An Empirical Investigation on the Readability of Manual and Generated Test Cases

Anonymous Author(s)

ABSTRACT

Software testing is one of the most crucial tasks in the typical development process. Developers are usually required to write unit test cases for the code they implement. Since this is a time-consuming task, in last years many approaches and tools for automatic test case generation — such as EvoSuite— have been introduced. Nevertheless, developers have to maintain and evolve tests to sustain the changes in the source code; therefore, having readable test cases is important to ease such a process. However, it is still not clear whether developers make an effort in writing readable unit tests. Therefore, in this paper, we conduct an explorative study comparing the readability of manually written test cases with the classes they test. Moreover, we deepen such analysis looking at the readability of automatically generated test cases. Our results suggest that developers tend to neglect the readability of test cases and that automatically generated test cases are generally even less readable than manually written ones.

CCS CONCEPTS

• **Software and its engineering → Maintaining software**; **Search-based software engineering**;

KEYWORDS

Readability, Automated Testing, Empirical Software Engineering

ACM Reference Format:

Anonymous Author(s). 2018. An Empirical Investigation on the Readability of Manual and Generated Test Cases. In *Proceedings of IEEE/ACM International Conference of Program Comprehension (ICPC 2018).* ACM, New York, NY, USA, 4 pages. https://doi.org/10.475/123_4

1 INTRODUCTION

Software testing is among the most expensive processes in software development [6]. In many contexts, developers are required to thoroughly test the software they write to guarantee a certain level of external quality. Agile development methodologies and, most notably, test-driven development (TDD), put a strong accent on testing: test cases are written when the code itself does not exist yet [5]. As well as source code, test cases need maintenance: when a part of the code changes, all the test cases that depend on it may need to change accordingly, to avoid erroneous failure reports [22]. If the test cases have poor readability, it may be harder for developers to evolve and maintain them.

ICPC 2018, May 27-28, 2018, Gothenburg, Sweden

ACM ISBN 123-4567-24-567/08/06…$15.00
https://doi.org/10.475/123_4

Being the test cases writing a hard and time-consuming task, in last years different approaches for automatic case generation have been proposed. They generate test cases for a given unit to test, aiming at maximizing one or more coverage criteria; such an unit is typically a *class under test* (CUT). However, those tests still need human intervention: human effort in reading, understanding and modifying test cases is necessary, especially since the oracles have to be manually specified [2].

Even if the quality of test cases is important to ease their maintenance [1, 23], it was shown that test smells, *i.e.*, symptoms of possible design issues of the test code [22], are very spread both in open source and industrial code; moreover, they have a strong impact on comprehension and maintenance [4]. This situation is even worse in automatically generated test cases, since they are more affected by test smells than their manually written counterparts [17]. In this context, it is clear that having simple and readable test cases helps developers to keep the pace with fast development cycles. Therefore, developers should spend the same effort in writing good —and readable— unit tests, as they do for the source code. However, to the best of our knowledge, no previous study analyzed the relationship between the quality —readability, in particular — of the test cases and source code. Thus, in this paper we aim to answer the following question: *are manually written test cases more readable than the source code they exercise?*

To do that, we conduct an empirical investigation in which we compare the readability of test cases to the one of the corresponding CUTs. Moreover, we deepen our analysis studying the difference between the readability of automatically generated test cases and manually written ones, to understand if what was found for test smells [17] is true also for readability.

Our results show that source code is significantly more readable than test cases. This might suggests that developers do not focus much on writing good quality test cases. Moreover, we observe that manually written test cases are significantly more readable than automatically generated ones, despite the fact the tool we used applied specific techniques to improve their readability [13]. These results open new lines of research aimed at automatically improving the quality of existing test cases, both manually written and automatically generated, with the effect of reducing maintenance and testing costs.

2 BACKGROUND & RELATED WORK

Code readability is an aspect of source code that allows developers to quickly read the code and get information from it. Recent studies defined approaches that automatically measure code readability [7, 18, 21]. Such approaches are based on structural [7, 18], visual [10] and textual [21] features that can be measured on source code. Given a dataset of snippets evaluated by developers, all such approaches consist in training a classifier able to distinguish *readable* snippets from *unreadable* ones.

When developers perform maintenance tasks on the source code, they are required to update the related test cases. Therefore, having readable test cases constitutes a benefit for developers. Van Deursen *et al.* [22] defined a set of test smells, *i.e.*, symptoms of possible design problems in the test cases. Bavota *et al.* [4] studied the prevalence and the impact of such smells, showing that they are spread and have a strong impact on comprehension and maintenance. Our study differs from the one by Bavota *et al.* [4] because we do not only look at the quality of test code but, instead, we compare it to the source code, to check if there is any difference.

Writing test cases can be a hard task. For this reason, tools that automatically generate test cases were introduced. Examples of automatic test case generation tools are Randoop [15] and EvoSuite [11]. Such tools use search-based approaches [12, 16]: the main goal is to find a sub-optimal set of test cases that achieve the maximum coverage of the CUT.

Recent studies have focused on the quality of automatically generated test cases. Palomba *et al.* [17] showed that such test cases are significantly more prone to contain test smells compared to manually written ones. Daka *et al.* [9] introduced test-specific features to measure and improve readability of automatically generated test cases. Our study is different from the one by Palomba *et al.* [17], because (i) we make a comparison between source code and test code and (ii) we use readability as a proxy of code quality instead of the number of test smells, trying to evaluate test quality from a different perspective.

3 EMPIRICAL STUDY

The *goal* of our empirical study is to initially investigate the readability of test cases in relation to the CUTs. Moreover, we aim to detect possible differences in readability metrics between manual and automatically generate test cases. Our study is steered by the following research questions:

RQ$_1$. *Are test cases equally readable as the corresponding CUT?*

With this first research question, we aim at understanding how test cases are readable compared to the classes they test. We want to understand whether developers put an unbalanced effort in writing readable code, *i.e.*, they care less about the quality of tests.

RQ$_2$. *Are automatically generated test cases equally readable as the manually written ones?*

With our second research question, we focus on detecting possible differences in readability between manually written and automatically generated test cases. We argue that a lower readability might be a limiting factor in the adoption of such tools.

Project Subjects. The context of this study is composed by 3 popular Apache projects: Apache Commons BCEL[1], Apache Commons Math[2] and Apache Commons Lang[3]. The first one provides APIs for manipulating binary Java classes; the second one is a library of mathematics and statistic operators; the third one provides helper utilities for Java core classes. Such a selection is driven by the popularity of Apache projects in software evolution and maintenance literature [3]. Table 1 summarizes the characteristics of the test cases and the relative CUTs we consider for every project.

[1]http://commons.apache.org/proper/commons-bcel/
[2]http://commons.apache.org/proper/commons-math/
[3]http://commons.apache.org/proper/commons-lang/

Table 1: Projects used for the empirical study

Project	CUTs	CUTs LOC	Tests LOC
Commons-BCEL	12	5,563	984
Commons-Math	356	122,545	89,766
Commons-Lang	111	74,639	52,317
Total	479	202,747	143,067

3.1 Experiment Methodology

In our study, we use a state-of-the-art readability model [21] to compute the readability of both tests and CUTs. We do not use a specialized readability model [9], since we would not be able to use it for both test cases and source code. On the other hand, the model we use is generic, since it is trained on both source code and test cases. Such a model is based on logistic regression [21] and it classifies a given snippet as *readable* or *unreadable*. In this study, we compute the continuous readability level $r \in [0, 1]$ as the probability associated to the class *readable*. Therefore, $r = 0$ means that the the classifier is confident in classifying it as *unreadable*, while $r = 1$ means that the classifier is sure that it is *readable*. The training set of the model is constituted by snippets (*e.g.*, methods) manually annotated by human developers as for their readability; we used as trainig set the union of the datasets in the state of the art, *i.e.*, the ones by Buse and Weimer [7], Dorn [10] and Scalabrino *et al.* [21]. Since we train our model on small snippets, directly computing the readability of whole classes may mislead the classifier. Therefore, we compute the readability of a class C as the mean of the readability computed on all the methods belonging to C.

3.1.1 Tests vs Source Code Readability. To answer **RQ$_1$**, we compute the readability of both (i) the test cases and (ii) the CUTs. To select the set of test cases and the corresponding CUTs, we rely on the Maven `pom` file of each project. Such a file contains the rule to identify test classes to run when the project has to be built. Then, using both the name of the test classes and the patterns in the `pom`, we detect the CUTs. For example, given a test case `SimpleCurveFitterTest.java` and pattern `**/*Test.java`, we remove the word *Test* at the end of the name of the test case to determine the name of the CUT, *i.e.*, `SimpleCurveFitter.java`, in this case.

We use a two tailed Wilcoxon test to check whether there is any difference between the readability of test cases and the readability of the CUTs. Our null hypothesis is that there is no difference between test cases and CUTs as for code readability. We reject the null hypothesis if the p-value is lower than 0.05. Finally, if the difference is significant, we compute the Cliff's delta δ [8] to measure the magnitude of such difference. We consider the difference *negligible* for $|\delta| < 0.148$, small for $0.148 \leq |\delta| < 0.33$, medium for $0.33 \leq |\delta| < 0.474$ and large for $|\delta| \geq 0.474$ [14].

Table 2: Average readability scores for the CUTs and the corresponding tests.

Project	CUTs	Tests	p-value	Cliff δ
commons-bcel	0.74	0.50	0.009	0.79 (large)
commons-lang	0.86	0.64	< 0.001	0.83 (large)
commons-math	0.82	0.61	< 0.001	0.68 (large)
Overall	**0.83**	**0.61**	**< 0.001**	**0.71 (large)**

3.1.2 Manual vs Automated Test Cases. To answer **RQ**$_2$, we compute the readability of the manually written test cases and the automatically generated ones. We rely on EvoSuite [11] to automatically generate tests, because it is one of the most popular tools in automatic test case generation literature. We use its default algorithm, *i.e.*, the *whole test suite* approach proposed by Fraser and Arcuri [12]. We set to 180 seconds the search budget for each CUT. We run EvoSuite on a Linux machine running Ubuntu 16.04, having 16 cores and 64GB of RAM. To avoid that the randomness of automatic test case generation influences our results, we repeat the process 5 times for each CUT and we compute the average readability of the all the different versions of each test. It is worth noting that EvoSuite is not able to generate test cases for some type of classes (*e.g.*, abstract classes). We ignore such classes in **RQ**$_2$. In total, it was not possible to generate test cases for 8 classes over the 479 we took into account (< 2%).

As we do for **RQ**$_1$, we use a two tailed Wilcoxon test to check if the readability of manually written and automatically generated test cases differ. Our null hypothesis is that there is no difference between such test cases as for readability. We reject the null hypothesis if the p-value is lower than 0.05. Also in this case, we report the Cliff's delta δ of the difference as previously described for **RQ**$_1$.

3.2 Results & Discussion

3.2.1 Tests vs Source Code Readability. Table 2 reports, for every considered project, the average readability for both the CUTs and the tests. We show in the "*Overall*" row the values on the whole dataset (*i.e.*, not aggregated by project). It is clear that the p-values of the Wilcoxon tests are lower than 0.05 for all the projects and the same is also true when looking at the dataset in its entirety. Thus, we can reject our null hypothesis, *i.e.*, we can report a difference in term of code readability between CUTs and tests. To understand the magnitude of such difference, we compute the Cliff's delta (δ). Looking at both the score achieved in the single projects and the overall score, it can be observed a *large* magnitude of difference, (0.71, overall). The CUTs classified as *readable* by the model are 459, while the tests classified as readable are 370 (~ 19% percent less).

Result 1. Our analysis shows a significantly lower code readability for the tests cases compared to the corresponding CUTs.

3.2.2 Manual vs Automated Test Cases. In Table 3 we report the results of our analysis, similarly as we did for RQ$_1$. Analyzing the results at project-level, we notice that for Apache Commons BCEL

Table 3: Average readability scores for the manually and the automatically generated tests.

Project	Man.	Aut.	p-value	Cliff δ
commons-bcel	0.50	0.48	0.791	-
commons-lang	0.64	0.53	< 0.001	0.41 (medium)
commons-math	0.61	0.52	< 0.001	0.29 (small)
Overall	**0.61**	**0.52**	**< 0.001**	**0.31 (small)**

the difference is not significant ($p > 0.05$). However, this might be caused by the few tests available for such project (only 12). We observe that the readability of manually written test cases is slightly higher, on average, in any case. On the other hand, for the other two projects and overall, the difference in readability between the manually written and the automatically generated tests is significant. Thus, we reject again the null hypothesis and we can say that the test cases written by developers are more readable than the automatically generated ones (by EvoSuite, in this case). However, differently from what happened in **RQ**$_1$, the overall magnitude of the difference, is *small*. In this case, the manually written tests classified as *readable* are 365, while the automatically generated tests classified as *readable* are 270 (~ 26% less). It is worth noting that the absolute number of manually written tests classified as *readable* is lower than in **RQ**$_1$: here, indeed, as previously explained, we had to ignore some manually written tests, because EvoSuite could not generate automatic tests for the related CUTs.

Result 2. Tests cases manually written by developers tend to be more readable than the automatically generated ones.

3.2.3 Discussion. The results of **RQ**$_1$ can be interpreted in two different ways. It is possible that developers care less about the quality of test cases from the beginning, *i.e.*, their effort in making them readable is relatively small. On the other hand, it is possible that the lower readability is due to a slow decay in the quality of test cases during the evolution of the project [22]. In both the cases, developers do not seem to perceive the quality of test cases as important. It is worth noting that test cases have some characteristics that should make them *more readable* than CUTs. Indeed, they tend to be shorter and to have a lower number of control structures. However, despite this theoretical advantage, in reality the opposite is true. Consider this line from a test case in Commons-Math:

```
Assert.assertEquals(l.getLocation().getAlpha(), l.getReverse().
      getLocation().getAlpha(), 1.0e-10);
```

the main issue is the fact that it contains longer chains of method calls, *i.e.*, developers did not use intermediate variables to store partial results. This results in a longer line, negatively correlated with readability [7]. Moreover, the only identifier used for a local variable, l, is not meaningful. Our results pave the way for a broader investigation aiming at understanding the differences in quality between tests and CUTs, why developers underestimate the importance of quality in test cases and how to improve their awareness of such a problem. The results of our second research

question, instead, confirm what Palomba *et al.* [17] observed, *i.e.*, that automatically generated test cases, compared to manually written ones, tend to have a lower quality, *i.e.*, more test smells and, as we found, lower readability. Overall, we can conclude that there is still a long way to go to have an adequate level of quality in both manually written and, mostly, automatically generated tests.

4 THREATS TO VALIDITY

Internal Validity. Our study is mainly focused on code readability as a proxy for code quality. We used an automatic approach for code readability measurement. It may not perfectly match the opinion of actual human developers; however, the approach we used achieves a reported accuracy in snippet classification of about 80% [21]. Automatic test case generation is random at its base. To reduce the influence of randomness in our results, we run EvoSuite 5 times for each CUT. Previous research [9] focused on the improvement of the readability of automatically generated test cases. Unfortunately, we could not replicate the results on our dataset, since the approach is not available in EvoSuite. However, the authors report a small average improvement in readability (1.9%), while our results show an average difference between manual and generated tests of 9%, overall. To reduce this threat, we also simulated the best-case scenario for automatic test case generation as for readability: instead of considering the average readability among the five runs of EvoSuite, we tried to consider, for each class, the most readable automatically generated test case. In other words, given a class C and the five tests $T_C^1, ..., T_C^5$ generated in the five runs of EvoSuite, we considered $\max_i readability(T_C^i)$. Even in this extreme scenario, manually written test cases are still significantly more readable than automatically generated ones. Finally, while readability is an important aspect of source code, it was showed that it is not related to understandability [20]. Therefore, it cannot be stated that test cases are less *understandable* than source code. The assessment of this different aspect needs further investigation.

External Validity. We run our experiment on 3 open source projects, for a total of 479 test cases and CUTs. A larger dataset might obviously improve the generalizability of our findings. This is part of our future agenda. In order to automatically generate test cases we rely on EvoSuite, because it is one of the most mature tools available. However, we cannot guarantee similar results using different tools, *e.g.*, Randoop [15] or JTExpert [19]. Further investigation for such tools is also part of our future work.

5 CONCLUSION AND FUTURE WORK

In this work we conducted an empirical study aiming at investigating the readability of test cases. In particular, we focused on the difference in code readability between (i) test cases and correspondent classes under test and (ii) manually written and automatically generated test cases. Our preliminary results open new interesting research directions. First of all, we showed that test cases are significantly less readable than source code. We argue that this fact might suggest that developers tend to neglect the quality of such artifacts in favor to the one of CUTs.

We plan to investigate more in depth this phenomenon. If confirmed, this early result would justify the design and implementation of specific tools to support developers in improving the quality of test cases. The second finding of our study is that automatically generated test cases are significantly less readable than the manually written ones. Despite the magnitude of such a difference is small, we argue that this phenomenon might be one of the causes of the low adoption of test case generation tools. To tackle such a problem, we plan to experiment new techniques to improve the quality of automatically generated test cases.

REFERENCES

[1] D. Athanasiou, A. Nugroho, J. Visser, and A. Zaidman. Test code quality and its relation to issue handling performance. *IEEE Transactions on Software Engineering*, 40(11):1100–1125, 2014.

[2] E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, and S. Yoo. The oracle problem in software testing: A survey. *IEEE transactions on software engineering*, 41(5):507–525, 2015.

[3] G. Bavota, G. Canfora, M. D. Penta, R. Oliveto, and S. Panichella. The evolution of project inter-dependencies in a software ecosystem: The case of apache. In *Proceedings of the 2013 IEEE International Conference on Software Maintenance*, ICSM '13, pages 280–289, Washington, DC, USA, 2013. IEEE Computer Society.

[4] G. Bavota, A. Qusef, R. Oliveto, A. De Lucia, and D. Binkley. Are test smells really harmful? an empirical study. *Empirical Software Engineering*, 20(4):1052–1094, 2015.

[5] Beck. *Test Driven Development: By Example*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[6] B. Beizer. *Software testing techniques*. Dreamtech Press, 2003.

[7] R. P. Buse and W. R. Weimer. Learning a metric for code readability. *IEEE Transactions on Software Engineering*, 36(4):546–558, 2010.

[8] N. Cliff. Dominance statistics: Ordinal analyses to answer ordinal questions. *Psychological Bulletin*, 114(3):494, 1993.

[9] E. Daka, J. Campos, G. Fraser, J. Dorn, and W. Weimer. Modeling readability to improve unit tests. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, pages 107–118. ACM, 2015.

[10] J. Dorn. A general software readability model. *MCS Thesis available from (http://www. cs. virginia. edu/˜ weimer/students/dorn-mcs-paper. pdf)*, 2012.

[11] G. Fraser and A. Arcuri. Evosuite: Automatic test suite generation for object-oriented software. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, ESEC/FSE '11, pages 416–419, New York, NY, USA, 2011. ACM.

[12] G. Fraser and A. Arcuri. Whole Test Suite Generation. *IEEE Transactions on Software Engineering*, 39(2):276–291, 2013.

[13] G. Fraser and A. Zeller. Mutation-driven generation of unit tests and oracles. *IEEE Transactions on Software Engineering*, 38(2):278–292, 2012.

[14] R. Grissom and J. Kim. *Effect Sizes for Research: A Broad Practical Approach*. Lawrence Erlbaum Associates, 2005.

[15] C. Pacheco and M. D. Ernst. Randoop: feedback-directed random testing for java. In *Companion to the 22nd ACM SIGPLAN conference on Object-oriented programming systems and applications companion*, pages 815–816. ACM, 2007.

[16] C. Pacheco, S. K. Lahiri, M. D. Ernst, and T. Ball. Feedback-directed random test generation. In *Proceedings of the 29th international conference on Software Engineering*, pages 75–84. IEEE Computer Society, 2007.

[17] F. Palomba, D. Di Nucci, A. Panichella, R. Oliveto, and A. De Lucia. On the diffusion of test smells in automatically generated test code: An empirical study. In *Proceedings of the 9th International Workshop on Search-Based Software Testing*, pages 5–14. ACM, 2016.

[18] D. Posnett, A. Hindle, and P. Devanbu. A simpler model of software readability. In *Proceedings of the 8th working conference on mining software repositories*, pages 73–82. ACM, 2011.

[19] A. Sakti, G. Pesant, and Y. Gueheneuc. Instance generator and problem representation to improve object oriented code coverage. *IEEE Transactions on Software Engineering*, 41(3):294–313, March 2015.

[20] S. Scalabrino, G. Bavota, C. Vendome, M. Linares-Vásquez, D. Poshyvanyk, and R. Oliveto. Automatically assessing code understandability: how far are we? In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*, pages 417–427. IEEE Press, 2017.

[21] S. Scalabrino, M. Linares-Vásquez, D. Poshyvanyk, and R. Oliveto. Improving code readability models with textual features. In *Program Comprehension (ICPC), 2016 IEEE 24th International Conference on*, pages 1–10. IEEE, 2016.

[22] A. Van Deursen, L. Moonen, A. van den Bergh, and G. Kok. Refactoring test code. In *Proceedings of the 2nd international conference on extreme programming and flexible processes in software engineering (XP2001)*, pages 92–95, 2001.

[23] A. Zaidman, B. Van Rompaey, A. van Deursen, and S. Demeyer. Studying the co-evolution of production and test code in open source and industrial developer test processes through repository mining. *Empirical Software Engineering*, 16(3):325–364, 2011.