Poster — Unit Tests and Component Tests do Make a Difference on Fault Localisation Effectiveness

Gulsher Laghari
Universiteit Antwerpen, België
University of Sindh, Pakistan
gulsher.laghari@uantwerpen.be

Serge Demeyer
Universiteit Antwerpen, België
Flanders Make, België — INRIA Lille, France
serge.demeyer@uantwerpen.be

ABSTRACT

Agile testers distinguish between unit tests and component tests as a way to automate the bulk of the developer tests. Research on fault localisation largely ignores this distinction, evaluating the effectiveness of these techniques irrespective of whether the fault is exposed by unit tests—where the search space to locate the fault is constrained to the unit under test— or by component tests—where the search space expands to all objects involved in the test. Based on a comparison of sixteen spectrum based fault localisation techniques, we show that there is indeed a big difference in performance when facing unit tests and component tests. Consequently, researchers should distinguish between easy and difficult to locate faults when evaluating new fault localisation techniques.

KEYWORDS

Spectrum based fault localisation; Component tests; Unit tests.

ACM Reference Format:
Gulsher Laghari and Serge Demeyer. 2018. Poster — Unit Tests and Component Tests do Make a Difference on Fault Localisation Effectiveness. In *ICSE '18 Companion: 40th International Conference on Software Engineering Companion, May 27-June 3, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3183440.3194970

1 INTRODUCTION

Software testing is the activity of executing a program with the intent of finding a defect. A software test brings the implementation under test in a given state, then administers a sequence of stimuli and subsequently verifies whether the resulting state corresponds with the expectations. Once a software test exposes a defect, a software engineer still has to search for its root cause —the *fault*— and fix it accordingly. To minimise the search space, testing handbooks distinguish between *unit tests* and *component tests* [2]. A unit test isolates the implementation under test (typically a method or a class) from the rest of the system so that the tester can be confident that the defect is located within the unit. A component test, on the other hand, exercises the interactions between objects; when a component test exposes a defect, the defect should be in the code that manipulates the interface. Unfortunately, one cannot entirely rule out the code in the constituting objects (even with the presence of stubs and mock objects), thus the search space for locating the defect comprises all the objects involved in the component test.

To help software engineers locate the root cause of a defect, the research community has forwarded *fault localisation* heuristics, which produce a ranked list of program elements, indicating the likelihood of a program element being at fault [6]. A particular branch in this research is *spectrum based fault localisation* which analyses program traces generated by failing and passing tests to deduce the location of the fault [1, 4, 6].

Given that unit tests and component tests represent different strategies to pinpoint the fault, one would expect that the research on fault localisation also makes this distinction. However, none of the currently published evaluations do so: all of them rely on datasets such as the Siemens set, the Software-Artifact Infrastructure Repository (SIR), iBugs and the most recent Defects4J [3]. None of these datasets distinguish between unit tests and component tests, hence it is currently unknown how fault localisation heuristics deal with the more challenging faults involving a larger search space.

To illustrate the differences, we showcase two examples from Defects4J. An easy case for fault localisation is fault 3 in project Math. The unit test testLinearCombinationWithSingleElementArray in test class MathArraysTest calls only a single method (the one containing the fault) —linearCombination(double[], double[]) in class MathArrays. The test fails because of an ArrayIndexOutOfBoun- dsException, which is immediately visible in the stack trace and readily points to the location of the fault. In such cases, *any* fault localisation technique —even the most naive one— should have an accuracy of 100%. In contrast, one of the most difficult faults to locate is fault 74 in project Math. The test case polynomial in class AdamsMoultonIntegratorTest exposes a fault in method integrate(FirstOrderDifferentialEquations, double, double- [], double, double[]) within class EmbeddedRungeKuttaIntegra- tor. The test fails because one assertion fails: the returned value does not match the expected value because of some erroneous state manipulation earlier in the implementation under test. The faulty method does not appear in the stack trace, so one needs to search through all the 264 project methods indirectly called by the test case. Such needle-in-a-haystack cases are real challenges for fault localisation techniques because the search space of potential fault locations is so large.

In this paper, we demonstrate with sixteen established spectrum based fault localisation techniques that there is indeed a big difference in performance when facing faults revealed by unit tests or component tests.

ICSE '18 Companion, May 27-June 3, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5663-3/18/05.
https://doi.org/10.1145/3183440.3194970

2 CASE STUDY SETUP

Fault Localisation Techniques. We compare the performance of two families of spectrum based fault localisation on the defects exposed by unit tests and component tests. The first Basic family (B), is the standard implementation based on what is named the *Basic Hit-Spectrum* [1]. The second Extended family (E) employs an *Extended Hit-Spectrum*, which is a recent improvement using frequent itemset mining [4]. Each family is parameterised with the eight best performing fault locators as known today, thus resulting in sixteen different spectrum based fault localisation heuristics.

Dataset. We use the Defects4J dataset for our evaluation but extend the dataset to account for faults exposed by unit tests or component tests. To that extent, we adopt the definitions by Crispin and Gregory [2]: a unit test isolates the implementation under test (typically a method or a class), whereas a component test, exercises the interactions between objects (classes). Since we have the call traces anyway, we adopt a dynamic analysis heuristic, inspired by the one of Weijers [5].

Evaluation metrics. We use the most commonly adopted metrics *acc@n* (n ∈ {1, 3, 5}), *mean average precision* (MAP), and *mean wasted effort* (MWE) to evaluate the results.

3 RESULTS

Search space. Figure 1 illustrates that the search space differs quite a lot for unit tests versus component tests. The figure shows a violin plot, with the left side (red) the total number of project methods executed in failing tests categorised as component tests (red) and the right side (blue) in failing tests categorised as unit tests (blue). The graph for component tests is top heavy, indicating that these indeed call a lot more methods than their unit tests counterpart. A t-test with p-value less than 2.2e-16 and Cohen's d with 1.2, confirm that the search space is indeed significantly larger for component tests.

Unit tests. Given that the search space is smaller for defects exposed by unit tests, the fault localisation techniques are expected to perform better and this is confirmed by the results: for both of the families, there are very good scores for the metrics. Table 1 summarises the results. Overall, the top performing heuristics from Extended family successfully localise 45/73 (≈ 62%) defects, with mean average precision 0.70 and mean wasted effort ≈ 4. Similarly, those in the Basic family, successfully localise 35/73 (≈ 48%) defects, with mean average precision 0.64 and mean wasted effort ≈ 2 (2 methods to search in vain before reaching the faulty method).

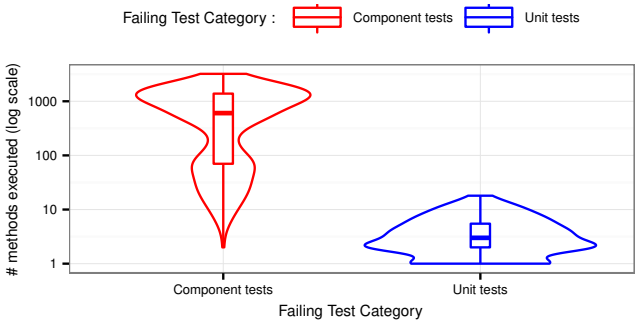

Figure 1: Assessment of the size of the search space (executed methods) for unit tests and component tests.

Table 1: Comparisons of the Two Families for Defects Revealed by Unit Tests (UT) and Component Tests (CT).

Family	T	acc@1	acc@3	acc@5	MAP	MWE
E	UT	45	58	64	0.7021061	3.85
	CT	53	95	123	0.2851449	39.44
B	UT	35	61	67	0.6440786	2.44
	CT	30	65	80	0.1913367	120.58

Component tests. For the defects exposed by component tests, the performance of the techniques from both families has decreased compared to the results for unit tests. The very same top performing techniques for the Extended family, now successfully localise 53/273 (≈ 19%) defects. Also the mean average precision 0.29 is lower and the mean wasted effort ≈ 39 is higher. Likewise, the Basic family only localises 30/273 (≈ 11%) defects successfully. Here as well, the mean average precision 0.19 is lower and the mean wasted effort ≈ 121 is higher. This observation indicates that when the techniques are evaluated on whole dataset without distinction of test types, the better performance of fault localisation techniques can be attributed to the easy-to-localise defects exposed by unit tests.

4 CONCLUSION

In this paper, we measured the size of the search space in the Defects4J dataset to be considered for fault localisation techniques. We demonstrated that it is rather small for defects exposed by unit tests (such tests call fewer methods), hence the spectrum based fault localisation heuristics perform rather well. However, the search space is considerably larger for defects exposed by component tests, which results in a decrease in performance.

This has an important consequence for future research in fault localisation: the best result depends a lot on the presence of unit tests or component tests. Consequently, researchers should distinguish between easy and difficult to locate faults when evaluating new spectrum based fault localisation techniques.

Acknowledgments. This work is sponsored by (a) the Higher Education Commission of Pakistan under a project titled "Strengthening of University of Sindh (Faculty Development Program)"; (b) Flanders Make vzw, the strategic research centre for the manufacturing industry; (c) the Conseil Régional Hauts-De-France; Nord Pas de Calais — Picardie.

REFERENCES

[1] José Campos, André Riboira, Alexandre Perez, and Rui Abreu. 2012. GZoltar: An Eclipse Plug-in for Testing and Debugging. In *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering (ASE 2012)*. ACM, New York, NY, USA, 378–381. https://doi.org/10.1145/2351676.2351752

[2] Lisa Crispin and Janet Gregory. 2009. *Agile Testing: A Practical Guide for Testers and Agile Teams.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[3] René Just, Darioush Jalali, and Michael D. Ernst. 2014. Defects4J: A Database of Existing Faults to Enable Controlled Testing Studies for Java Programs. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis (ISSTA 2014)*. ACM, New York, NY, USA, 437–440. https://doi.org/10.1145/2610384.2628055

[4] Gulsher Laghari, Alessandro Murgia, and Serge Demeyer. 2016. Fine-tuning Spectrum Based Fault Localisation with Frequent Method Item Sets. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering (ASE 2016)*. ACM, New York, NY, USA, 274–285. https://doi.org/10.1145/2970276.2970308

[5] Joep Weijers. 2012. *Extending Project Lombok to improve JUnit tests.* Master's thesis. Delft University of Technology, the Netherlands. http://repository.tudelft.nl/islandora/object/uuid:1736d513-e69f-4101-8995-4597c2a4df50/datastream/OBJ/download

[6] W. E. Wong, R. Gao, Y. Li, R. Abreu, and F. Wotawa. 2016. A Survey on Software Fault Localization. *IEEE Transactions on Software Engineering* 42, 8 (Aug 2016), 707–740. https://doi.org/10.1109/TSE.2016.2521368