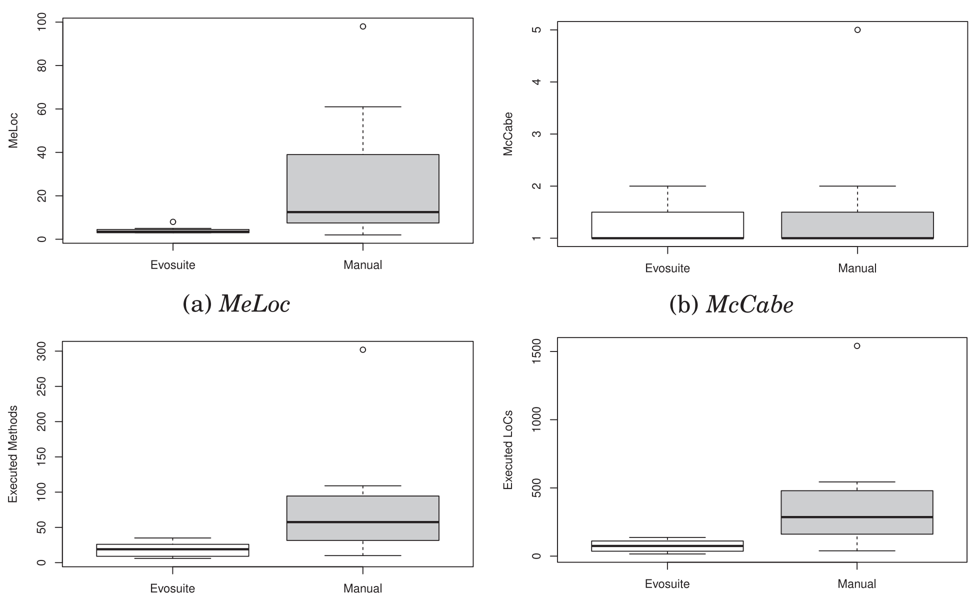
Manual (M) vs. EvoSuite (E); *p*-values in boldface indicate a statistically significant difference between manual and EvoSuite tests.

meaningless (artificial) identifiers in EvoSuite tests is higher than in manual tests and the number of meaningful (user-defined) identifiers is smaller.

Figure 12 compares the complexity metrics computed on EvoSuite and manual test cases. While the two types of tests are very similar with respect to *McCabe* complexity, manual tests have higher *MeLoc*. Manual tests have also higher dynamic complexity than EvoSuite tests in terms of both *executed methods* and *executed LoCs*.

These trends are confirmed by the results of the Wilcoxon test reported in Table XVIII. While there is no statistically significant difference on *McCabe* complexity, differences in all the other (static and dynamic) metrics are statistically significant. So we can reject the null hypothesis DH_{04E} and formulate the alternative hypothesis that static (MeLoC) and dynamic complexity of manually written test cases is significantly higher than in EvoSuite test cases.

In summary, as observed in the case of Randoop test cases, manual tests on the one hand contain a higher number of meaningful identifiers than EvoSuite tests. On the other hand, manual tests are more complex than EvoSuite test cases. Differently from



(a) *MeLoc* (b) *McCabe*

(c) executed methods (d) executed LoCs

Fig. 12. Boxplots of test case size and complexity (manual vs. EvoSuite). The two types of tests are comparable with respect to *McCabe* complexity. Manual tests have higher *MeLoc*, executed methods, and executed LoCs.

Table XVIII. Descriptive Statistics and Paired Analysis (Wilcoxon's test) of Static (top) and Dynamic (bottom) Test Case Metrics

Metric	N	M.mean	M.sd	E.mean	E.sd	diff.mean	diff.median	diff.sd	p.value
MeLoc	8	27.25	34.03	4.12	1.73	23.12	7.00	34.46	**0.02**
McCabe	8	1.62	1.41	1.25	0.46	0.38	0.00	1.60	0.85
Methods	8	84.88	93.07	18.62	10.70	66.25	27.00	91.14	**0.01**
LoCs	8	429.12	477.50	74.50	43.94	354.62	161.50	473.89	**0.01**

Manual (M) vs. EvoSuite (E): Manual tests are substantially more complex than EvoSuite tests according to the static metrics *MeLoC* and the dynamic metrics Methods and LoCs.

the MvR experiment, in this experiment the lower complexity of EvoSuite test cases is not associated with a higher effectiveness or efficiency of debugging. At the same time, the reduced understandability of EvoSuite tests does not make them less effective than manual test cases during debugging in terms of effectiveness and efficiency.

5.4. Analysis of Post-Questionnaire

Results of the Mann-Whitney test for Questions Q1–Q4 are shown in Table XIX. Statistical significance is observed for Questions Q1, Q2, and Q4, but not for Q3. Participants *strongly agree* that they had enough time to complete the tasks and that tasks were clear. While they agree that understanding the application is required to complete the tasks, they are not certain that understanding the test cases is also required.

Questions Q7 and Q8 deal with the used (Q7) and most used (Q8) features of the IDE. All the features mentioned in the questionnaire have been used, with debugger and code navigation reported as the most used features.

Table XIX. Analysis of Post-Questions Q1–Q4

Question	Median	P-value
Q1: Enough time	strongly agree	<**0.01**
Q2: Tasks clear	strongly agree	<**0.01**
Q3: Test case understanding required	not certain	0.68
Q4: Application understanding required	agree	<**0.01**

Mann-Whitney test for the null hypothesis $median(Qx) = 0$ (manual vs. EvoSuite); p-values in boldface indicate that the result is statistically significant at level 0.05.

Participants *never* used either JTopas or XML-Security before the experiment (Question SQ1), and *never* used the faults from the SIR repository related to these two applications (SQ2).

Questions OQ1 and OQ2 deal with the main challenges faced during debugging and with the followed process, respectively. The most frequently reported challenge during debugging (OQ1) is understanding the application code, both when using manual and EvoSuite tests. Only 2 participants indicated understanding the test cases as a challenge, which corroborates our interpretation of the answers to Q3.

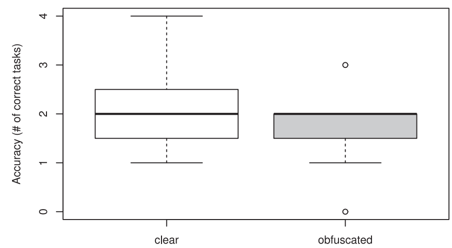
The strategy adopted to fix faults (OQ2) does not show relevant differences when using manual or EvoSuite test cases. Interestingly, understanding the test cases is not among the actions taken by participants to locate and fix the faults (still in line with Q3).

According to the post-questionnaire, the code of the test cases does not necessarily have to be understood in the opinion of the expert subjects involved in MvE, while it is supposed to be thoroughly understood according to the less experienced subjects involved in MvR. This could explain the different results obtained in the two studies. The analysis of the manual tests, which are more complex than Randoop and EvoSuite tests, is hard and takes time. The less experienced subjects performed better with Randoop tests because they are simpler to understand as compared to the manual tests. The expert subjects involved in MvE, who did not spend much time understanding the tests, performed equally well with EvoSuite and manual tests, despite the higher complexity of the latter. On the other hand, deep understanding of the application logic was reported as the major challenge by many subjects when using either random and EvoSuite tests. Another remarkable difference is in the use of the debugger and of the code navigation functionalities offered by the IDE. Differently from experiment MvR, the expert subjects involved in MvE made extensive use of the IDE, in particular debugging and code navigation functionalities.

6. RESULTS OF MANUAL VS. OBFUSCATED [MVO]

Since MvR and MvE indicate that the presence of meaningful identifiers and the fact that tests are manually created with a specific intent in mind are not relevant for debugging, we designed this experiment to specifically investigate the impact of identifiers on the debugging activity. This study was conducted involving 11 MSc students from the University of Milano-Bicocca. Participants have been asked to locate and fix faults, supported by: (i) manually written test cases, or (ii) manually written test cases with obfuscated identifiers.

We produced the obfuscated test cases from the manually written ones by changing the name of every local variable and method parameter into `x` followed by an incremental number, and the name of every class attribute into `y` followed by an incremental number. In this way no variable or parameter has a name that describes the semantics of the value it stores.



	Mean	Median	Sd
Manual tests	2.18	2.00	1.08
Obfuscated tests	1.75	2.00	0.89

Fig. 13. Boxplots and descriptive statistics for effectiveness (manual vs. obfuscated): debugging was equally effective with manual and obfuscated test cases.

Table XX. GLM Analysis of Effectiveness (Manual vs. Obfuscated)

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	0.6641	0.4319	1.54	0.1464
Treatment	−0.0348	0.1202	−0.29	0.7763
Ability	−0.0472	0.0994	−0.48	0.6420
System	−0.1598	0.1202	−1.33	0.2049
Lab	0.1371	0.1274	1.08	0.3002

No factor has a statistically significant (p-value < 0.05) influence on effectiveness.

6.1. Debugging Effectiveness

Figure 13 shows boxplots of the effectiveness in fault fixing. The figure compares the number of correctly fixed faults when participants worked with the original, manually written test cases (indicated as *clear* boxplot) to the number of correct fixes when working with obfuscated test cases (indicated as *obfuscated* boxplot). The effectiveness of clear tests is sometimes higher than that of obfuscated tests. However, the two distributions have the same median, suggesting that the type of the identifier (clear or obfuscated) has no impact on the effectiveness of debugging tasks.

Table XX shows the GLM for effectiveness when clear and obfuscated test cases are used. Statistical significance is not reached (p-value is not <0.05), that is, we cannot reject the null hypothesis: there is no difference in the effectiveness of debugging when debugging is supported by manually written test cases, either with clear or obfuscated identifiers. The probability of a Type-II error in this claim is 23%; this relatively high value is due to the low number of subjects (11) and the high dispersion of data.

For what concerns the other factors, we can notice that none of them had a statistically significant influence on effectiveness (see Table XX), with the exception of faults for XML-Security (see Table XXI).

By looking at the interaction plot shown in Figure 14, we can notice that some faults are fixed equally well on clear and on obfuscated code (Faults 3–4), while on other faults (1–2) having either clear or obfuscated code is slightly preferable.

6.2. Debugging Efficiency

Figure 15 shows boxplots of efficiency when the faults are debugged using manually written test cases, either with clear or obfuscated identifiers. Also in this case the efficiency of clear tests is sometimes higher than the efficiency of obfuscated. However,

Table XXI. GLM Analysis of Correctness by Treatment (Manual vs. Obfuscated) and Fault

(a) Jtopas

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	0.9743	0.3043	3.20	0.0028
Treatment	−0.2171	0.1730	−1.25	0.2173
Fault	−0.0160	0.0782	−0.20	0.8388

(b) Xml-Security

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	1.0911	0.3292	3.31	0.0029
Treatment	−0.0050	0.1664	−0.03	0.9764
Fault	−0.2877	0.0837	−3.44	**0.0021**

p-values in bold face indicate a statistically significant influence on correctness.

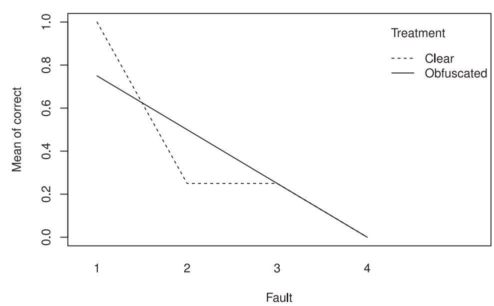
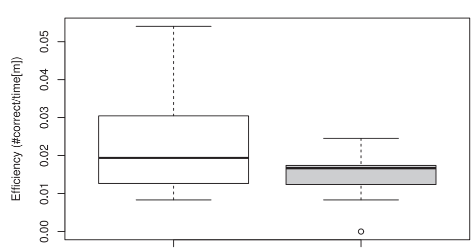


Fig. 14. Interaction plot of correctness between treatment (manual vs. EvoSuite) and fault on XML-Security; diverging/converging lines indicate potential interactions.



	Mean	Median	Sd
Manual tests	0.02	0.02	0.01
EvoSuite tests	0.01	0.02	0.01

Fig. 15. Boxplots and descriptive statistics for efficiency (manual vs. obfuscated): debugging was equally efficient with manual and obfuscated test cases.

the two distributions have similar medians, suggesting that the type of the identifier (clear vs. obfuscated) has no impact on the efficiency of debugging.

Table XXII reports the GLM for efficiency. The test did not reach statistical significance, so also for efficiency we cannot reject the null hypothesis: there is no difference in the efficiency of debugging when debugging is supported by manually written test cases, either with clear or obfuscated identifiers. The probability of a Type-II error is 13%.

Table XXII. GLM Analysis of Efficiency (Manual vs. Obfuscated)

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	0.0224	0.0207	1.08	0.2988
Treatment	−0.0041	0.0058	−0.70	0.4931
Ability	−0.0018	0.0048	−0.38	0.7088
System	−0.0053	0.0058	−0.91	0.3769
Lab	0.0094	0.0061	1.54	0.1470

No factor has a statistically significant (p-value < 0.05) influence on efficiency.

Table XXIII. GLM Analysis of Time by Treatment (Manual vs. Obfuscated) and Fault

(a) Jtopas

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	32.3684	10.2913	3.15	0.0032
Treatment	1.8230	5.8527	0.31	0.7571
Fault	−2.5308	2.6453	-0.96	0.3448

(b) Xml-Security

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	18.1722	15.0818	1.20	0.2400
Treatment	7.8179	7.6261	1.03	0.3155
Fault	1.9106	3.8342	0.50	0.6228

No factor has a statistically significant (p-value < 0.05) influence on time.

For what concerns the other factors, we can notice that none of them had a statistically significant influence on effectiveness, including the faults (see Tables XXII and XXIII).

6.3. Test Case Understandability

In Sections 4 and 5, we have found no significant difference in the effectiveness or efficiency of debugging when comparing Randoop/EvoSuite against manual tests. In Section 6, identifier names alone (clear vs. obfuscated) have been found to be insufficient to explain the lack of apparent difference between autogen and manual tests. We now investigate whether the understandability of the obfuscated and the clear tests that we used in the study was indeed significantly different.

Even if we renamed the identifiers in the tests, references to external entities, such as library or application methods called from the test cases, are not changed, so meaningful identifiers are still present in the obfuscated test cases. Table XXIV reports the number of the obfuscated and user-defined identifiers for obfuscated test cases in the second and third columns, and the number of obfuscated and user-defined identifiers for clear test cases in the fourth and fifth columns. Table XXV reports the results of the Wilcoxon test on identifiers. The results confirm the impact of obfuscation: the difference in the number of the obfuscated identifiers (first row) and of user-defined identifiers (second row) between obfuscated and clear test cases is statistically significant (p-value $= 0.01$). We can thus reject DH_{03O}.

Since the obfuscation process does not alter the static/dynamic metrics, DH_{04O} is not applicable in this study.

In summary, the use of test cases that are significantly more difficult to understand than manually written test cases did not result in any major difference in the effectiveness and efficiency of debugging.

Table XXIV. Occurrences of Obfuscated/User-Defined Identifiers in the Test Cases (Manual vs. Obfuscated)

	Obfuscated tests		Manual tests	
	Obfuscated IDs	UserDef IDs	Obfuscated IDs	UserDef IDs
		JTopas		
T1	9	35	0	44
T2	12	31	0	43
T3	5	13	0	18
T4	6	18	0	24
		XML-Security		
T1	3	15	0	18
T2	8	18	0	26
T3	6	25	0	31
T4	5	12	0	16

Table XXV. Paired Analysis (Wilcoxon's test) of Static (top) and Dynamic (bottom) Test Case Metrics (Manual vs. Obfuscated vs. User-Defined)

Id type	N	diff.mean	diff.median	diff.sd	P-value
Obfuscated	8	6.75	6.00	2.82	**0.01**
UserDef	8	−6.62	−6.00	2.92	**0.01**

Table XXVI. Analysis of Post-Questions Q1–Q4

Question	Median	P-value
Q1: Enough time	not certain	0.85
Q2: Tasks clear	agree	0.10
Q3: Test case understanding required	agree	**0.04**
Q4: Application understanding required	agree	<**0.01**

Mann-Whitney test for the null hypothesis $median(Qx) = 0$ (manual vs. obfuscated); p-values in boldface indicate that the result is statistically significant at level 0.05.

6.4. Analysis of Post-Questionnaire

Results of Questions Q1–Q4 are summarized in Table XXVI. Questions Q1–Q2 do not have statistical significance, while Questions Q3–Q4 do (p-value < 0.05). Participants (MSc students) *agree* that test case and application code comprehension are required to complete the debugging tasks.

We compared the answers of participants who worked with clear test cases with those of participants who worked with obfuscated test cases. No question shows any statistically significant difference.

Questions Q7 and Q8 investigate what features of the IDE have been used to complete the debugging task and which was the most useful. The debugger and code navigation facilities were the most used features. For each feature, we used the Fisher's exact test to compare the answers given when using clear tests with those given when using obfuscated tests. No statistically significant difference is observed.

Questions OQ1 and OQ2 are open questions, meant to let participants formulate free comments on how they faced the debugging tasks. The main challenges (OQ1) for the subjects who worked with both clear and obfuscated tests are comprehension of the application code and domain, the lack of comments, and lack of a general overview of the application architecture. It is interesting to notice that the presence of meaningless identifiers in obfuscated test cases is not mentioned as a challenge. The steps followed by participants to fix the faulty code (OQ2) are quite standard and the same with either clear or obfuscated identifiers.

Summarizing the results of the post-questionnaire, participants agree that code (both test code and application code) understanding was very important for the successful completion of the debugging tasks. This is consistent with the results reported for MvR, where the subjects (MSc and BSc students) indicated understanding test cases as a relevant factor. However, obfuscation of the identifiers does not seem to be a barrier to the usability of the test cases in debugging. This is also in agreement with the observation (see experiments MvR and MvE) that static, and even moreso dynamic, complexity metrics better reflect the difficulty of debugging, as compared to understanding the tests. The answers to the open questions of the post-questionnaire support this finding: the presence of meaningless identifiers was not reported as a challenge, while comprehension of code, domain, and overall architecture has been reported as the major difficulty encountered while executing the debugging tasks.

7. DISCUSSION

In this section we report the findings (*Find*) that we derived from our experiments. Each finding is summarized with one sentence, followed by a summary of the piece of evidence that supports the finding. We discuss how we interpreted the piece of evidence and present a list of the practical implications generated by the finding.

Find1. Meaningfulness of test case identifiers does not affect debugging effectiveness and efficiency.

Pieces of Evidence

—Test cases generated with EvoSuite and Randoop include meaningless identifiers (see the analysis reported in Sections 4.3 and 5.3).

—The presence of meaningless identifiers did not negatively affect effectiveness and efficiency (see the analysis about effectiveness reported in Sections 4.1, 5.1, and 6.1, and the analysis about efficiency reported in Sections 4.2, 5.2, and 6.2).

—Experienced subjects did not spend time understanding the purpose of the test cases, either manual or autogen (see the analysis of the post-questionnaire reported in Section 5.4).

Interpretation. Manual test cases are implemented with a specific intent in mind so as to exercise a meaningful and representative test scenario. Moreover (in our experiments), manual tests include mostly user-defined identifiers. On the contrary, the autogen test cases generated by Randoop or EvoSuite do not explicitly cover any meaningful testing scenario and include a large number of artificially generated identifiers. Obfuscated test cases are still associated with a meaningful test scenario because they are derived from manual test cases, but they include a substantial number of artificial, meaningless identifiers. All these differences do not result in superior debugging performance of subjects using manual test cases. We conjecture that the presence of meaningless identifiers in autogen tests is not an influential factor because such identifiers appear only when debugging the top-level methods in a test execution (i.e., the test methods). Below such top level, identifiers are perfectly understandable and meaningful. Moreover, any difficulty in interpreting a test method due to its identifiers does not matter as long as the test reveals a fault. Subjects (in particular, subjects with higher experience such as PhD/Post-docs and researchers/professors) did not even attempt to attribute any intent to autogen tests. They did not spend any time understanding the purpose of the test cases, rather they focused on understanding the bug and the application code.

Practical Implications

Imp1.1. Since lack of meaningful identifiers does not affect debugging performance, while the simplicity of autogen tests can ease debugging, developers should consider testing components with autogen tests first to quickly rule out the faults that can be addressed with automatic tools, and then design the manual tests to reveal the other faults that cannot be addressed with autogen tests.

Find2. Test case complexity affects debugging effectiveness and efficiency of less experienced subjects.

Pieces of Evidence

—Autogen test cases are simpler than manual ones (see static and dynamic test case complexity in Sections 4.3 and 5.3).
—Less experienced subjects performed better with autogen test cases than with manual (see the analysis of efficiency and effectiveness in Sections 4.1 and 4.2).
—Experienced subjects performed almost equally well with autogen and with manual test cases (see the analysis of effectiveness and efficiency in Sections 5.1 and 5.2).
—Less experienced subjects tried to understand the purpose of the analyzed tests (see the answers to the post-questionnaire in Section 4.4).
—Test case complexity is not reported as a meaningful factor for debugging (see the answers to the post-questionnaire in Section 5.4).

Interpretation. Manual test cases exercise complex, long execution scenarios that would require substantial effort to be fully understood. Autogen test cases are simple, short linear sequences of method invocations. In experiment MvR involving less experienced subjects (BSc and MSc students), this difference provides an explanation for the superior performance of autogen test cases. In fact, the (less experienced) subjects involved in this experiment report a substantial effort devoted to test case understanding, which is a major obstacle with manual test cases. On the contrary, the experienced subjects involved in experiment MvE (PhD/Post-docs and researchers/professors) performed equally well with manual and autogen test cases, showing that for experienced testers the complexity of the test cases is not a relevant obstacle. This is confirmed by their answers to the post-questionnaire in which test case complexity is never mentioned as a major factor affecting the debugging process.

Practical Implications

Imp2.1. Less experienced developers should debug the failures produced by autogen tests and simple manual tests before being allocated to the debugging of complex manual test cases.

Find3. Ability and experience are key factors affecting the debugging performance.

Pieces of Evidence

—The low-ability students had a hard time with both the debugging process and debugging tools (see our analysis of the post-questionnaire in Section 4.4), and experienced difficulties in fixing faults regardless of the type of test cases used (see the analysis of interactions between treatment and experience in Sections 4.1 and 4.2).
—The high-ability students knew the debugging process and the debugging tools (see our discussion of the post-questionnaire in Sections 4.4 and 6.4) and performed significantly better with autogen than with manual tests (see the analysis of interaction between treatment and experience in Sections 4.1 and 4.2).

—The experienced subjects spent more time on the bug than on the test code (see our analysis of the post-questionnaire in Section 5.4) and performed well with both autogen and manual test cases (see the analysis of efficiency and effectiveness in Sections 5.1 and 5.2).

Interpretation. We considered four levels of experience (BSc students, MSc students, PhD/Post-docs, and researchers/professors) and we further analyzed the actual ability of BSc/MSc students through questionnaire and debugging exercises. The performance of subjects is distributed along a spectrum that nicely follows their levels of ability and experience. Low-ability/BSc students have quite poor performance, regardless of the type of test cases used. They had a hard time fixing the faults and they encountered difficulties in the whole debugging process, including the use of tools and environments. High-ability/MSc students show good performance when using autogen test cases. For them, availability of focused, simple test cases empowers their fault-finding capabilities. These subjects have enough ability and skills to take advantage of the simplicity of the fault-revealing test cases generated automatically by tools, while they face more difficulties when provided with complex, manually written test cases. They know the debugging process and associated tools relatively well, but they still work much better if simple test cases are provided. At the end of the spectrum are experienced subjects (PhD/Post-docs and researchers/professors), who have good debugging performance with any kind of test case (manual or autogen). As long as a test case reveals the fault, these subjects can perform debugging accurately and efficiently. They do not spend much time on the test case itself, but rather focus on understanding the bug and application code.

Practical Implications

Imp3.1. Low-ability developers should not be allocated debugging at all.
Imp3.2. Debugging of complex manual tests should be allocated to senior developers.

Find4. The debugging performance depends on the complexity of the system.

Pieces of Evidence

—The subject system has a significant influence on the efficiency for experienced subjects (see our analysis in Sections 5.1 and 5.2).
—The control flow of failing tests is more complicated in XML-Security than in JTopas (see the analysis of dynamic complexity in Section A.2).
—Understanding the application code has been reported as a significant factor in all the experiments (see our analyses of the post-questionnaire in all experiments, reported in Sections 4.4, 5.4, and 6.4).

Interpretation. Experienced subjects had similar performance with both manual and autogen tests, but demonstrated a different effectiveness when working with JTopas rather than XML-Security, although the faults themselves consist in either case of similar unit-level defects. Actually, in the failing executions, XML-Security follows a control flow that is more complicated to understand and analyze than JTopas. This is expected to be associated with a higher application code understanding effort. Since such an effort has been reported by all subjects as a major factor affecting their debugging performance, we conclude that the complexity of the system when exercised under the failing scenario is a key factor affecting the capability of accurately and efficiently fixing the bug.

Practical Implications

Imp4.1. It is important to take into consideration the complexity and nature of the system, and not only the nature of the tests, when allocating debugging tasks to developers.

Find5. Usage of advanced debugging environments is fundamental with complex test scenarios.

Pieces of Evidence

—Only the subjects who took advantage of the Eclipse debugger mastered the most complex manual test cases (see the analysis of our feedback questionnaire for MvR reported in Section 4.4, and of the post-questionnaire for MvE reported in Section 5.4).

Interpretation. Only subjects able to effectively use the Eclipse debugger could take advantage of the more complex manual test cases to fix faults. When the complexity of a test case becomes high, automation of the debugging activities is required in order for the tester to be able to effectively and efficiently investigate the execution and locate and fix the fault. Experienced subjects used the debugging functionalities offered by Eclipse extensively and were not impacted by the complexity of the test scenarios.

Practical Implications

Imp5.1. The use of proper automation tools is fundamental for success when the system and the faults are not trivial.

Imp5.2. Training junior developers on the use of debugging tools is an investment with a very high potential return, to be seen during the execution of actual debugging tasks.

To summarize, we can highlight two key findings. First, the efficiency and the effectiveness of debugging are not affected by autogen test cases; rather, autogen test cases, compared to manually written tests, are easier to debug for the least experienced developers. Second, the understandability of the test cases (e.g., the presence of meaningful identifiers and the existence of a meaningful test scenario) does not affect debugging, while the static and, more importantly, the dynamic complexity of the tests strongly impacts the effectiveness and the efficiency of debugging.

We have listed a number of implications derived from these findings. Concerning the use of tools for automated test case generation, we observe that the debugging capabilities of testers, especially the less experienced ones, can be amplified by providing them with focused and simple autogen test cases that reveal the faults that need to be fixed. Such a benefit is not compromised by the use of meaningless identifiers in the test cases. Hence, whenever the same fault can be revealed by complex manual test cases, but also by simple automated tests, the latter are preferable because they can usually be generated faster than manually written tests and, for the less experienced people, they even maximize the debugging performance. The faults that can be revealed by both autogen and manual test cases are typically the ones that cause failures that can be detected without exploiting any specific knowledge of the application, such as crashes, hangs, exceptions, and assertion violations. Based on the results obtained in our experiments, we reconsider the whole testing process and the potential room for automated test case generation. We think that our results suggest the following strategy: (1) first, generate automated test cases and fix any bugs revealed by them; (2) then write/consider manual test cases. Compared to autogen tests, manual tests are more expensive to implement and use by less experienced subjects. As a result, less experienced developers should first consider autogen tests in debugging. Autogen and manual test cases are equally effective for experienced developers, so the choice between them is not critical for these subjects.

When using autogen test cases, developers might occasionally experience false positives. For instance, an autogen test case might fail because it violates an implicit method precondition. However, this is a general problem of automated test case generation and is out of the scope of the present investigation. Under the assumption of a reasonably low false-positive rate, according to the results of our empirical study, increasing the number of faults that are debugged using failing autogen test cases and decreasing the ones that are debugged using manual test cases can significantly improve the debugging effectiveness and efficiency.

8. THREATS TO VALIDITY

The main threats to the validity of this experiment belong to the conclusion, internal, construct, and external validity threat categories.

Conclusion validity threats concern the relationship between treatment and outcome. We used statistical tests (general linear models and Wilcoxon) to draw our conclusions. Inability to reject the null hypothesis exposes us to Type-II errors (incorrectly accepting a false null hypothesis) when we claim that no statistically significant difference was observed in the experiments. However, this probability was computed and reported, and was always fairly low. We have further mitigated this threat by replicating the experiment MvE, in which the null hypothesis could not be rejected, two times so as to increase the number of participants. In fact, the probability of a Type-II error can be reduced by increasing the sample size. In MvO the probability of a Type-II error is relatively high (23%) on effectiveness, but acceptable (13%) on efficiency.

Since we used GLM to determine the statistical significance of our results, we have applied the Shapiro-Wilk test to check the normality of the residuals. The only case of deviation from normality is on the first experiment (i.e., MvR), but inspection of the corresponding Q-Q plot did not reveal major problems. The survey questionnaire was designed using standard scales and improved after experiment MvR to better detect issues and opinions.

Internal validity threats concern external factors that may affect the independent variable. Subjects were not aware of the experimental hypotheses. Subjects were not rewarded for participation in the experiment and were not evaluated on their performance in doing the experiment.

Construct validity threats concern the relationship between theory and observation. They are mainly due to how we measure the effectiveness of debugging. We relied on previously defined test cases to objectively evaluate whether the fixes were correct. The order in which subjects face tasks might affect the results. To control this factor, we pre-ordered tasks (by difficulty). The ability of students was estimated according to their development background and using their exam scores. For the BSc students involved in MvR, we also used the result of the training lab.

External validity concerns generalization of the findings. In our experiments we considered test cases generated by Randoop and EvoSuite. Although other generators could be used, the results obtained with two generators, working according to different principles, already well support our interpretations.

Our studies exploited two different real-world systems from different domains and with different complexity. In principle, different results could be obtained for different systems. To mitigate this issue we identified the domain of validity of the reported results by characterizing the two applications, their tests, and their faults, within the domain of open-source software. The characterization reported in Appendix A could be used by other researchers to compare their results with ours.

The study was performed in an academic environment which may differ substantially from an industrial setup. However, we mitigate this threat by using subjects with different levels of experience and ability, including highly experienced PhD/Postdocs and researchers/professors, some with experience in professional software development. Moreover, we considered ability and experience as factors to detect any influence on the results.

9. CONCLUSION

We conducted a family of three experiments having a common goal: understanding the impact of automatically generated test cases on the effectiveness and efficiency of debugging. The first two experiments are based on test cases produced by two different test case generators, Randoop and EvoSuite. The third experiment used manually written test cases with obfuscated identifiers. It investigated the impact of identifier obfuscation alone, since all test case generators produce "obfuscated" (meaningless) identifiers. Experiments were conducted on two applications, JTopas and XML-Security, which have been found empirically to include test cases that are representative of medium-complexity test suites available with open-source projects and faults that are representative of real faults. In total, we involved 55 human subjects in the experiments, with a wide range of experience and ability, from MSc and BSc students to PhD/Post-docs and researchers/professors.

The key findings obtained from our experiments are that, while the meaningfulness (or lack thereof) of the identifiers that appear in test cases is not significantly detrimental to debugging, the complexity of the test cases is a major factor affecting both effectiveness and efficiency of debugging. Although autogen test cases contain meaningless identifiers, they usually consist of very simple, linear statement sequences. They have been found equally effective as manual test cases for debugging, in general. Indeed, they are even more effective than manual test cases when they are used by subjects with intermediate testing experience and ability (such as MSc students with high ability), thanks to their low dynamic complexity. Experienced subjects (PhD/Post-docs and researchers/professors) are affected more by the complexity of the system under test than by the test cases themselves. According to the answers they gave to the post-questionnaire, their understanding effort was focused mostly on application code, not on test case code, during debugging, while for less experienced subjects understanding the test code was also quite important.

The overall result of our family of experiments indicates that automatically generated test cases are not a major factor that affects the performance of testers while debugging. Other factors—test case dynamic complexity, system complexity, developers' experience, use of tools—have been found to play a much more prominent role. Hence, in the testing process, automated test case generation has the potential to give a key contribution by inexpensively providing evidence of faults and by supporting debugging of such faults as effectively as do manually defined test cases. We recommend to run automatically generated test cases first and to use them in debugging the detected faults. Developers can take advantage of the fault-revealing capability of automated tools without any major negative impact on the debugging effectiveness and efficiency.

As with any empirical research, our study is open to further validation and refinement. By replicating our study in different configurations (with alternative systems, faults, test case generation tools, and subjects) we will be able to accumulate a body of knowledge on the impact of automated test case generation on debugging, which is a key issue when tools are to be adopted in production software development environments. We provide all the material and data of our study to facilitate and support future replications[7].

ELECTRONIC APPENDIX

The electronic appendix to this article can be accessed in the ACM Digital Library.

REFERENCES

J. H. Andrews, L. C. Briand, and Y. Labiche. 2005. Is mutation an appropriate tool for testing experiments? In *Proceedings of the 27th International Conference on Software Engineering (ICSE'05)*. 402–411.

J. H. Andrews, A. Groce, M. Weston, and R.-G. Xu. 2008. Random test run length and effectiveness. In *Proceedings of the 23rd IEEE/ACM International Conference on Automated Software Engineering (ASE'08)*. IEEE Computer Society, 19–28.

S. Artzi, J. Dolby, F. Tip, and M. Pistoia. 2010. Directed test generation for effective fault localization. In *Proceedings of the 19th International Symposium on Software Testing and Analysis (ISSTA'10)*. ACM Press, New York, 49–60.

N. E. Beckman, A. V. Nori, S. K. Rajamani, R. J. Simmons, S. D. Tetali, and A. V. Thakur. 2010. Proofs from tests. *IEEE Trans. Softw. Engin.* 36, 495–508.

J. Burnim and K. Sen. 2008. Heuristics for scalable dynamic test generation. In *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering (ASE'08)*. 443–446.

C. Cadar, V. Ganesh, P. M. Pawlowski, D. L. Dill, and D. R. Engler. 2008. EXE: Automatically generating inputs of death. *ACM Trans. Inf. Syst. Secur.* 12, 2.

M. Ceccato, A. Marchetto, L. Mariani, C. D. Nguyen, and P. Tonella. 2012. An empirical study about the effectiveness of debugging when random test cases are used. In *Proceedings of the 34th International Conference on Software Engineering (ICSE'12)*. 452–462.

M. Ceccato, C. D. Nguyen, A. Marchetto, L. Mariani, and P. Tonella. 2013. A family of experiments to assess the impact of automated test case generation on the accuracy and efficiency of debugging, data analysis of five replications. Tech. rep. FBK, TR-FBK-SE-2013-2. `https://se.fbk.eu/technical-reports`.

I. Ciupa, A. Leitner, M. Oriol, and B. Meyer. 2007. Experimental assessment of random testing for object-oriented software. In *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA'07)*. ACM Press, New York, 84–94.

T. M. Cover and P. E. Hart. 1967. Nearest neighbor pattern classification. *IEEE Trans. Inf. Theory* 13, 1, 21–27.

H. Do, S. G. Elbaum, and G. Rothermel, G. 2005. Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact. *Empir. Softw. Engin.* 10, 4, 405–435.

J. W. Duran. 1984. An evaluation of random testing. *IEEE Trans. Softw. Engin.* 4, 438–444.

P. G. Frankl and S. N. Weiss. 1991. An experimental comparison of the effectiveness of the all-uses and all-edges adequacy criteria. In *Proceedings of the Symposium on Testing, Analysis, and Verification (TAV4'91)*. ACM Press, New York, 154–164.

G. Fraser and A. Arcuri. 2011. Evolutionary generation of whole test suites. In *Proceedings of the 11th International Conference on Quality Software (QSIC'11)*. 31–40.

G. Fraser and A. Zeller. 2010. Mutation-driven generation of unit tests and oracles. In *Proceedings of the 19th International Symposium on Software Testing and Analysis (ISSTA'10)*. ACM Press, New York, 147–158.

Z. P. Fry and W. Weimer. 2010. A human study of fault localization accuracy. In *Proceedings of the IEEE International Conference on Software Maintenance (ICSM'10)*. IEEE Computer Society, 1–10.

P. Godefroid, N. Klarlund, and K. Sen. 2005. DART: Directed automated random testing. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'05)*. ACM Press, New York, 213–223.

P. Godefroid, M. Y. Levin, and D. A. Molnar. 2008. Automated whitebox fuzz testing. In *Proceedings of the Network and Distributed System Security Symposium (NDSS'08)*.

S. Holm. 1979. A simple sequentially rejective multiple test procedure. *Scandinav. J. Statist.* 6, 2, 65–70.

L. Huang and M. Holcombe. 2009. Empirical investigation towards the effectiveness of test first programming. *Inf. Softw. Technol.* 51, 1, 182–194.

[7]Replication package available at `http://selab.fbk.eu/ceccato/replication_packages/debugging_replication_package.tgz`.

J. Itkonen, M. V. Mantyla, and C. Lassenius. 2009. How do testers do it? An exploratory study on manual testing practices. In *Proceedings of the 3rd International Symposium on Empirical Software Engineering and Measurement (ESEM'09).* IEEE Computer Society, 494–497.

J. A. Jones, M. J. Harrold, and J. Stasko. 2002. Visualization of test information to assist fault localization. In *Proceedings of the International Conference on Software Engineering (ICSE'02).* ACM Press, New York, 467–477.

A. N. Oppenheim. 1992. *Questionnaire Design, Interviewing and Attitude Measurement*. Pinter, London.

C. Pacheco and M. D. Ernst. 2007. Randoop: Feedback-directed random testing for Java. In *Companion to the 22nd ACM SIGPLAN Conference on Object-Oriented Programming Systems and Applications Companion (OOPSLA'07).* ACM Press, New York, 815–816.

C. Parnin and A. Orso. 2011. Are automated debugging techniques actually helping programmers? In *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA'11).* ACM Press, New York, 199–209.

F. Ricca, M. Torchiano, M. Di Penta, M. Ceccato, and P. Tonella. 2009. Using acceptance tests as a support for clarifying requirements: A series of experiments. *Inf. Softw. Technol.* 51, 2, 270–283.

J. R. Ruthruff, M. Burnett, and G. Rothermel. 2005. An empirical study of fault localization for end-user programmers. In *Proceedings of the 27th International Conference on Software Engineering (ICSE'05).* ACM Press, New York, 352–361.

K. Sen, D. Marinov, and G. Agha. 2005. CUTE: A concolic unit testing engine for C. *SIGSOFT Softw. Engin. Notes* 30, 5, 263–272.

N. Tillmann and J. De Halleux. 2008. Pex: White box test generation for .NET. In *Proceedings of the 2nd International Conference on Tests and Proofs (TAP'08).* Springer, 134–153.

P. Tonella. 2004. Evolutionary testing of classes. In *Proceedings of the ACM/SIGSOFT International Symposium on Software Testing and Analysis (ISSTA'04).* 119–128.

C. J. Van Rijsbergen. 1979. *Information Retrieval,* 2nd ed. Butterworths, London.

M. Weiser and J. Lyle. 1986. Experiments on slicing-based debugging aids. In *Proceedings of the 1st Workshop on Empirical Studies of Programmers.* Ablex Publishing, Norwood, NJ, 187–197.

C. Wohlin, P. Runeson, M. Host, M. C. Ohlsson, B. Regnell, and A. Wesslen. 2012. *Experimentation in Software Engineering*. Springer.

Y. Yu, J. A. Jones, and M. J. Harrold. 2008. An empirical study of the effects of test-suite reduction on fault localization. In *Proceedings of the 30th International Conference on Software Engineering (ICSE'08).* ACM Press, New York, 201–210.

A. Zeller and R. Hildebrandt. 2002. Simplifying and isolating failure-inducing input. *IEEE Trans. Softw. Engin.* 28, 2, 183–200.

Received May 2014; revised November 2014, March 2015; accepted April 2015