Evaluation of Fault Localization Techniques

Spencer Pearson
University of Washington
Seattle, WA, USA
suspense@cs.washington.edu

ABSTRACT

Fault localization (FL) takes as input a faulty program and produces as output a list of code locations ranked by probability of being defective. A programmer doing debugging, or a program repair tool, could save time by focusing on the most suspicious locations.

Researchers evaluate new FL techniques on programs with known faults, and score a technique based on where in its list the actual defect appears. This enables comparison of multiple FL techniques to determine which one is best.

Previous research has primarily evaluated FL techniques using artificial faults, generated either by hand or automatically. Other prior work has shown that artificial faults have both similarities to and differences from real faults; given this, it is not obvious that the techniques that perform best on artificial faults will also perform best on real faults.

This work compares 7 previously-studied FL techniques, both on artificial faults (as a replication study) and on real faults (to validate the assumption that artificial faults are useful proxies for real faults for comparisons of FL techniques). Our replication largely agreed with prior work, but artificial faults were not useful for predicting which FL techniques perform best on real faults.

We also studied which characteristics make FL techniques perform well on real faults. We identified a design space that includes those 7 previously-studied FL techniques as well as 149 new ones, and determined which decisions were most important in designing a new technique.

CCS Concepts

•Software and its engineering → Software testing and debugging;

Keywords

fault localization; debugging; software testing

1. PROBLEM AND MOTIVATION

Dozens of FL techniques have been proposed [19]. It is desirable to evaluate and compare these techniques, both so that practitioners can choose the ones that help them solve their debugging problems, and so that researchers can better build new FL techniques.

A FL technique is valuable if it works on real faults. However, evaluations [1–6, 10, 11, 13–16, 18–20] of FL efficacy have used only 35 real faults of a single small numerical program (`space`) [17], or on artificial faults created by making small, local changes to the program text: for example, replacing `a+5` with `a-5` or `a+0`.

These "mutants" differ from real faults in many respects, including their size, their distribution in code, and their difficulty of being detected by tests [9]. It is possible that an evaluation of FL techniques on real faults would yield different outcomes than previous evaluations on mutants. If so, previous recommendations would need to be revised and previous studies replicated using real faults.

Our contributions include: a replication study on artificial faults, confirming 70% of previous comparisons; a study comparing the same techniques on real faults, contradicting *all* previous comparisons; creation of a design space of FL techniques including many previously studied techniques as well as 149 new hybrids; and identification of the dimensions of the design space most important for good performance on real faults.

2. BACKGROUND AND RELATED WORK

2.1 Evaluation Metrics

To compare two FL techniques, we used the standard metric, EXAM score, which computes the fraction of the way through the statement-ranking the faulty statement appears. (Alternative metrics include LIL [13], T-score [12], and Expense [5].) We had to extend the EXAM score in several ways, such as to account for real-world faults that contain multiple faulty statements.

2.2 Programs and Defects

We used the Defects4J database of real faults [8], which provides faulty and fixed program versions for 357 faults collected from the version-control histories of 5 projects (*JFreeChart*, *Google Closure compiler*, *Apache Commons Lang*, *Apache Commons Math*, and *Joda-Time*). We considered a statement faulty if it was altered by Defects4J's bugfix patch.

We used the Major mutation framework [7] to generate artificial faults by taking each fixed program version in Defects4J, and inserting mutants into the statements that had been changed or inserted by the bugfix. In this way, we generated several artificial faults corresponding to each real fault, located in the same region of the program. After discarding un-localizable artificial faults (e.g., ones that made the program uncompilable, or did not trigger any test failures), we had 2831 artificial faults, corresponding to 327 real faults.

FSE'16, November 13–18, 2016, Seattle, WA, USA
ACM. 978-1-4503-4218-6/16/11...$15.00
http://dx.doi.org/10.1145/2950290.2983967

Table 1: Previously-reported comparisons, and our results for those comparisons on artificial and real faults, including statistical significance and effect size (Cohen's *d*).

Previous comparisons Winner > loser	Artificial agree?	Artificial *d*	Real agree?	Real *d*
Ochiai > Tarantula [10,11,14,18,20]	**yes**	(-0.25)	*(insig.)*	*(-0.11)*
Barinel > Ochiai [3]	**no**	(0.26)	*(insig.)*	*(0.06)*
Barinel > Tarantula [3]	**yes**	*(-0.06)*	*(insig.)*	*(-0.09)*
Op2 > Ochiai [14]	**yes**	*(-0.13)*	no	*(0.14)*
Op2 > Tarantula [13,14]	**yes**	(-0.26)	*(insig.)*	*(0.08)*
D* > Ochiai [10,18]	**yes**	*(-0.14)*	*(insig.)*	*(-0.02)*
D* > Tarantula [6,10,18]	**yes**	(-0.25)	*(insig.)*	*(-0.11)*
Metallaxis > Ochiai [15]	**yes**	*(-0.16)*	**no**	(0.2)
MUSE > Op2 [13]	**no**	*(0.08)*	**no**	**0.81**
MUSE > Tarantula [13]	*(insig.)*	*(0.03)*	**no**	**0.86**

Formatting of agreement indicates p-value: **p<0.01**, p<0.05, *(p≥0.05)*, and of Cohen's *d* indicates effect size: **large**, medium, (small), *(negligible)*.

2.3 FL Techniques

This paper considers two families of FL techniques [19]:

Spectrum-based techniques (SBFL techniques) assign a suspiciousness to each statement based on how many passing and failing tests execute it. The most suspicious statements are those that are covered by many failing tests and few passing tests. We evaluated 5 techniques from this family: Tarantula [5], Ochiai [2], D* [18], Barinel [3], and Op2 [14].

Mutation-based techniques (MBFL techniques) assign a suspiciousness to each mutant. The most suspicious statements are those with mutants that change the behavior of many failing tests and few passing tests. We evaluated 2 techniques from this family: Metallaxis [15] and MUSE [13].

3. ARTIFICIAL VS. REAL FAULTS

3.1 Replication on Artificial Faults

We reproduced several previous comparisons of FL techniques, by running each technique on our set of artificial faults. The left column of table 1 shows the previous results, and the middle columns (labeled "Artificial") show our results. The notable features are:

Small effect sizes. Regardless of any statistical significance, there are no *practically* significant differences (that is, large effect sizes) between any pair of previously-compared techniques. Differences between scores are all small fractions of a standard deviation.

Agreement with most prior SBFL–SBFL comparisons. We agree with 6/7 of the previous SBFL–SBFL comparisons, disagreeing only with the "Barinel > Ochiai" claim.

Disagreement with prior SBFL–MBFL comparisons. Prior studies introducing MBFL techniques found them to be superior to SBFL techniques. We find the opposite. We found that MBFL techniques outperform SBFL on "reversible" mutants (those for which there exists a mutant that can repair the fault), but do very poorly on "irreversible" faults.

3.2 Replication on Real Faults

The columns labeled "Real" in table 1 show the same replication study, performed on real faults instead of artificial faults. Notably:

Insignificant differences between SBFL techniques. There are no large effect sizes and no statistically significant comparisons.

Practical significance: MUSE performs poorly. The only practically significant differences we found show that MUSE performs poorly on real faults. This is due to MUSE's assumption that mutating a faulty statement will (occasionally) fix a failing test: "reversible" faults are common among artificial faults, but very rare among real faults.

In conclusion, previous recommendations should not be followed: on real faults, there are essentially no statistically significant differences between techniques, and what significance there is *contradicts* prior work that used artificial faults.

4. EXPLORATION AND EXTENSION

What design decisions are most important for FL techniques? What choices lead to the best results on real faults?

We identified a design space for FL techniques that includes all 7 techniques of table 1. We observed that all these techniques have a common format: (1) count how many passing/failing tests interact with each program element (e.g., cover a statement, or change from failing to passing due to a mutant, or fail with a different stack trace due to a mutant); (2) assign suspiciousness to elements by plugging those counts into a formula (e.g., Tarantula's or Ochiai's); (3) for MBFL, aggregate mutants' suspiciousnesses by statement (e.g., by averaging or taking the max); and (4) sort statements by suspiciousness. By combining different implementations of these steps, as they appear in the 7 prior techniques, we arrived at a design space containing 156 techniques.

We evaluated all of these techniques on real faults, analyzing (1) which design factors have the greatest impact on FL technique quality and (2) which FL techniques do best.

Results. The choice of spectrum-based technique doesn't matter. All SBFL techniques in our design space performed almost identically (having insignificant differences and/or miniscule effect sizes), supporting the conclusion from table 1 that there are no practically significant differences between spectrum-based techniques.

Variation among mutation-based techniques was both statistically and practically significant. By far the most important factor in the design of a mutation-based technique is the definition of whether a test "interacts" with a mutant. For example, MUSE defines interaction as the mutant changing whether the test passes, which means triggering tests rarely interact with any mutants. Metallaxis uses a more discriminating definition: interaction means producing some change in the test's output, such as changing the message of the exception thrown by a failing test.

Judged by mean EXAM score, the best techniques were consistently spectrum-based (the best being D*), but judged by *median* EXAM score, the best techniques were mutation-based (the best being a variant of MUSE that uses Metallaxis's definition of interaction). The cause of the difference was that (on our set of real faults) mutation-based techniques usually do slightly better than spectrum-based techniques, but have more outliers on which they do exceptionally poorly. Many of these outliers are due to faulty statements being unmutatable and therefore unrankable by MBFL techniques, but others have no clear explanation.

5. CONCLUSION

Fault localization techniques are typically evaluated using artificial faults. We investigated the assumption that artificial faults are good proxies for real faults when evaluating FL techniques. We replicated prior studies using artificial faults, confirming 70% of their results. We replicated them again using real faults, contradicting *all* prior comparisons. To determine what makes FL techniques do well on real faults, we identified common patterns between techniques, created a design space, and determined which parameters are most important to good performance.

6. REFERENCES

[1] R. Abreu, P. Zoeteweij, R. Golsteijn, and A. J. Van Gemund. A practical evaluation of spectrum-based fault localization. *Journal of Systems and Software*, 82(11):1780–1792, 2009.

[2] R. Abreu, P. Zoeteweij, and A. J. Van Gemund. On the accuracy of spectrum-based fault localization. In *Testing: Academic and Industrial Conference Practice and Research Techniques-MUTATION, 2007. TAICPART-MUTATION 2007*, pages 89–98. IEEE, 2007.

[3] R. Abreu, P. Zoeteweij, and A. J. Van Gemund. Spectrum-based multiple fault localization. In *Automated Software Engineering, 2009. ASE'09. 24th IEEE/ACM International Conference on*, pages 88–99. IEEE, 2009.

[4] S. Ali, J. H. Andrews, T. Dhandapani, and W. Wang. Evaluating the accuracy of fault localization techniques. In *ASE*, pages 76–87, Nov. 2009.

[5] J. A. Jones and M. J. Harrold. Empirical evaluation of the Tarantula automatic fault-localization technique. In *ASE*, pages 273–282, Nov. 2005.

[6] X. Ju, S. Jiang, X. Chen, X. Wang, Y. Zhang, and H. Cao. HSFal: Effective Fault Localization Using Hybrid Spectrum of Full Slices and Execution Slices. *Journal of Systems and Software*, 90:3–17, Apr. 2014.

[7] R. Just. The Major mutation framework: Efficient and scalable mutation analysis for Java. In *ISSTA*, pages 433–436, July 2014.

[8] R. Just, D. Jalali, and M. D. Ernst. Defects4J: A Database of existing faults to enable controlled testing studies for Java programs. In *ISSTA*, pages 437–440, July 2014. Tool demo.

[9] R. Just, D. Jalali, L. Inozemtseva, M. D. Ernst, R. Holmes, and G. Fraser. Are mutants a valid substitute for real faults in software testing? In *FSE*, pages 654–665, Nov. 2014.

[10] T.-D. B. Le, D. Lo, and F. Thung. Should i follow this fault localization tool's output? *Empirical Softw. Engg.*, 20(5):1237–1274, Oct. 2015.

[11] T.-D. B. Le, F. Thung, and D. Lo. Theory and practice, do they match? A case with spectrum-based fault localization. In *ICSM*, pages 380–383, Sep. 2013.

[12] C. Liu, L. Fei, X. Yan, J. Han, and S. P. Midkiff. Statistical debugging: A hypothesis testing-based approach. *Software Engineering, IEEE Transactions on*, 32(10):831–848, 2006.

[13] S. Moon, Y. Kim, M. Kim, and S. Yoo. Ask the mutants: Mutating faulty programs for fault localization. In *ICST*, pages 153–162, Apr. 2014.

[14] L. Naish, H. J. Lee, and K. Ramamohanarao. A model for spectra-based software diagnosis. *ACM Transactions on software engineering and methodology (TOSEM)*, 20(3):11, 2011.

[15] M. Papadakis and Y. Le Traon. Metallaxis-FL: Mutation-based fault localization. *STVR*, 25(5-7):605–628, Aug.–Nov. 2015.

[16] R. Santelices, J. Jones, Y. Yu, and M. J. Harrold. Lightweight fault-localization using multiple coverage types. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 56–66, 2009.

[17] F. I. Vokolos and P. G. Frankl. Empirical evaluation of the textual differencing regression testing technique. In *Proceedings of the International Conference on Software Maintenance*, ICSM '98, pages 44–, Washington, DC, USA, 1998. IEEE Computer Society.

[18] W. E. Wong, V. Debroy, R. Gao, and Y. Li. The DStar method for effective software fault localization. *IEEE Trans. Reliab.*, 63(1):290–308, Mar. 2014.

[19] W. E. Wong, R. Gao, Y. Li, R. Abreu, and F. Wotawa. A survey of software fault localization. *IEEE Transactions on Software Engineering (TSE)*, 2016.

[20] J. Xuan and M. Monperrus. Test case purification for improving fault localization. In *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2014, pages 52–63, New York, NY, USA, 2014. ACM.