SCIENCE CHINA
Information Sciences

• LETTER •

September 2022, Vol. 65 199103:1–199103:2
https://doi.org/10.1007/s11432-019-3020-4

Automated regression unit test generation for program merges

Tao JI[1], Liqian CHEN[1*], Xiaoguang MAO[1*], Xin YI[1] & Jiahong JIANG[2]

[1]*College of Computer, National University of Defense Technology, Changsha* 410073*, China;*
[2]*Beijing Institute of Tracking and Telecommunication Technology, Beijing* 100095*, China*

Received 12 December 2019/Revised 13 March 2020/Accepted 16 July 2020/Published online 23 August 2022

Citation Ji T, Chen L Q, Mao X G, et al. Automated regression unit test generation for program merges. Sci China Inf Sci, 2022, 65(9): 199103, https://doi.org/10.1007/s11432-019-3020-4

Dear editor,

Merging of various branches of software with the current branch is common in collaborative software development. After decades of hard work on detecting merge conflicts, various tools have been proposed [1] for assisting software developers in detecting and resolving conflicts. However, software developers are still deeply relying on textual merge tools to handle complicated merge tasks [2]. They may not be able to detect latent semantic merge conflicts, which reduces the software quality.

Recently, Sousa et al. [3] have made progress on guaranteeing the quality of a three-way merge, by proposing the contract of semantic conflict freedom and developing the tool SafeMerge to verify whether a three-way merge meets this contract. However, the above mentioned verification approach still has limitations and faces challenges in verifying merges. First, besides a three-way merge, Git also supports the merging of two branches without the common ancestor (two-way merge) or more than two branches (octopus merge). SafeMerge only supports a three-way merge and thus fails to deal with other common merges. Second, SafeMerge only works on cases wherein two branches make changes to the same Java method. Third, as described in [3], SafeMerge has limitations on changes made to method signatures, an analysis scope, and exceptions.

Regression testing can prevent regression faults and is widely used in real-world software development. Meanwhile, rerunning the test suite seems to be enough for detecting merge conflicts if we have a high-coverage test suite before merging branches. However, in practice, the test suite may still face a high probability in missing the merge conflicts owing to the workflow of collaborative development. Imagine two branches are developed by different software developers. The newly added or changed test cases may not cover changes made by the other branch since the developers are not aware of each other's work. As a result, the conflicts between these changes cannot be detected.

Test oracles. We find some inconsistencies between the contract of semantic conflict freedom and the verification algorithm of SafeMerge, and then revise the algorithm in Appendix A. In Appendix B, we negate the formula that needs to be verified in the revised algorithm. According to the results, we propose general test oracles for all kinds of merges as follows.

Definition 1 (Test oracles on program merges)**.** Given an n-parent merge in which every program version is compilable, to find the merge conflicts, we generate tests, which achieve the following goals.

(1) Unexpected behavior. Suppose that one test case t is generated for the merged program v_m. We say that v_m has some unexpected behavior, if for any parent version v_i, the same assertion φ is violated.

(2) Lost behavior. Suppose that parent versions have the common original v_o. For one parent version v_i, we say its newly introduced behavior is missing after merging, if one test t for v_i fails on v_o and v_m over the same assertion φ.

According to our oracles, for each merge, we repeatedly generate test cases for one target version and then check their executions on all variant versions.

Implementation. As shown in Figure 1, we present the workflow of our tool TOM (testing on merges). Given one n-parent merge, we generate tests for $\{v_m, v_1, \ldots, \text{and } v_n\}$ in an order, as shown in the left part of Figure 1. Then, for each case, according to the extracted dependencies between different code entities, we select the UUTs (units under testing) and then generate tests to reveal the conflicts. The algorithm for selecting UUTs can be found in Appendix C. We employ the advanced test generation tool EvoSuite [4] to implement the test generation for program merges.

We implement the diff-line coverage criterion to guide the search to cover different line-level parts between the target version and its variants. If someone generates a non-flaky test that cannot cover any different parts between two versions, we do not need to re-execute the test on the other version. Otherwise, we execute it on all the variants. Given any two executions of the test on the two versions, we generate assertions on the variables with different values to capture the behavior difference between the two versions. After running one test on all the variants, all the assertions that appear in each execution comparison are extracted. If

* Corresponding author (email: lqchen@nudt.edu.cn, xgmao@nudt.edu.cn)

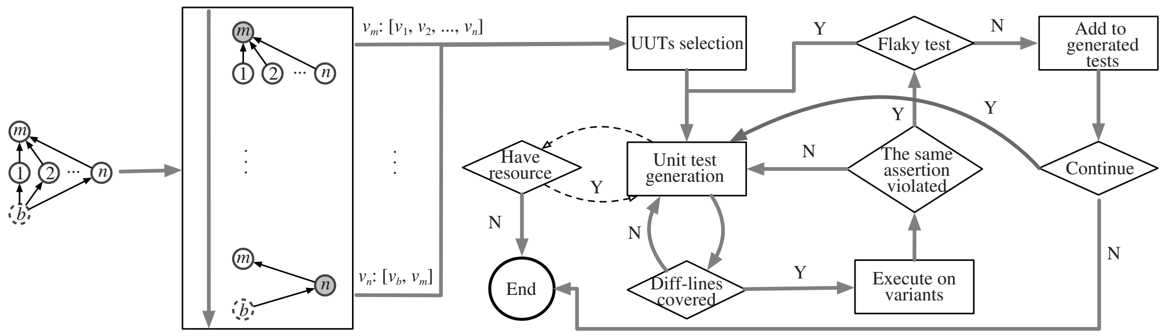

Figure 1 Workflow of TOM.

one test case triggers exceptions for the target version, we leave the case to software developers because the exceptions may not be desired. For each statement having exceptions thrown in the execution on the variant, all assertions based on the execution of the target version are generated to describe the different values and states. If we have the same assertion generated by executions on all variants, we then check the stability of this test by rerunning it five times. If we find one stable test, we add it to the test list, which will be provided to software developers to examine conflicts. As shown in Figure 1, we configure TOM to stop unit test generation once a test case revealing conflicts is generated or the given resource has been consumed (time out).

Evaluation. According to our oracles, detecting semantic conflicts for two-way merges is the sub-task of detecting conflicts for a three-way merge, and one benchmark consisting of three-way and octopus merges is needed. Based on the notion of semantic conflict, we construct conflict merges by making bug-fix tests fail. For the buggy program P_b, we use Major [5] to generate mutants on code statements covered by a bug-fix test. Thus, we have the bug-fixed version P_f and mutant version P_t. After merging P_f and P_t, we execute the bug-fix test case on the merge P_m, and get one conflict merge if this test case fails. For each constructed three-way merge, we create another branch by randomly choosing one mutant of the same class mutated in P_t. Finally, on top of the bug-fix benchmark Defects4J [6], a total of 389 conflict three-way merges and 389 conflict octopus merges are collected. The information of this constructed benchmark MCon4J can be found in Appendix D.

SafeMerge is run on 18 merges in which two branches modify the same Java method to evaluate its effectiveness. However, SafeMerge fails to deal with 17 out of 18 merges, showing the same error, which seems to be related to its implementation. Regarding the remaining merge, we find that the two branches modify the same body of one while loop. The results indicate that SafeMerge fails to deal with the complicated loop in this merge. Unfortunately, we conclude that SafeMerge fails to detect conflicts on all the merges from MCon4J.

Furthermore, we conduct two groups of experiments guided by different coverage criteria. First, we only use the proposed diff-line coverage criterion. Second, we add more coverage criteria used in EvoSuite by default. Because random operators exist in a search process, TOM is run on each three-way merge three times to have a comprehensive view of the tool's performance. There are 42 and 40 conflict three-way merges detected by two groups of experiments. By examining these experimental results, we found that the performance of the multiple criteria is better than the diff-line criterion.

Moreover, a total of 87 conflict octopus merges are detected by TOM. We found that 35 conflict octopus merges created from 35 out of 45 detected conflict three-way merges, are also detected by TOM. A total of 52 conflict octopus merges from newly appeared cases are detected. Note that, we construct octopus merges by adding one mutated branch based on the constructed three-way merge. With more mutated code changing program behaviors, TOM can find more conflict octopus merges than the conflict three-way merges. The detected merges can be found in Appendix E.

Conclusion. We propose test oracles for real-world program merges including two-way, three-way, and octopus merges. On this basis, we implemented a tool called TOM to automatically generate test cases to reveal merge conflicts. In addition, we designed the benchmark MCon4J to support further studies on merges. In our experiments, a total of 45 conflict three-way merges and 87 conflict octopus merges were detected using TOM, while the verification-based tool SafeMerge failed to work on MCon4J.

Acknowledgements This work was supported by National Key R&D Program of China (Grant No. 2017YFB1001802) and National Natural Science Foundation of China (Grant Nos. 61672529, 61872445).

Supporting information Appendixes A–E. The supporting information is available online at info.scichina.com and link.springer.com. The supporting materials are published as submitted, without typesetting or editing. The responsibility for scientific accuracy and content remains entirely with the authors.

References

1 Mens T. A state-of-the-art survey on software merging. IEEE Trans Software Eng, 2002, 28: 449–462

2 Mckee S, Nelson N, Sarma A, et al. Software practitioner perspectives on merge conflicts and resolutions. In: Proceedings of IEEE International Conference on Software Maintenance and Evolution (ICSME), 2017. 467–478

3 Sousa M, Dillig I, Lahiri S K. Verified three-way program merge. In: Proceedings of the ACM on Programming Languages, 2018. 2: 165

4 Fraser G, Arcuri A. Evosuite: automatic test suite generation for object-oriented software. In: Proceedings of the ACM SIGSOFT Symposium and the European Conference on Foundations of Software Engineering (FSE), 2011. 416–419

5 Just R. The Major mutation framework: efficient and scalable mutation analysis for Java. In: Proceedings of the International Symposium on Software Testing and Analysis (ISSTA), San Jose, 2014. 433–436

6 Just R, Jalali D, Ernst M D. Defects4j: a database of existing faults to enable controlled testing studies for java programs. In: Proceedings of the International Symposium on Software Testing and Analysis (ISSTA), 2014. 437–440