Change-Aware Mutation Testing for Evolving Systems

Miloš Ojdanić*
milos.ojdanic@uni.lu
University of Luxembourg
Luxembourg

ABSTRACT

Although the strongest test criteria, traditional mutation testing has shown to not scale with modern incremental development practices. In this work, we describe our proposal of commit-aware mutation testing and introduce the concept of commit-relevant mutants suitable to evaluate the system's behaviour after being affected by regression changes. We show that commit-relevant mutants represent a small but effective set that assesses the delta of behaviours between two consecutive software versions. Commit-aware mutation testing provides the guidance for developers to quantify to which extent they have tested error-prone locations impacted by program changes. In this paper, we portray our efforts to make mutation criteria change-aware as we study characteristics of commit-relevant mutants striving to bring mutation testing closer to being worthwhile for evolving systems.

CCS CONCEPTS

• **Software and its engineering** → **Software testing and debugging**; **Software evolution**.

KEYWORDS

Software Testing, Mutation Testing, Continuous Integration, Continuous Testing, Evolving Systems

ACM Reference Format:
Miloš Ojdanić. 2022. Change-Aware Mutation Testing for Evolving Systems. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '22), November 14–18, 2022, Singapore, Singapore.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3540250.3558911

1 INTRODUCTION

As the software functionalities evolve, it grows in complexity, opening opportunities for experiencing faulty behaviour due to frequent code modifications [18]. Modifications usually result from maintenance, code improvement or introducing a new feature. This continuous development assumes that the previous version of a system is operational, making practitioners particularly interested in not breaking the existing stable version. To assure that the updated software still possesses the functionality it had before the updates, it is important to test the system after each evolution cycle to keep the system behaviour as expected by eliminating occurring faults as early as possible [2]. Thus, automated regression testing is used as gatekeeping to establish confidence that modifications do not break any previously developed functionality, ensuring that the system version is stable and behaves as expected [2].

In such a scenario, developers want a change-aware metric to identify how thorough they have tested the changes and stress their dependencies. In other words, developers are interested in assessing the delta of the behaviour of previous and current versions. Unfortunately, few scientific observations have been devoted to forming and studying such change-aware testing criteria. It is of far-reaching importance to emphasise that such criteria would be a viable solution from both the quality assurance and economic perspectives.

Among many proposed testing techniques to guide toward higher-quality software, mutation testing arose as a technique that evaluates and guides to the most thorough tests [28]. The technique generates slight syntactic program alterations - mutants - as test requirements or criteria to design tests and detect these so-called artificial faults. For a long time, it has been empirically proved that mutants strongly correlate with real faults while mutation score correlates with fault detection [8, 14, 29]. Among all other code coverage criteria (e.g., branch, statements), mutation analysis is long established as the strongest one, guiding the developer to write semantically sensitive tests [3, 8].

Even though that mutation testing is an established testing technique, it assumes the static nature of software and uses blindly the mutation score, which comprises all devised mutants [20]. This strategy inflates the analysis as most mutants do not relate to the task of the code-change in questions. We advocate that we should use only mutants that interact with the changed program behaviours. Thus, this doctoral research proposes and studies code commit-relevant mutants to allow such focused testing. In our work, we first formally define these mutants that serve as change-aware test requirements; when satisfied, exercise the committed code and its integration to the rest of the program under tests. Similarly, we propose a commit-relevant mutation score as a substitute for the traditional mutation score to serve as a metric to judge whether the test suite is adequate in testing commit change and provide guidance for further improvements. It is of far-reaching importance to study the nature and properties of these mutants and their utility over time in continuously evolving systems. Thus, as a continuance of our work, we ventured to study the existence of implicit interaction between the changed and unchanged code parts through the notion of high-order mutants. As described in the paper, our work also studies the advantages of commit-relevant mutants selection compared with the state-of-the-art. In our analysis, we also report on the show-case, describing and reporting the use of commit-relevant mutants in assessing the regression test-case prioritisation methods. After identifying and studying the advantages and properties of

ESEC/FSE '22, November 14–18, 2022, Singapore, Singapore
© 2022 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9413-0/22/11.
https://doi.org/10.1145/3540250.3558911

this category of mutants, as a work in progress, we report on the study concerning the in-time ability of mutants to test evolving systems for a considerable period of time. Thus, extending our work from commit2commit practice (evaluation in a sequential commit by commit manner) to the notion of long-standing mutants.

2 BACKGROUND AND RELATED WORK

Test criteria are metrics based on test requirements that yield specific elements of a software artefact that a test case must cover or satisfy, thus quantifying the testing quality of the system. Different coverage criteria have been proposed and used to guide test generation and selection. Mutation coverage measures how well a test suite can distinguish between the original program and the variant, i.e., the original with slight syntactic deviation - mutant. Well-established terminology defines the status of a variant as killed when it is distinguished by a test case while surviving if not. This variant is called mutant, and its syntactic deviation is traditionally defined by grammar-based rules called mutation operators. These operators are numerous, targeting all code elements exhaustively, resulting in numerous and often impractical mutations.

The number of generated mutants has long been recognized as a problem of mutation testing. Selective mutation started as an approach to constrain mutant generation by applying a set of carefully crafted mutation operators [4, 9, 24, 25, 32–34]. While other studies identified subsumption relationships between mutant operators, thus targeting generation with operators that subsume other operators [15, 16, 35]. That subsumption goes beyond operators to mutant execution behaviour, Ammann et al. [1, 19] recognized and defined subsuming relationship between mutants, thus suggesting that majority of the mutants fall into the redundancy basket. Following this line of work, Kurtz et al. [20] elaborate on test completeness, suggesting that calculating mutation score based on subsuming mutants [1] [1] is superior to the traditional mutation score for determining test completeness, w.r.t., eliminating redundancy. Yet, even though subsuming mutants distinction will recognize all other mutants and accurately measure test completeness, calculating this subsumption relationship depends on the tests themselves, thus, enabling this dynamic action only after the testing process, making the process worth exploring but impractical in real-time unless the mutants are prioritized based on their test completeness advancement probability which requires learning on these mutants properties [17].

Accordingly, some approaches emerged to target the selection of subsuming mutants over all mutants [13, 23]. On the other hand, the problem of commit-relevant test requirements has not been investigated by the literature [28]. A few studies reach to be the closest work to ours, suggesting different approaches to scale mutation testing in CI settings. Incremental Mutation Testing [5] proposes the use of mutants on changed lines. Regression Mutation Testing [37] is an approach that incrementally calculates the mutation score by maintaining a mapping of mutants across program versions and (re)-calculating the mutation score based on the mutants that lie on changes and dependencies. Thus, aiming to speed up test execution while still considering the entire set of mutants when testing evolving software systems. Predictive Mutation Testing [36] is an approach which tries to predict mutation scores based on code features. Existing mutation testing tools use some form of incremental analysis through history logging [21]. Some reports from the industry suggest a random selection of a few mutants from the modified code areas for targeting the intent of code review [30, 31]. Overall, it is clear that the proposed solutions target the entire set of mutants (over-approximating), thus inflating test completeness and introducing noise in the analysis by considering every test requirement blindly without considering the task of the code change in question. Or either select a very few mutants, thus under-approximating change-aware test thoroughness and potentially missing some essential test requirements.

[1]Given a finite set of mutants M and a finite set of tests T, mutant m_i is said to dynamically subsume mutant m_j if some test in T kills m_i and every test in T that kills m_i also kills m_j

3 COMPLETED WORK

3.1 Commit-Aware Mutation Testing

The problematic regression faults arise from the unforeseen interaction of modified and unmodified parts of the code. The change-aware test requirements representing the delta of behaviours between two observed program versions would capture this interaction and allow for accurately and adequately testing of program changes – providing the quantitative metric of the extent of the change-aware test thoroughness. As we already pointed out, the current reported research in this area comes short with such an approach and metric. In intention to fill this gap, we defined and formalized commit-relevant mutants - the mutants that capture the interaction between the behaviours of the versions of the system under change. The formal definition suggests that a mutant becomes relevant if at least one test makes observable any behavioural difference between the version that includes only the mutant (pre-commit version) and the version that includes the committed changes (post-commit) version. These two conditions ensure the presence of observable dependency between a mutant and committed changes, i.e., mutant changes its behaviour due to the code modifications. Thus, distinguishing commit-relevant mutants results in tests capable of detecting any potential faults that depend on the commit (30% more chances of detecting commit introducing faults over state-of-the-art as we will further report). It is true to say that our definition allows the inclusion of mutants that can be killed by tests that are not all relevant to a committed change. Although those mutants are indeed relevant as they depend on the changed part of the code, we also ventured to propose different levels of mutant relevance - considering the strength of dependency between mutants and the changed part of the code. Thus, we defined that the value of relevance lies between 0 and 1, whereas relevance takes a value of 0 if there is no observable difference between a mutant and code change. While the value increases, the mutant becomes more relevant until the point when the observable difference can be detected by every test, making a mutant strongly relevant with relevance value 1. The selection of strong commit-relevant mutants would always lead to tests that exercise the unforeseen interactive dependencies between the changed and unchanged part of the code.

Overall, the first part of this work introduces commit-relevant mutants, while formal definition details can be found in our journal article [22, 26].

3.2 On the Use of Commit-Relevant Mutants

In addition to the introduction and formal definition of commit-relevant mutants, we studied and confirmed our hypothesis that most mutants introduce noise in the mutation score while being irrelevant to the code-change. To be precise – we identified that the portion of relevant mutants to commit change is between 0.5% to 47%. Whereas, for most observed program versions, the set of commit-relevant mutants is small. In contrast, it happened the scenario where the source code under test is not large, though the change locates on the crucial points in the system under test, resulting in more extensive sets. After identifying the distribution of the existence of relevant mutants and substantial noise of irrelevant mutants, we had to check the extent to which the noise influences the mutation score. In particular, we studied whether the traditional mutation score correlates with the relevant mutation score, which serves as evidence of commit-aware test assessment, i.e., reflects the level at which the altered code has been tested. We found no correlation suggesting that the traditional mutation score is a metric that can be used as a proxy for altered code, suggesting that the effect of irrelevant mutant may distort the testing process. Besides, one may suggest that random selection of mutants can significantly kill commit-relevant mutants. We explored this hypothesis by simulating a scenario where a tester analyses mutants and kills them. The simulation comprised the best effort bases, analyzing the same number of mutants. We identified that by analyzing up to 50 mutants, a developer would miss around 50-60% of relevant mutants for both C and Java studied project versions. At the same time, we measure fault detection – how writing tests to kill relevant mutants detect real faults – and detect that targeting commit-relevant mutants leads to a 30-40% of difference in fault detection when compared with random selection and mutants on modification. As one may wonder, we also ventured to explore how different the relevant mutants are from the existing subsuming and hard-to-kill classes. The interesting finding suggests that relevant mutants are also non-subsuming, indicating many redundancies among relevant mutants and further direction to explore. In this work, we also showcase the use of relevant mutants in assessing regression test prioritization methods. Our work argues that the explored standard test case prioritization guiding metrics show relatively small values to detect change-aware mutants. This observation raises the question of how well and complete the prioritization techniques can identify regression faults. We motivate future researchers to explore this line of work and use relevant mutants as guidance and a proxy for the introduced regression faults.

Overall, this work demonstrates the need and advantages of applying commit-aware mutation testing over traditional mutation testing in the environment of continuous software evolution. Additionally, let us point to our published journal article for more details and descriptions [26].

3.3 Approximating Relevant Mutants with High-Order Mutations

Applying mutation testing in continuously evolving systems is not a trivial task due to the scale of possible mutations, program complexity and the difficulty of determining the impact of the dependencies of the program change. Although previously proposed commit-aware mutation testing is a powerful technique, one may argue that it includes complex semantics, viewed through the strict clean test contract assumption - no changed tests between the versions, which is true to say, according to our scientific observations, that is challenging to satisfy assumption in open-source software, while it is common in industrial settings. Another complexity in design can be recognized as a need to analyze and employ a test suite from pre-, and post-commit versions, using test oracle as observational behaviour. Aiming to alleviate these design requirements, we proposed an approach for identifying commit-relevant mutants based on the special notion of observational slicing that employs high-order mutants. In particular, to observe the relevance of a mutant to the point of interest (code change), we rely on second-order mutants, where the comparison mutant is on the changed part of the code and serves to capture the existence of an implicit relationship between a mutant outside the changed code. In short, the intuition is that the mutants located on the modified code (note that the mutant is syntactic change by itself) impact the behaviour of the mutants on the unmodified code, thus making it relevant as it depends on the changed code. More formally, if we consider two first-order mutants Mx and My, where one is located outside the change, and another is located on a change, then the high-order mutant Mxy is their by-product. We say that Mx is commit-relevant if the high order mutant Mxy has different behaviour than Mx and My. This formalization releases strict commit-aware mutation testing design requirements, employing only the post-commit version and its test suite. At the same time, it is essential to note that we even ventured into instrumenting test suites (test assertions), such as observing test inputs/outputs over test oracles to define more fine-grained behaviour and study the mutant impact.

Please refer to our journal-first article for a brother and deeper view of this work [27].

3.4 Studying Relevance of Mutants to Code-Evolution

Due to the diversity and portability of the proposed approaches, we managed to create and study the most extensive dataset of mutants (over 10 million) to date in a continuous integration environment. We set the ground truth for our study for around 300 commits for different projects. Our analysis inquires about the locations of mutants and their scope inside the changed program files as we argue this is the scope of developers interest. In this study, we confirmed the previous finding suggesting that around 30% of mutants are commit-relevant. At the same time, we extended our analysis and focused on a minimal (subsuming) set of commit-relevant mutants. This analysis shows intriguing results and indicates that by selecting only subsuming commit relevant mutants, we reduce the number of mutants requiring analysis by around 93% on average. Furthermore, we report that 69% of the mutants are located outside the changed methods, indicating the importance of testing the interdependency of units (methods) in Java class modules. The merit of the observations is that executing commit-relevant mutants may reduce the test execution compared to a random selection of all mutants in commit-changed files. In our study, we identified a reduction of up to 16 times. Another task we explored relates to the selection of mutants. We identified that standard selection techniques would miss approximately 45% of subsuming commit-relevant mutants when analyzing the scope of up to 20 mutants. We studied commit-relevant mutants features (related to code graphs, abstract syntax trees, mutants operators, etc.), utilities and commit properties and identified that commit-relevant mutants in Java language cannot be reliable and precisely predicted by those features which are all previously reported by the literature for different prediction tasks. Overall our work delivers the characteristics of commit-relevant mutants, where we study their prevalence, location, effectiveness, and efficiency – while we inform on their predictability using the features from the literature and discuss the potential guidelines and implications for practice.

All in all, our published journal article contains more details and descriptions [27].

4 EVALUATION

We performed mutation testing and analyses on both Java and C programming language. For mutants generation, we employed state-of-the-art tools Pitest [21] and Mart [7] and their diverse operators. Our study subjects count around 50 GNU Coreutils [12] shell utility programs producing 500k mutants and 8 Apache Commons Utility [10] programs with around 300 commits resulting in over 10 million mutants for C and Java, respectively. We collected both developer-written and automatically generated tests to obtain a rich test suite that stresses program semantics. We augmented our test suite using KLEE [6] and Evosuite [11] for C and Java, respectively. For our experiments related to fault detection, we used COREBench program versions that introduce faults as they were compatible with the versions we use to study commit-relevant mutants. In addition, it is important to note that in all the parts of our experiments performing statistical analysis, we use three correlation metrics depending on the context: Kendall rank coefficient, Pearson product-moment correlation coefficient and Spearman's rank correlation coefficient. In all cases, we ensured that our results were statistically significant with a significance level of 0.05 and measured the strength of the relationships with effect size. As manually analyzing mutants is a time-consuming and tedious task (no doubt that is unrealistic on the scale of our studies), we performed developer simulations to analyze the importance of commit-relevant mutants. These work simulations have been used thoroughly in various reported studies and set up the scenario in which a developer selects a mutant from a pool – guided by a selection strategy – and writes a test to distinguish it while checking whether the test detects the remaining set of undetected mutants. The procedure is usually repeated 100 times to remove the threat of randomness in the process. Since we perform on a best-effort evaluation, we focus our study on the initial few mutants (up to 20 and 50) that a practitioner would analyze in order to test a commit under test.

5 CURRENT WORK

In-Time Testing of Evolving systems with Long-Standing mutants. The completed work introduced the definition and shows the advantages of obtaining a change-aware metric to evaluate the thoroughness of testing altered software while measuring whether it still possesses its pre-update stability. It is true to say that the technique focuses on impacted mutants that aim to provide guidance for test augmentation and evaluation by targeting the changed program functionality. However, our scientific observation recognizes that the many of the mutants do not change over time, which is especially noticeable once the system reaches a certain level of maturity. Thus, in the current work, we ventured to explore to what extent the mutants and associated mutant selection can provide accurate test assessment over a considerably extended period of time - being diametric to the commit2commit practice we explored so far. We hypothesize that many mutants become obsolete and offer poor test assessment, while we envision devising a technique that selects mutants based on their maturity. In particular, these mutants have the potential to carry the knowledge of dynamic relationships, tackle departed technical debts and evaluate or keep from breaking testing requirements for mature code components. Thus, our current work defines the notion of long-standing mutants, which maximize the return of investment put at a given time - provide test assessment for a prolonged time. In addition to defining the benefits, we demonstrate that mutants have diverse lifetimes over project evolution lifetime and demonstrate that efficient selection of long-standing mutants can provide benefits for at least 10x the amount of time longer than a random selection.

This line of work is still in progress and is yet to be considered for publication.

6 CONTRIBUTION TO KNOWLEDGE

To make the mutation testing technique scale and apply its full fault detection potential in the context of evolving systems, it is necessary to switch the stance from traditional mutation testing and pave the way for future research in the direction of continuous integration. Until the present, we witnessed little effort from the community to tackle the problem, which suggests potential irrecognition of the benefits. We argue that our work can result in a thesis dissertation that provides more than one contribution to the field of mutation testing. First and foremost, our work opens a new direction toward commit-aware mutation testing. We study and report on the usage benefits of this approach over traditional mutation testing and standard random and incremental mutant selection strategies. Additionally, we study the properties and utilities of commit-relevant mutants aiming to inform researchers about their predictable nature - hoping that the future direction can result in a reliable and robust learning selection strategy. In the case of perfect strategy, we inform on the existence of the minimal subset of commit-relevant mutants that significantly reduces the complete number of mutants for change-aware commit testing. Altogether our works pioneer the change-aware mutation testing criteria.

ACKNOWLEDGMENTS

The author is advised by Prof. Mike Papadakis, who is associate professor at the University of Luxembourg. This work has been done under the PayPal project, supported and funded by the SnT - Interdisciplinary Centre for Security, Reliability and Trust at University of Luxembourg and FNR - Luxembourg National Research Fund.

REFERENCES

[1] Paul Ammann, Marcio Eduardo Delamaro, and Jeff Offutt. 2014. Establishing theoretical minimal sets of mutants. In *2014 IEEE seventh international conference on software testing, verification and validation*. IEEE, 21–30.

[2] Paul Ammann and Jeff Offutt. 2016. *Introduction to software testing*. Cambridge University Press.

[3] James H Andrews, Lionel C Briand, Yvan Labiche, and Akbar Siami Namin. 2006. Using mutation analysis for assessing and comparing testing coverage criteria. *IEEE Transactions on Software Engineering* 32, 8 (2006), 608–624.

[4] Ellen Barbosa, José Maldonado, and Auri Vincenzi. 2001. Toward the determination of sufficient mutant operators for C. *Softw. Test., Verif. Reliab.* 11 (06 2001), 113–136. https://doi.org/10.1002/stvr.226

[5] Mark Anthony Cachia, Mark Micallef, and Christian Colombo. 2013. Towards incremental mutation testing. *Electronic Notes in Theoretical Computer Science* 294 (2013), 2–11.

[6] Cristian Cadar, Daniel Dunbar, Dawson R Engler, et al. 2008. Klee: unassisted and automatic generation of high-coverage tests for complex systems programs.. In *OSDI*, Vol. 8. 209–224.

[7] Thierry Titcheu Chekam, Mike Papadakis, and Yves Le Traon. 2019. Mart: a mutant generation tool for LLVM. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1080–1084.

[8] Thierry Titcheu Chekam, Mike Papadakis, Yves Le Traon, and Mark Harman. 2017. An empirical study on mutation, statement and branch coverage fault revelation that avoids the unreliable clean program assumption. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. IEEE, 597–608.

[9] Marcio Eduardo Delamaro, Lin Deng, Vinicius Humberto Serapilha Durelli, Nan Li, and Jeff Offutt. 2014. Experimental Evaluation of SDL and One-Op Mutation for C. In *2014 IEEE Seventh International Conference on Software Testing, Verification and Validation*. 203–212. https://doi.org/10.1109/ICST.2014.33

[10] The Apache Software Foundation. accessed July 14, 2022. Apache Commons. https://commons.apache.org/.

[11] Gordon Fraser and Andrea Arcuri. 2011. 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*. 416–419.

[12] Inc. Free Software Foundation. accessed July 14, 2022. GNU Coreutils shell utility programs. https://www.gnu.org/software/coreutils/.

[13] Aayush Garg, Milos Ojdanic, Renzo Degiovanni, Thierry Titcheu Chekam, Mike Papadakis, and Yves Le Traon. 2022. Cerebro: Static Subsuming Mutant Selection. *IEEE Transactions on Software Engineering* (2022), 1–1. https://doi.org/10.1109/TSE.2022.3140510

[14] René Just, Darioush Jalali, Laura Inozemtseva, Michael D Ernst, Reid Holmes, and Gordon Fraser. 2014. Are mutants a valid substitute for real faults in software testing?. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. 654–665.

[15] Gary Kaminski, Paul Ammann, and Jeff Offutt. 2011. Better Predicate Testing. In *Proceedings of the 6th International Workshop on Automation of Software Test* (Waikiki, Honolulu, HI, USA) *(AST '11)*. Association for Computing Machinery, New York, NY, USA, 57–63. https://doi.org/10.1145/1982595.1982608

[16] Gary Kaminski, Paul Ammann, and Jeff Offutt. 2013. Improving Logic-Based Testing. *J. Syst. Softw.* 86, 8 (aug 2013), 2002–2012. https://doi.org/10.1016/j.jss.2012.08.024

[17] Samuel J Kaufman, Ryan Featherman, Justin Alvin, Bob Kurtz, Paul Ammann, and René Just. 2022. Prioritizing mutants to guide mutation testing. In *Proceedings of the 44th International Conference on Software Engineering*. 1743–1754.

[18] Gene Kim, Patrick Debois, John Willis, and Jez Humble. 2016. *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations*. IT Revolution Press.

[19] Bob Kurtz, Paul Ammann, Marcio E Delamaro, Jeff Offutt, and Lin Deng. 2014. Mutant subsumption graphs. In *2014 IEEE Seventh International Conference on Software Testing, Verification and Validation Workshops*. IEEE, 176–185.

[20] Bob Kurtz, Paul Ammann, Jeff Offutt, and Mariet Kurtz. 2016. Are We There Yet? How Redundant and Equivalent Mutants Affect Determination of Test Completeness. *IEEE International Conference on Software Testing, Verification and Validation*, 142–151. https://doi.org/10.1109/ICSTW.2016.41

[21] Thomas Laurent, Mike Papadakis, Marinos Kintis, Christopher Henard, Yves Le Traon, and Anthony Ventresque. 2017. Assessing and Improving the Mutation Testing Practice of PIT. *IEEE International Conference on Software Testing, Verification and Validation (ICST)*, 430–435. https://doi.org/10.1109/ICST.2017.47

[22] Wei Ma, Thomas Laurent, Miloš Ojdanić, Thierry Titcheu Chekam, Anthony Ventresque, and Mike Papadakis. 2020. Commit-aware mutation testing. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 394–405.

[23] Michaël Marcozzi, Sébastien Bardin, Nikolai Kosmatov, Mike Papadakis, Virgile Prevosto, and Loïc Correnson. 2018. Time to Clean Your Test Objectives. In *Proceedings of the 40th International Conference on Software Engineering* (Gothenburg, Sweden) *(IEEE/ACM International Conference on Software Engineering '18)*. Association for Computing Machinery, New York, NY, USA, 456–467. https://doi.org/10.1145/3180155.3180191

[24] A.J. Offutt, G. Rothermel, and C. Zapf. 1993. An experimental evaluation of selective mutation. In *Proceedings of 1993 15th International Conference on Software Engineering*. 100–107. https://doi.org/10.1109/ICSE.1993.346062

[25] A. Jefferson Offutt, Ammei Lee, Gregg Rothermel, Roland H. Untch, and Christian Zapf. 1996. An Experimental Determination of Sufficient Mutant Operators. *ACM Trans. Softw. Eng. Methodol.* 5, 2 (apr 1996), 99–118. https://doi.org/10.1145/227607.227610

[26] Miloš Ojdanić, Wei Ma, Thomas Laurent, Thierry Titcheu Chekam, Anthony Ventresque, and Mike Papadakis. 2022. On the use of commit-relevant mutants. *Empirical Software Engineering* 27, 5 (2022), 1–31.

[27] Milos Ojdanic, Ezekiel Soremekun, Renzo Degiovanni, Mike Papadakis, and Yves Le Traon. 2022. Mutation Testing in Evolving Systems: Studying the relevance of mutants to code evolution. *ACM Transactions on Software Engineering and Methodology* (2022).

[28] Mike Papadakis, Marinos Kintis, Jie Zhang, Yue Jia, Yves Le Traon, and Mark Harman. 2019. Mutation testing advances: an analysis and survey. In *Advances in Computers*. Vol. 112. Elsevier, 275–378.

[29] Mike Papadakis, Donghwan Shin, Shin Yoo, and Doo-Hwan Bae. 2018. Are mutation scores correlated with real fault detection? a large scale empirical study on the relationship between mutants and real faults. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*. IEEE, 537–548.

[30] Goran Petrović and Marko Ivanković. 2018. State of mutation testing at google. *IEEE/ACM International Conference on Software Engineering*, 163–171. https://doi.org/10.1145/3183519.3183521

[31] Goran Petrović, Marko Ivanković, Bob Kurtz, Paul Ammann, and René Just. 2018. An industrial application of mutation testing: Lessons, challenges, and research directions. *IEEE International Conference on Software Testing, Verification and Validation*, 47–53. https://doi.org/10.1109/ICSTW.2018.00027

[32] Akbar Siami Namin and James Andrews. 2006. Finding Sufficient Mutation Operators via Variable Reduction. *2nd Workshop on Mutation Analysis (Mutation 2006 - ISSRE Workshops 2006), MUTATION'06* (11 2006). https://doi.org/10.1109/MUTATION.2006.7

[33] Akbar Siami Namin, James Andrews, and Duncan Murdoch. 2008. Sufficient mutation operators for measuring test effectiveness. In *2008 ACM/IEEE 30th International Conference on Software Engineering*. 351–360. https://doi.org/10.1145/1368088.1368136

[34] W.Eric Wong and Aditya P. Mathur. 1995. Reducing the cost of mutation testing: An empirical study. *Journal of Systems and Software* 31, 3 (1995), 185–196. https://doi.org/10.1016/0164-1212(94)00098-0

[35] Xiangjuan Yao, Mark Harman, and Yue Jia. 2014. A Study of Equivalent and Stubborn Mutation Operators Using Human Analysis of Equivalence. In *Proceedings of the 36th International Conference on Software Engineering* (Hyderabad, India) *(IEEE/ACM International Conference on Software Engineering 2014)*. Association for Computing Machinery, New York, NY, USA, 919–930. https://doi.org/10.1145/2568225.2568265

[36] Jie Zhang, Lingming Zhang, Mark Harman, Dan Hao, Yue Jia, and Lu Zhang. 2019. Predictive Mutation Testing. *IEEE Transactions on Software Engineering* 45, 9 (2019), 898–918. https://doi.org/10.1109/TSE.2018.2809496

[37] Lingming Zhang and Darko Marinov. 2012. Regression Mutation Testing. *The ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*, 341.