Empirical Software Engineering (2022) 27: 132
https://doi.org/10.1007/s10664-022-10177-8

Mutation testing in the wild: findings from GitHub

Ana B. Sánchez[1] · Pedro Delgado-Pérez[2] · Inmaculada Medina-Bulo[2] · Sergio Segura[1]

Accepted: 16 May 2022 / Published online: 18 July 2022

Abstract

Mutation testing exploits artificial faults to measure the adequacy of test suites and guide their improvement. It has become an extremely popular testing technique as evidenced by the vast literature, numerous tools, and research events on the topic. Previous survey papers have successfully compiled the state of research, its evolution, problems, and challenges. However, the use of mutation testing in practice is still largely unexplored. In this paper, we report the results of a thorough study on the use of mutation testing in GitHub projects. Specifically, we first performed a search for mutation testing tools, 127 in total, and we automatically searched the GitHub repositories including evidence of their use. Then, we focused on the top ten most widely used tools, based on the previous results, and manually revised and classified over 3.5K GitHub active repositories importing them. Among other findings, we observed a recent upturn in interest and activity, with Infection (PHP), PIT (Java) and Humbug (PHP) being the most widely used mutation tools in recent years. The predominant use of mutation testing is development, followed by teaching and learning, and research projects, although with significant differences among mutation tools found in the literature—less adopted and largely used in teaching and research—and those found in GitHub only—more popular and more widely used in development. Our work provides a new and encouraging perspective on the state of practice of mutation testing.

Keywords Mutation testing · Mutation tools · Data mining · GitHub

Communicated by: Lei Ma

✉ Ana B. Sánchez
anabsanchez@us.es

Pedro Delgado-Pérez
pedro.delgado@uca.es

Inmaculada Medina-Bulo
inmaculada.medina@uca.es

Sergio Segura
sergiosegura@us.es

1 SCORE Lab, I3US Research Institute, Universidad de Sevilla, Seville, Spain

2 Escuela Superior de Ingeniería, Universidad de Cádiz, Cádiz, Spain

1 Introduction

Mutation testing measures the effectiveness of tests based on their ability to detect artificial faults. Such faults, so-called *mutants*, are created by applying syntactic changes to the original program, for example, by replacing a relational operator by another one: $a < b \rightarrow a > b$. The types of changes applied for the generation of mutants are determined by transformation rules called *mutation operators*. The main outcome of mutation testing is the so-called *mutation score*, which measures the effectiveness of the test set as the ratio of detected mutants over the total number of mutants. Any mutant not being detected by the tests provides helpful information for including additional tests, and therefore guides the improvement of the test set.

Since its introduction back in the 70s (DeMilo et al. 1978, 1979), research on mutation testing has thrived until becoming a well-established testing technique. In a recent survey by Papadakis et al. (2019), the authors identified more than 400 papers published in the time period 2008-2017 and 87 different mutation tools for a variety of programming languages and artifacts including Java, C, C++, C#, JavaScript, HTML/CSS, Ruby, and UML models, among many others. Mutation testing is a recurrent topic in most software testing and software engineering venues, being the central topic of the series of international workshops on mutation (celebrating its 17th edition in 2022 (https://icst2022.vrain.upv.es/home/mutation-2022/)) and several special issues in top-tier journals (e.g., Just et al. 2019; Papadakis and Just 2017).

Previous survey papers on mutation testing have successfully compiled the state of research, its evolution, fundamental problems, and challenges (Jia and Harman 2011; Offutt and Untch 2001; Papadakis et al. 2019). They all agree on the relevance of mutation testing in the research arena, especially as a common experimental methodology for evaluating the effectiveness of testing techniques. Some authors have also explored the benefits and limitations of using mutation testing in open-source applications (Just et al. 2014) and industrial projects (Delgado-Pérez et al. 2018). Some recent papers also investigate the use of mutation testing in large companies such as Google (Petrovic et al. 2021b) and Facebook (Beller et al. 2021). However, the overall impact of mutation testing beyond research is still limited (Arcuri 2018) and mostly unexplored: to what extent is mutation testing adopted in practice? is the impact of the technique on research reflected in real-world software projects? which kind of projects is the technique used for? which are the most popular mutation testing tools in practice? how has the use of mutation testing changed over the years? These are some of the questions addressed in our work.

In this article, we report the findings of a mining study in GitHub on the use of mutation testing in practice. Specifically, we searched and analyzed the GitHub repositories including evidence of the use of existing mutation testing tools. The study was performed in three steps. First, we performed a thorough search for mutation testing tools, identifying a total of 127 tools released in the period 2001-2021, 40 of them new with respect to the tool compilation by Papadakis et al. (2019). Then, we performed a systematic search for GitHub repositories including evidence of the use of the mutation tools found. For this step, we resorted to both automated searches in the web search interface of GitHub and its GraphQL API (https://docs.github.com/en/graphql). Finally, we selected the top 10 more popular tools, based on the results of the previous search, and we manually revised and classified over 3.5K active repositories including evidence of their use.

Among the numerous findings, we observed a significant boost in the number of mutation tools and the number of GitHub repositories including traces of their use in recent years. Infection (https://infection.github.io/) (a mutation tool for PHP) is by far the most widely used tool in GitHub—imported in about one third of the repositories found—followed by PIT (Coles et al. 2016) (Java), Humbug (https://github.com/humbug/humbug) (PHP), StrykerJS (https://stryker-mutator.io/docs/stryker-js/introduction) (JavaScript), and Mutant (https://github.com/mbj/mutant) (Ruby). The predominant use of mutation testing is development, followed by teaching and learning, and research. We traced 21% of the repositories to industry, 6.8% to academia, and 3.4% to public institutions (e.g., HM Courts & Tribunals Service from the UK government) and open-source projects (e.g., phpMyAdmin). Moreover, the identification of mutation tools through two different sources —by literature review and GitHub search— allowed us to assess the current status of mutation testing from a new point of view with respect to previous studies on the topic. Overall, we observed significant differences among the mutation tools reported in the literature and those found in GitHub and built around open-source communities. Judging by our results, the former are less spread and have a greater presence in the set of projects related to teaching and research. The latter, in contrast, are significantly more popular and are mostly used in development projects. This leads to mixed conclusions. On the one hand, the degree of adoption is encouraging with the three most widely used tools referenced above being imported in more than 500 repositories, some of them being quite active —in terms of commits—and popular—in terms of watchers, stars or forks. On the other hand, the results show a gap between the state of research and practice with some of the most widely used tools rarely being referenced in research papers (e.g., Infection, Humbug or StrykerJS).

The results of our study have direct applications such as pointing readers to the most widely used tools in practice and promoting the use of the technique beyond academia, based on its current adoption in industrial projects. However, the main contribution of our work lies on its novel perspective on the state of practice of mutation testing. This new view on mutation opens new and promising research directions by leveraging the collected data. Among others, we envision new empirical studies, both quantitative and qualitative, that complement our findings and delve into their causes, e.g., conducting surveys among the developers of the most relevant projects. Finally, our work paves the path for new mining studies that contribute to closing the gap between research and practice on mutation testing.

The rest of the article is organized as follows. Section 2 introduces mutation testing technique. Our review method and research questions are described in Section 3. The results of our study are detailed in Section 4. Section 5 proposes new promising research directions to complement our findings. Section 6 discusses related work on mutation testing and mining studies on GitHub. The potential threats to validity are discussed in Section 7. Finally, we conclude the article in Section 8.

2 Mutation Testing

Mutation testing is a well-known fault-based technique to evaluate and improve the quality of test suites. This technique not only encourages testers to exercise as much code as possible but also to uncover possible mistakes made by their programmers. In the absence of real faults, simple syntactic modifications are inserted in the hope that they resemble plausible real faults (Competent Programmer Hypothesis (Acree et al. 1979)) and that those simple changes are able to reveal other more complex ones (Coupling Effect Hypothesis (DeMillo et al. 1978)). The injected faults are known as *mutations* and the new faulty versions of the program under test, as *mutants*. Once the mutants have been generated, each mutant as well as the original program is executed against the test suite to produce an output. An observable difference between the output of the original version and a mutant serves to classify that mutant as detected or *killed*. On the contrary, when the output is the same, the mutant remains *alive* and requires further analysis, as it can point out a deficiency in the fault detection ability of the test suite. This is not always the case, however, because a mutant can turn out to be functionally equivalent to the original program; these are the so-called *semantically-equivalent* mutants. It follows that a tester should aspire to kill as many mutants as possible to increase the detection power of the test suite. The number of killed mutants over the whole set of non-equivalent mutants is called *mutation score*.

The injection of mutations is generally systematized with the development of *mutation tools*, which implement different *mutation operators*. These operators are applied each time a pattern is found in the program (e.g., each appearance of the relational operator '>' is replaced by '<'). As shown later on, more than one hundred mutation testing tools have been developed in the past for most of the widely-used programming languages, including Java, C/C++, Python or C# among many others, and the list has grown in the last years with new mutation tools for emerging domains such as Deep Mutation systems (Ma et al. 2018) or Smart Contracts (Li et al. 2019).

Mutation tools can be used or integrated into other software projects for testing purposes through different means including executable files (e.g., https://github.com/mull-project/mull/releases), static libraries (e.g., https://cs.gmu.edu/~offutt/mujava/#Links), build tools like Gradle or Maven (e.g., https://pitest.org/quickstart/maven/, https://gradle-pitest-plugin.solidsoft.info/), and IDE plugins (e.g., https://github.com/gomezabajo/Wodel/wiki/Get-Started). The rationale behind our work is that most of these mechanisms can be traced back to the code of the application under test leaving evidence of the use of the mutation tools.

3 Review Method

In this section, we describe the systematic approach followed for studying the use of mutation testing in practice by looking into GitHub. Specifically, we detail the research questions and the data collection process. All the searches reported were performed on June 2021.

3.1 Research Questions

We aim to answer the following research questions (RQs):

RQ1: *What is the current tool support for mutation testing?* As a first step, we aim to study the tool support for mutation testing by looking at the mutation tools that have been proposed over the years and their target artifacts.

RQ2: *To which extent is mutation testing adopted in practice?* We aim to study the adoption of mutation testing by examining the number of GitHub projects including evidence of the use of mutation testing tools. We also intend to identify the most frequently used tools and usage patterns.

RQ3: *Which type of projects are mutation testing tools used for?* We aim to analyze the use of each mutation testing tool on different types of projects including teaching, research, and software development. We are particularly interested in studying the use of mutation testing beyond academia.

RQ4: *Which is the activity and relevance of the projects using mutation tools?* Finally, we want to evaluate how active and popular within the community are these projects by looking at different statistics of the GitHub repositories, including commits (activity), and contributors, watchers, stars and forks (popularity). While this analysis cannot give evidence of the particular use given to the mutation tools, it can shed light on the tools most used in relevant projects (i.e., those highly active and popular). This can be a valuable source of information for further analyses which attempt to assess the current practices of developers and companies regarding mutation testing.

3.2 Data Collection and Analysis

Data collection was performed in three steps, graphically depicted in Fig. 1. First, we performed a systematic search for mutation testing tools (RQ1). Second, we conducted a systematic search for GitHub repositories including evidence of use of those tools and, based on the previous findings, we selected and analyzed the 10 tools most frequently used in practice (RQ2). Third, we performed a thorough manual revision and classification of the GitHub repositories using the ten most widely used tools identified in the previous step (RQ3 and RQ4). In what follows, we describe these steps in detail.

3.2.1 Mutation Tools Search

We started by performing a systematic search for mutation testing tools in three steps.

Previous tool compilation First, we selected the tools identified in the survey on mutation testing by Papadakis et al. (2019) including papers published between 2008 and 2017. In

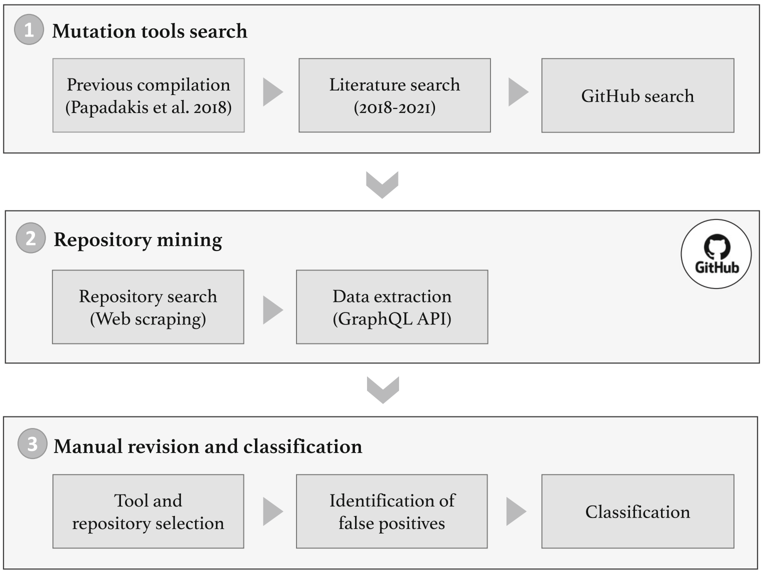

1 Mutation tools search
Previous compilation (Papadakis et al. 2018)
Literature search (2018-2021)
GitHub search
2 Repository mining
GitHub
Repository search (Web scraping)
Data extraction (GraphQL API)
3 Manual revision and classification
Tool and repository selection
Identification of false positives
Classification

Fig. 1 Data collection and analysis process

particular, the survey includes a compilation of the mutation testing tools introduced or used in the surveyed papers. This yielded an initial set of 87 mutation testing tools.

Literature search Second, we performed a systematic review of the literature for papers proposing mutation testing tools published between 2018 (the year in which the survey by Papadakis et al. was published) and May 2021. In contrast to the survey of Papadakis et al., focused on mutation testing advances in general, we were only interested in papers referencing mutation testing tools, and therefore we adjusted our search strings accordingly. Specifically, we searched for papers including the terms *"mutation OR mutation testing OR mutant generation"* and *"tool OR system OR application OR framework"* within the paper title, abstract and/or keywords. The search was performed in the online repositories of Scopus, ACM and IEEE Xplore. As a result, we obtained 84 unique papers meeting the search criteria. Then, we performed a manual revision of the papers for checking whether they proposed a new mutation testing tool. This was an iterative process where all the authors participated by reviewing their assigned papers and later meeting together until reaching a consensus in the cases of doubts. We excluded 49 papers not presenting actual mutation tools (e.g., theoretical studies) or presenting tools already identified. Original tools and their extensions are considered together in our work. As a result, we identified 31 new mutation testing tools. The papers identified in the literature search can be found in the online resource 1 of the supplementary material (https://doi.org/10.5281/zenodo.5713585).

GitHub search During the preparation of our work, we identified some mutation testing tools that were highly popular in GitHub but, interestingly, were not referenced in the research literature. In an attempt to complement the set of tools with those, we performed an additional search in GitHub for mutation testing tools repositories including the search strings "mutation testing" sorted by "Most stars". GitHub stars are recognized as a reliable source of the popularity of GitHub projects (Borges et al. 2016). The search returned hundreds of results. We analyzed the fifteen top results (i.e., those shown in the first results page), and we selected 9 new mutation tools not included in the tool set derived from the previous tool compilation and our literature search.

Overall, we identified 127 mutation testing tools: 87 from the literature review by Papadakis et al. (2019), 31 from an updated literature search, and 9 tools selected among those with more stars in GitHub. Table 3 shows the list of all mutation testing tools, including source, name, release year and domain.

3.2.2 Repository Mining in GitHub

Our work is based on the observation that most mutation tools leave evidence of their (potential) use in the code repository of the applications under test. These pieces of evidence include references to static libraries, dependency declarations, or configuration files, among others. Based on this, we mined data from GitHub in two steps. First, we searched for repositories including evidence of the mutation testing tools under study scraping the web interface of GitHub. We resorted to the web interface, instead of only the GitHub APIs, because the API constrains searches to user-specified repositories when filtering by code specifying file names and text content. However, that information was not known in advance. Once we had the repositories that met the specified code constraints, in a second step we collected detailed information about each repository using the GitHub GraphQL API (https://docs.github.com/en/graphql). Next, we explain both steps.

Repository search The search for repositories including evidence of use of the mutation tools was performed on the web search interface of GitHub. The search was performed automatically using web scraping, as described later on. It is worth noting that from December 2020 GitHub only indexes repositories that have had recent activity within the last year, this is, those that have had a commit or have shown up in a search result in the 12 months[1]. Hence, the scope of our search is limited to those GitHub repositories with recent activity at the time of performing our search, in June 2021. Finally, note that repositories forked from the main repository of the mutation tools under study do not appear in the search; forked repositories will be treated separately as a complementary metric of the mutation tool's relevance.

As an initial step, we designed the search queries for each mutation tool. Specifically, we analyzed the papers and official sites of each mutation tool to find out the different ways in which it can be used in third-party projects. For example, PIT can be integrated by directly importing the "pitest.jar" file or, alternatively, it can be automatically imported using tools like Maven (https://maven.apache.org/) or Gradle (https://gradle.org/), in which case their respective configuration files must include the string "org.pitest". We used this information to define queries for the search of repositories including evidence of use of the mutation testing tools under study. For instance, we searched for projects including the string "org.pitest" in pom.xml or .gradle files and therefore potentially using the tool PIT. This procedure could not be done however for some of the tools: those with no name, and those where a website was not available or whose website did not provide enough information to search for evidence of their use. As a result, we could define the search strings for 55 out of the initial 127 tools. The search strings and the number of repositories matching the queries for each mutation tool, 6,633 in total, can be found in the online resource 3 of the supplementary material (https://doi.org/10.5281/zenodo.5713585). Note that when using the query parameter "filename:" (e.g., filename:Jumble), the GitHub engine searches for files starting by the specified name followed by any possible string, (e.g., "Jumble-annotations").

For automating the search process, we implemented a web scraper using the tool HtmlUnit (https://htmlunit.sourceforge.io). The scraper automatically logs into GitHub and runs the customized queries for each tool in the web search form of GitHub, iterating over all the result pages until collecting all the names and owners of the returned repositories. Duplicates were automatically discarded. The initial search yielded 6,633 repositories including evidence of use of 35 out of the 55 mutation tools under study.

Data extraction We proceeded to collect detailed information about each repository. Specifically, we used the GraphQL API (https://docs.github.com/en/graphql) of GitHub to automatically extract data from the list of 6,633 repositories obtained in the previous step, that is, those GitHub projects potentially using the mutation testing tools under study. In particular, we automatically collected the following data of each repository: name, description, creation date, last update, URL, owner name, owner bio, owner login (unique username), company, country; primary language, secondary languages, license, topics, number of commits, date of first and last commit, number of issues, number of contributors, watchers, stars, and forks. We should note that some of the projects found in GitHub had been migrated from other GIT-based platforms and, therefore, the number of commits of those projects

[1]https://github.blog/changelog/2020-12-17-changes-to-code-search-indexing/

includes the commits received since the creation of the original repository. Collected data was saved into CSV files for later analyses.

3.2.3 Manual Revision and Classification

At this stage, we manually reviewed the data collected in three steps, namely:

Tool and repository selection As a part of the work, we performed a manual qualitative analysis of the selected repositories (see details below). To make this analysis affordable, we first reduced the number of mutation tools under study. Specifically, we selected the 10 mutation tools imported from a higher number of GitHub repositories (ranging from 42 repositories to 1,717) according to our previous search. This decision was also motivated by the fact that the rest of the mutation tools were found only in a few repositories and therefore it was difficult to draw meaningful conclusions from them. This reduced the set of repositories from 6,633 to 6,307. Then, we further reduced the set of repositories, by selecting those having at least one commit in the last year; we refer to these as *active* repositories. This choice allowed us to filter out repositories with no recent activity making our work affordable and, at the same time, to show a more precise picture of the adoption of mutation testing tools at the time of writing this paper. This step reduced the target set of repositories from 6,307 to 3,644.

Identification of false positives False positives are repositories matching the search criteria, but not referencing actual mutation tools. For example, we found some repositories[2] including the common word "jumble" not referencing the actual mutation testing tool for Java Jumble (Irvine et al. 2007). To identify and discard false positives, we split the active repositories among the authors, who carefully checked whether they included actual references to the target mutation tools. At this step, we also removed those repositories not available at the time of the revision (e.g., repositories removed by their owners). Finally, we identified the main repository in GitHub of these mutation tools, which are also excluded from the counting. This makes a total of 3,581 repositories under study.

Table 1 shows the ten mutation testing referenced from a higher number of repositories. For each tool, the table shows the search queries used in GitHub, the total number of repositories found, and the number of active repositories, i.e., projects with at least a commit in the last year. The reduction in the number of repositories varies significantly from one tool to another. It is notable the reduction of Humbug repositories, which started with 1,717 and is finally narrowed down to 636 active projects (i.e., with at least one commit in the last year). This is probably because Infection has replaced Humbug in recent years —as explained in the main repository of Humbug (https://github.com/humbug/humbug)—, so many repositories still referencing that tool may have become obsolete. MuJava also decreases considerably the number of active repositories from 50 to 9. We observed that many repositories using MuJava were created by students to accomplish university course projects; these are mainly single-use repositories, usually abandoned once the corresponding assignment expires.

Classification Finally, we manually classified the active repositories referencing the 10 selected tools, 3,581 in total. Specifically, repositories were classified according to their

[2]False positive of Jumble: https://github.com/PABeckett/Jumble-Solver

Table 1 GitHub search queries and results of repositories for each mutation tool

Name	Search queries	#Repos	#Active repos
Humbug	1. filename:humbug extension:phar	1,717	636
	2. filename:humbug.json extension:dist		
	3. infection/infection filename:composer extension:json		
Infection	1. filename:infection extension:phar	1,671	1,213
	2. filename:infection.json extension:dist		
	3. infection/infection filename:composer extension:json		
Major	1. XMutator filename:build extension:xml	42	10
	2. XMutator filename:run extension:sh		
MuJava	1. filename:mujava extension:jar extension:config	50	9
	2. filename:jmutation extension:jar extension:config		
Mutant	1. mutant-rspec extension:gemspec	749	296
	2. mutant-rspec filename:Gemfile		
	3. mutant-minitest extension:gemspec		
	4. mutant-minitest filename:Gemfile		
Mutmut	1. mumut filename:setup extension:cfg	56	54
	2. mutmut filename:travis extension:yml		
	3. mutmut filename:Makefile		
MutPy	1. mutpy filename:tox extension:ini	59	24
	2. mutpy filename:mut extension:py		
	3. mutpy filename:setup extension:py		
	4. mut.py filename:Makefile		
PIT	1. filename:pitest extension:jar		
	2. org.pitest filename:pom extension:xml		
	3. org.pitest extension:gradle		
	4. org.pitest filename:build extension:sbt	1,340	816
	5. org.pitest filename:ivy extension:xml		
	6. info.solidsoft.pitest filename:build extension:gradle		
	7. pitmp-maven-plugin filename:pom extension:xml		
StrykerJS	1. filename:stryker.conf extension:js extension:json	581	487
Stryker.NET	1. filename:stryker-config extension:json	42	36
Total		6,307	**3,581**

purpose (teaching, learning, research, development or extension) and origin (academia, industry or public institution). The details of the classification process are given in Section 4.3. The repositories were distributed among the authors for their review and classification. Given the high number of repositories, each repository was initially assigned to one of the authors for its analysis. Doubtful cases were highlighted and additional remarks were included when necessary or deemed appropriate for later revision. The process involved several meetings where the authors refined the classification criteria and discussed doubts until reaching a consensus. We resorted to browser extensions to automatically translate files not written in English.

4 Results

In this section, we present the results of our study and how they answer the target research questions.

4.1 Mutation Testing Tools

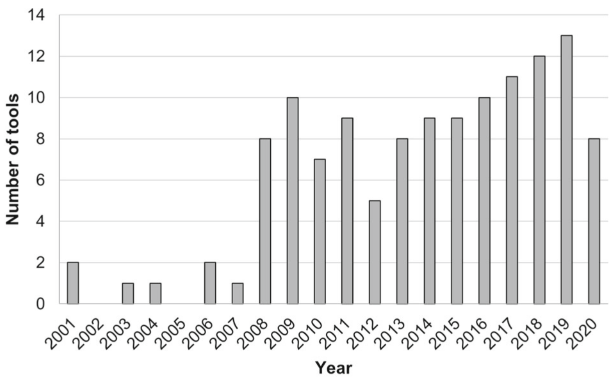
This section addresses RQ1 by studying the current tool support for mutation testing. Table 3 (Appendix) summarizes the list of mutation tools found, 127 in total, released between 2001 and 2020. For each tool, the table shows the source (survey of Papadakis et al. (2019), literature search or GitHub search), its name, release year and domain. Most tools address specific programming languages like Java, C, C++, HTML, JavaScript, or Python, among others. Other tools target high-level artifacts such as design models (e.g., UML) or specification languages (e.g., Z). Finally, a diverse group of tools mutates other types of artifacts including spreadsheets, annotations, regular expressions, security policies, and smart contracts, among others. It is noteworthy that about 35% (45 out of 127) of the tools found had no name, which suggests that they are research prototypes or proofs of concept.

Figure 2a shows the number of mutations tools released per year. The first mutation tools reported— Jester (http://jester.sourceforge.net/) for Java and Proteum (Delamaro et al. 2001) for C— were proposed back in 2001. The graph shows a noticeable increase in interest starting in 2008 to date, with between 5 and 13 mutation tools created per year. It is worth noting the peak in the number of tools proposed in the years 2017-2019. The overall increasing trend is clearly observed in the cumulative number of mutation tools per year in Fig. 2b.

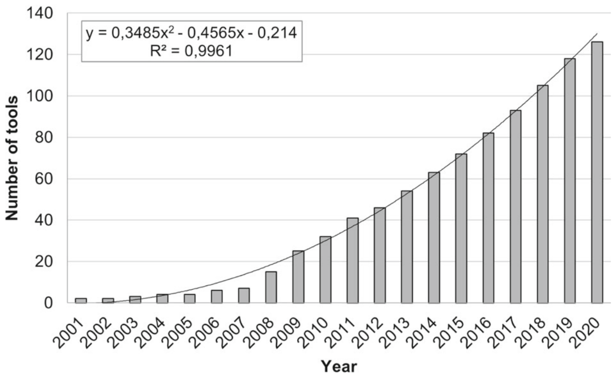
Figure 3 depicts the classification of mutation testing tools based on their target artifacts. Java is the predominant target language (16%), followed by C/C++ (14%), models and specification languages (10%), web-related technologies (7%), Android applications (5%), and security-related artifacts (4%). About one third of the tools found (34%)—those classified as “Others”—target other types of artifacts (e.g., spreadsheets, regular expressions, code annotations, etc.), and programming languages such as Smalltalk, PHP or Prolog, among others.

When looking into recent years, the interest in languages like Java and C/C++ seems to diminish in favor of other languages like JavaScript, Python, Swift, and Solidity (smart contracts), among others. Out of the 34 mutation testing tools proposed in the time range 2018-2020, only 6 tools target Java or C/C++ programs. This may be explained by the fact that already exist a good number of mutation tools for Java and C/C++.

We observed a significant divergence among the results of the literature search and our search in GitHub. Out of the 10 mutation tools with more stars in GitHub, 7 were not identified in our literature search, namely: StrykerJS (https://stryker-mutator.io/docs/stryker-js/introduction) (JavaScript), Infection (https://infection.github.io/) (PHP), Humbug (https://github.com/humbug/humbug) (PHP), Stryker.NET (https://stryker-mutator.io/docs/stryker-net/Introduction) (.Net), Cosmic-Ray (https://cosmic-ray.readthedocs.io/en/latest/index.html) (Python), Go-Mutesting (https://github.com/zimmski/go-mutesting) (Go) and Mutmut (https://github.com/boxed/mutmut) (Python). Interestingly, none of those seven tools target Java or C/C++ programs, despite those being the languages receiving more attention in research papers. This suggests a gap between the state of practice and research. We hypothesize that the tools born outside the academic environment are focused from their conception on usability, that is, they seek to be mainly useful for the community. Within academia, however, efforts tend to be more dedicated to research, where priority is given to



(a) Number of mutation testing tools released per year



(b) Cumulative number of mutation testing tools released per year

Fig. 2 Mutation testing tools released between January 2008 and May 2021

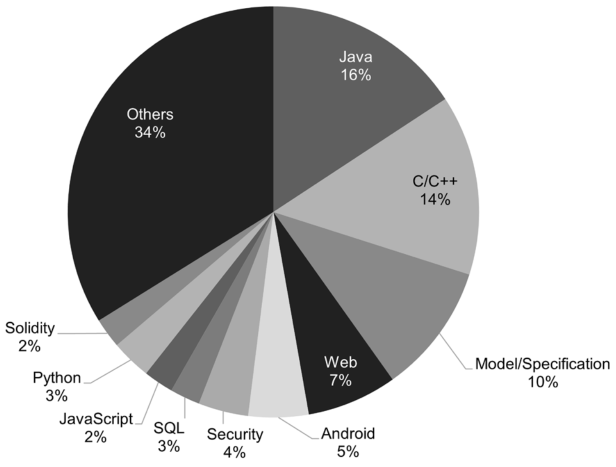


Fig. 3 Classification of mutation testing tools by target artifact

the evaluation of experimental features (e.g., new mutation operators) and its dissemination through research papers.

Table 2 shows the top 10 most popular mutation testing tools based on the number of GitHub repositories including evidence of their use. As explained in Section 3, in what follows we will focus on these 10 tools to make our work affordable. For each tool, the table shows its name, references (to the main repository in GitHub and the original research paper, if any), programming language, and standard statistics of the tool repository in GitHub (October 2021), namely: number of watchers (i.e, followers), number of stars, number of forks, number of commits, number of issues, and date of the first and the last commit. Note that we did not find a main GitHub repository for Major and therefore the corresponding statistics are missing. Also, the repository hosting MutPy was created before the corresponding mutation tool was released (in 2014), while the repository hosting MuJava was created well after the first version of the corresponding tool was released (in 2004).

As illustrated in Table 2, the set of tools most widely used based on the traces found in GitHub addresses six different programming languages (Java, JavaScript, PHP, Python, Ruby, and .NET), which shows that the impact of mutation testing is not related to specific domains. Specifically, Java is the dominant target language (Major, MuJava, and PIT), followed by PHP (Humbug and Infection) and Python (Mutmut and MutPy). The most popular tools in terms of GitHub watchers, stars and forks are Humbug (PHP), StrykerJS (JavaScript) and PIT (Java), respectively. Mutant and StrykerJS are the most active projects in terms of commits (3,612) and issues (868), respectively. Being this list of the most popular tools —based on the evidence of use found in GitHub—, it is notable that four of them were conceived in 2016 (Mutmut and StrykerJS) or later (Infection, 2017, and Stryker.NET, 2018). Most of the tools hosted in GitHub are active, having at least one commit in the last 12 months. The exceptions are MuJava (last updated in 2016), Humbug (2017)—its main repository has been archived by the owner— and MutPy (2019).

Table 2 Mutation testing tools under study: W (watchers), S (stars), F (forks), C (commits), I (issues), FC (first commit) and LC (last commit)

Name & Refs	Language	Tool main repository in GitHub						
		W	S	F	C	I	FC	LC
Humbug (https://github.com/humbug/humbug)	PHP	58	1,154	75	733	134	2015	2017
Infection (https://infection.github.io/)	PHP	33	1,538	129	1,290	421	2017	2021
Major (Just 2014)	Java	–	–	–	–	–	–	–
MuJava (https://github.com/jeffoffutt/muJava; Ma et al. 2005)	Java	9	57	38	19	18	2015	2016
Mutant (https://github.com/mbj/mutant)	Ruby	36	1,782	141	3,612	489	2012	2021
Mutmut (https://github.com/boxed/mutmut)	Python	10	530	58	429	150	2016	2021
MutPy (https://github.com/mutpy/mutpy; Derezińska and Hałas 2014)	Python	10	267	32	327	33	2011	2019
PIT (https://github.com/hcoles/pitest; Coles et al. 2016)	Java	57	1,295	297	1,908	620	2010	2021
StrykerJS (https://stryker-mutator.io/docs/stryker-js/introduction)	JavaScript	28	2,004	182	2,817	868	2016	2021
Stryker.NET (https://stryker-mutator.io/docs/stryker-net/Introduction)	.NET	24	618	115	838	525	2018	2021

Summary of answers to **RQ1**:

1. Tool support for mutation testing is rich with at least 127 mutation tools created between 2001 and 2020.
2. Over one third of the mutation tools found in the literature has no name, which suggests that they are research prototypes.
3. The trend of creation of new mutation tools shows an increasingly steep curve with a boost in interest starting in 2008. 2018-2019 were the most prolific years with 13 tools released per year.
4. Java and C/C++ are the dominant target programming languages of existing mutation tools. However, about two thirds of the tools found address other types of artifacts and programming languages.
5. In recent years, the interest in languages like Java and C/C++ seems to diminish in favor of trending languages such as JavaScript and Solidity.
6. There seems to be a gap between the state of research and practice with seven of the top ten most popular tools in GitHub (based on the number of stars) not being reported in the research literature.
7. The top ten most popular mutation tools, based on our search, target 6 different programming languages, which suggests that the impact of mutation testing transcends specific domains. Among them, Java (3 mutation tools), Python (2) and PHP (2) appear as the predominant programming languages.
8. The ranking of the top ten tools is mainly topped by active and recently created mutation tools. Notably, the main repository of 6 of them has at least one commit in 2021 and four of these repositories were created within the last five years (between 2016 and 2018).
9. Most of the repositories hosting the top ten mutation tools have attracted the attention of the GitHub community. Humbug, Infection, Mutant, PIT and StrykerJS count with more than 1K stars. They are also quite active in terms of commits and issues.

4.2 Adoption of Mutation Testing

This section addresses RQ2 by studying the number and the general characteristics of the 3,581 GitHub repositories including evidence of use of the ten mutation testing tools under study, i.e., those found in a larger number of repositories. Specifically, all these tools were used through a library import, and thus in what follows we will refer to repositories *importing* the mutation testing tools.

For each mutation tool, Table 1 shows its name, the search queries used, the total number of repositories found, and the total number of active repositories. Recall that, to make the manual revision of repositories affordable and identify more accurately the use of mutation testing tools at present, our study is based on the analysis of active repositories, i.e., those having a commit in the last 12 months. Regarding the number of active repositories, Infection is by far the most popular tool being imported in 1,213 repositories; followed by PIT (816), Humbug (636), StrykerJS (487) and Mutant (296). At the other extreme are Mutmut (54), Stryker.NET (36), MutPy (24), Major (10) and MuJava (9).

Figure 4 depicts the number of repositories importing each mutation tool created per year in the last decade. Notice that the corresponding mutation tool may have been imported in a

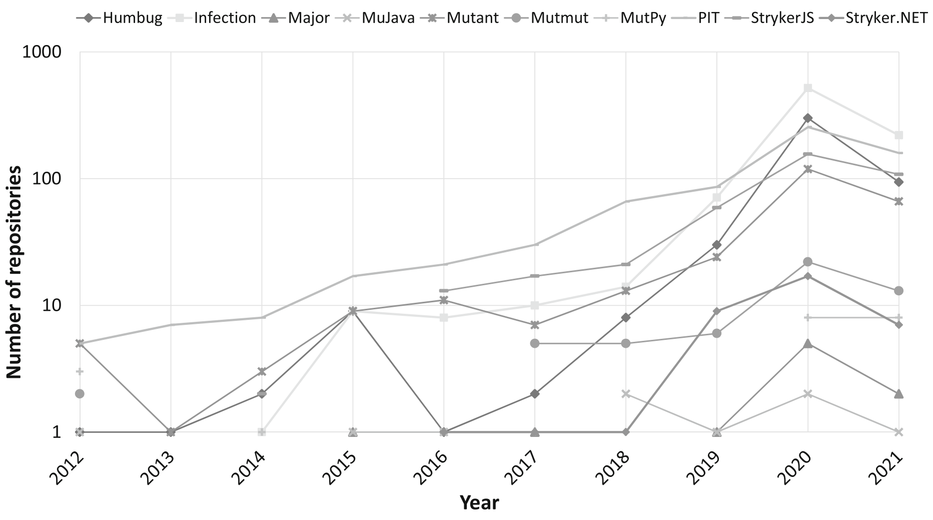


Fig. 4 Number of new repositories per mutation tool and year (y-axis on logarithmic scale)

later commit and not necessarily when the repository was created. For instance, some repositories importing Infection were created before 2017, the year when the first commit for this tool was made (according to Table 2). The number of new repositories importing Infection, Humbug, PIT, StrykerJS, Mutant and, on a smaller scale, Mutmut and Stryker.NET, has notably increased in the period 2018-2020. Indeed, all of them reach their peak in 2020, but we should interpret this information cautiously; unlike the rest of the years, part of the repositories in 2020 may appear as active simply because they were created that year. Also, note that we performed our search in June 2021, so the graph only reflects the repositories created during the first half of this year (that explains the decrease in the number of repositories from 2020 to 2021 in almost all the tools). The case of Infection is especially remarkable, with an increment in the number of repositories importing it from 10 in 2017 to 519 in 2020. It is also noteworthy the case of PIT, which shows a positive upward tendency since 2012, remaining as the most imported tool during the last decade until 2020, when it was surpassed by Infection and Humbug. We think this is due to the good support and documentation of PIT and the fact that the tool is popular inside and outside academia. Conversely, we cannot appreciate any clear increase trend for the rest of the tools. For instance, we found a single repository importing MutPy in the years 2014, 2016, 2017 and 2018. Similarly, the creation of the nine repositories integrating MuJava is well spread between 2012 and 2021, with a maximum of two repositories in 2018 and 2020. It is significant that MuJava, a tool widely known in the scientific community, does not present a greater number of active repositories using it. This may be explained by the fact that the tool has not been updated since 2016 —the date of its last commit.

As previously mentioned, the mutation tools under study were used through a library import in all the repositories found. Specifically, some applications use a direct import, this is, they include the mutation library directly (e.g., Jar files in Java or Phar files in PHP), whereas others resort to automated dependency management using build automation tools like Gradle, Maven or Tox. We did not observe any correlation between the import method and the adoption of the mutation tools. Instead, import methods vary among tools and languages. For example, direct import through a Jar file is the method offered by MuJava

(Java), and tools like Mutant (Ruby), StrykerJS (JavaScript) and Stryker.NET (.Net) offer build automation tools as the main way of import. Some other tools for Java (e.g., PIT), PHP (e.g., Infection) and Python (e.g., MutPy) provide both methods of import, but build automation tools is the clear choice when both ways are available –according to the number of traces found of each type when executing the search strings–. Overall, the majority of projects using mutation testing tools prefer build automation tools as means of import.

Summary of answers to **RQ2**:

1. The number of repositories importing any of the 10 mutation tools under study is relevant, with at least 3,581 active repositories detected and three of the tools —Infection, PIT and Humbug— being imported in more than 500 GitHub projects.
2. The most popular mutation tool by far is Infection, being imported in 1,213 repositories out of 3,581.
3. The repositories importing Humbug, Infection, Mutmut, StrykerJS and Stryker.NET represent 67.7% of the whole set of repositories. It is noteworthy that these five tools were identified in our GitHub search and not in the literature.
4. The number of projects importing Infection, Humbug, PIT, StrykerJS and Mutant has substantially increased in recent years. Such trend is not observed in the tools MutPy, Major and MuJava.
5. PIT has been the most imported tool in the last decade, only surpassed by Infection and Humbug in 2020.
6. Import methods vary among tools and programming languages, being build automation tools the most common mechanism of integration.

4.3 Classification of Repositories

In this section, we address RQ3 by studying the types of repositories importing mutation testing tools. Specifically, we classified the GitHub projects into five main groups according to their purpose: research, teaching, development, learning and extension. We classified as *research* those repositories including evidences of the use of mutation in research activities, for example, replication packages associated with research papers[3]. *Teaching* projects are those using mutation testing for teaching, including university courses[4], tutorials, books, or programming katas. We classified as *development* those repositories including evidence of the use of mutation testing for actual software development[5]. We identified a good number of personal repositories including toy examples used to learn how mutation and other technologies work, we classified those as *learning* projects[6]. We also found a few projects that were extensions of the mutation testing tools under study. We marked them as *extension* projects[7]. To categorize the repositories, we manually examined all the information

[3]*Research* project: https://github.com/DPerf-Github/DPerf
[4]*Teaching* project: https://github.com/andrewt0301/qa-testing-course
[5]*Development* project: https://github.com/ecphp/php-directive-bundle
[6]*Learning* project: https://github.com/MartinThoma/algorithms
[7]*Extension* project: https://github.com/saiema/MuJava

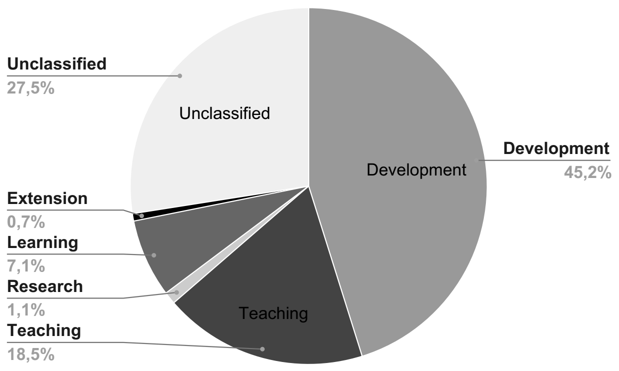


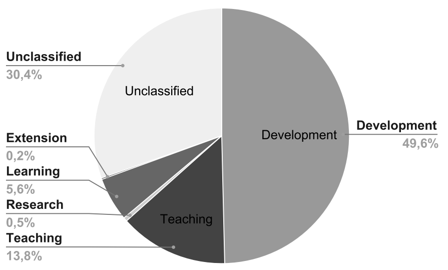
Fig. 5 Classification of repositories by category (all mutation testing tools under study)

available in the corresponding repositories, including their name, description, available documentation, and, exceptionally, even the source code. Despite our best efforts, there were cases where we were unable to identify the category a repository belongs to, for example, in repositories with no documentation nor descriptive names or description. We labeled these as *unclassified*[8].

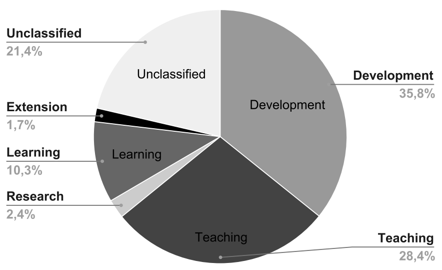
Figure 5 shows the percentage of projects belonging to each category. As illustrated, almost half of the repositories under study, 45.2% (1,617 out of 3,581) are dedicated to development, 18.5% (662) to teaching, 7.1% (254) to learning, 1.1% (40) to research, and 0.7% (24) to extensions of the mutation tools under study. *Unclassified* repositories represent 27.5% of the total of repositories. However, when analyzing the data, we observed some significant differences among the mutation tools found in our GitHub search (Humbug, Infection, Mutmut, StrykerJS and Stryker.NET) and those found in the literature (Major, MuJava, Mutant, MutPy and PIT). To investigate this, we performed a similar classification of the repositories importing both sets of tools, shown in Fig. 6a and b, respectively. Interestingly, we observe an increase in the percentage of projects dedicated to *development* and a decrease in the number of *teaching* repositories when focusing on tools found only in GitHub. On the contrary, mutation tools found in the literature (Fig. 6b) show the opposite behavior, with a considerable reduction in the percentage of repositories dedicated to *development* (35.8% vs 49.6%) and an increase up to double in the number of *teaching* repositories (28.4% vs 13.8%), *research* repositories (2.4% vs 0.5%) and *learning* repositories (10.3% vs 5.6%). Therefore, it seems that a higher proportion of the projects using tools from GitHub –presumably from outside the scientific world– have a developmental purpose, possibly because they have attracted more interest from non-academic developers. On the other hand, the tools found in the literature seem to be used in almost equal proportion in both *development* and *teaching* projects, and the sum of repositories from the categories *teaching*, *learning* and *research* is higher than the number of repositories dedicated to *development*. We hypothesize that tools found in the literature, possibly related to a scientific origin, have drawn more interest from the research and teaching community.

Figure 7 shows the number of repositories per tool involved in each category on a logarithmic scale. *Development* is the dominant category in the repositories importing the tools

[8] *Unclassified* project: https://github.com/douniaharag/GSB-Frais



(a) Repositories importing the mutation tools found in GitHub



(b) Repositories importing the mutation tools found in the literature

Fig. 6 Classification of repositories by category, divided by tools found in GitHub and in the literature

Humbug, Infection, Mutmut, MutPy, PIT and Stryker.NET. *Teaching* seems to be the most common use of the tools Major, Mutant and StrykerJS. MuJava seems to be mostly used in *research* projects. It is noteworthy that only three tools, Infection, PIT and StrykerJS, are imported in projects from all categories.

To investigate the origin of the repositories using mutation testing tools, we manually analyzed the information related to the ownership of the repositories. We labeled repositories as *Industry* if the owner's affiliation is related to the industry, either as an employee or freelance, *Academia* if the owner's affiliation is linked to a teaching center, and as *Public* when the owner is associated with a public institution, a non-profit organization or a known open-source community. Unfortunately, only 31.1% of the repositories (1,112 out of 3,581) included ownership information and therefore the results should be taken cautiously. The bulk of classified repositories sourced from *industry* (20.9%, 747 out of 3,581) with owners working in private companies[9]; followed by *academia* (6.8%, 242), like those owned by University professors[10] or students[11]; and a minority from *public* institutions, non-profit organizations or open-source communities (3.4%, 123). As examples, we found some popular repositories using Infection owned by the open-source community phpMyAdmin[12], a set of open-source projects developed by the European Commission[13], and the not-for-profit UK Centre for Ecology & Hydrology[14]. Interestingly, we also observed differences when analyzing the origin of repositories associated with mutation tools found in GitHub and tools found in the literature. The former ones come mostly from *industry*, 21.8% out of 31.1% of classified repositories, with a small percentage reserved for *academia* (4.3%) and *public* institutions (1.7%), whereas the latter are equally sourced from *industry* (19%) and from *academia* (11.9%) plus *public* institutions (7.2%). These results show that the uses of the mutation tools found in GitHub seem to be closely related to *development* projects and *industry* owners, while those tools found in the literature present more variety of uses and origins.

[9] *Industrial* owner: https://github.com/gwillem/flarum-multitenant
[10] *Academic* owner: https://github.com/andrewt0301/qa-testing-course
[11] *Academic* owner: https://github.com/enginyenice/Object-Detection-With-Smart-Phones
[12] *Public* institution: https://github.com/phpmyadmin/phpmyadmin
[13] *Public* institution: https://github.com/ecphp/php-directive-bundle
[14] *Public* institution: https://github.com/NERC-CEH

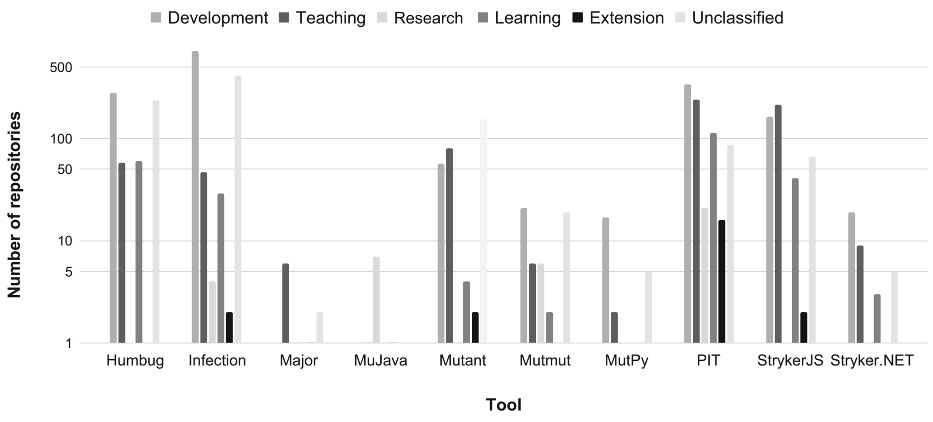


Fig. 7 Tools' repositories by category (y-axis on logarithmic scale)

Summary of answers to **RQ3**:

1. The predominant use of the repositories under study is development (45.2%), followed by teaching (18.5%), learning (7.1%) and research projects (1.1%). Extensions of the mutation tools under study were scarce (0.7%). About 27% of the repositories remained unclassified because of the lack of information.
2. Six out of the ten tools under study are mostly used for development purposes (Humbug, Infection, Mutmut, MutPy, PIT and Stryker.NET), whereas three of them are predominantly used for teaching (Major, Mutant and StrykerJS). MuJava is mostly used for research and PIT is the tool with more repositories dedicated to research (21).
3. Based on the owners' information, 20.9% of the repositories were linked to industry, 6.8% related to academia, and 3.4% to public institutions. Only one third of the repositories had this information available.
4. We found differences between the purpose and origin of the classified repositories that import mutation tools found in GitHub and those found in the literature. The former ones are mostly linked to development and mainly have their origin in the industry; the latter, more frequently linked to academia than the others, have a greater presence in the set of projects related to teaching, learning and research.

4.4 Repository Activity and Relevance

This section seeks to answer RQ4 by studying the activity and relevance of the repositories importing the mutation testing tools under study. To do this, we analyzed the values of different measures extracted from GitHub related to activity (commits of the repositories) and popularity (contributors, watchers, stars and forks), and calculated several statistics by mutation tool. This allows us to observe the tools attracting more attention from highly active and popular projects.

Figure 8 displays quartiles, median, minimum and maximum values of commits submitted to repositories importing each mutation tool. The highest median of commits is found

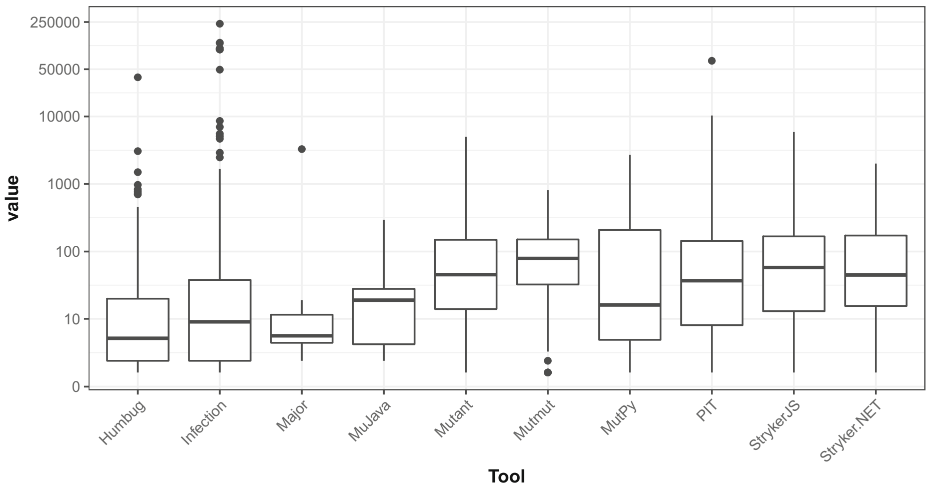
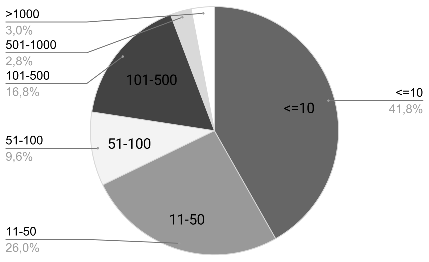


Fig. 8 Commits per mutation tool (*y-axis* on logarithmic scale)

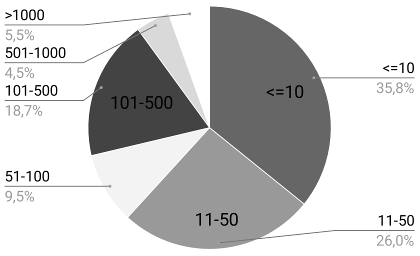
in Mutmut, with 79 commits. However, Mutmut and also MuJava are the only tools not referenced in at least one project with more than 1K commits. In a second group, Mutant, PIT, StrykerJS and Stryker.NET present a similar distribution, with medians in the range 45-60 commits. It is surprising, however, the low median of commits in repositories importing Infection and Humbug (9 and 5, respectively), being these two of the most used tools. This might be due to the boost in the number of repositories created in 2021 that use these two tools in comparison to other tools (see Fig. 4) —that is, the period of activity of those repositories is shorter—. It might also suggest that many of their repositories will not be active the next year. Regardless of its low median, Infection appears in a good number of projects with a high volume of commits —shown as outliers—. Namely, 16 out of the 19 projects with more than 10,000 commits reference Infection. This includes the project with the maximum number of commits observed overall (235,708)[15] —related to the procurement of masks in Taiwan—, followed by two projects belonging to phpMyAdmin —*composer*[16] and phpMyAdmin's main repository (see footnote 12)— with 123,474 and 121,828 commits, respectively.

Figure 9 shows the distribution of repositories based on the total number of commits overall, and per category. As illustrated in Fig. 9a, 41.8% of the repositories (1,488 out of 3,581) received 10 or fewer commits, and 26% (927 out of 3,581) received between 11 and 50 commits. This reflects a limited activity in the projects importing mutation tools. This low activity was especially observed in unclassified (Fig 9g) and learning projects (Fig 9f) where 81.1% and 78% of the projects, respectively, received 50 commits or fewer. As observed in our manual revision, a large portion of these projects are often small and poorly documented examples or proofs of concepts with limited activity. On the contrary, projects extending the mutation tools under study (Fig. 9e) are more active with 50% of the projects having received 51 commits or more. In the middle, we find teaching (Fig 9c), research (Fig 9d) and development projects (Fig 9b) with 40.4%, 40% and 38.2% of the projects, respectively, having received 51 commits or more. Finally, 79% of the projects

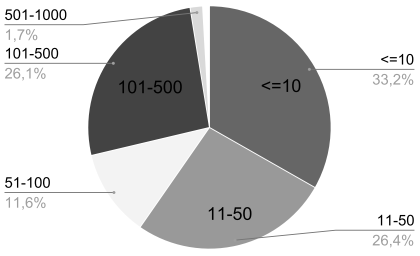
[15]https://github.com/kiang/pharmacies
[16]https://github.com/phpmyadmin/composer



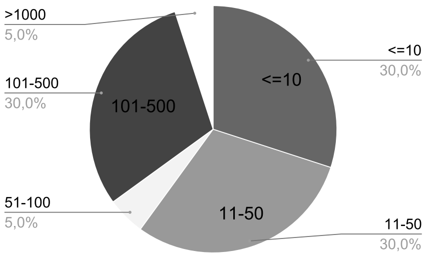
(a) Commits in all projects



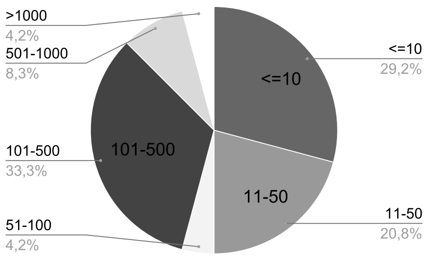
(b) Commits in Development projects



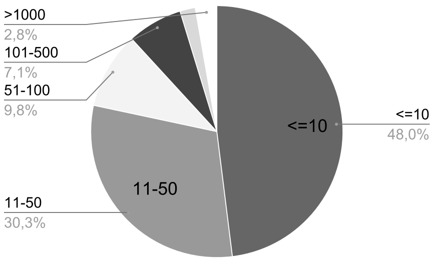
(c) Commits in Teaching projects



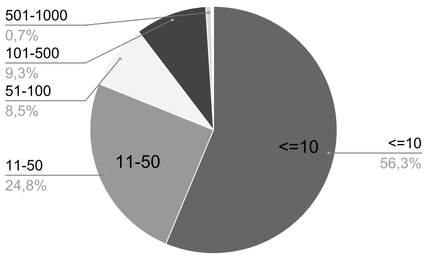
(d) Commits in Research projects



(e) Commits in Extension projects



(f) Commits in Learning projects



(g) Commits in Unclassified projects

Fig. 9 Distribution of commits

with more than 500 commits (162 out of 205) fall into the category *development*, being PIT, Infection and StrykerJS the main mutation testing tools used by these repositories. Teaching projects occupy the second position in this ranking with 17 repositories (8%) with more than

500 commits, leading the list of imports PIT, StrykerJS and Mutant. These results reflect that the most active projects are those dedicated to development, regardless of the mutation tool used.

Figure 10 shows several boxplots with the distribution of different popularity-related metrics of the GitHub projects importing the mutation testing tools under study. Specifically, the graphs show on a logarithmic scale the distribution of contributors, watchers, stars, and forks, including their quartiles, median value, minimum, maximum, and outliers. As illustrated, distributions are generally low, with noticeable exceptions in the forms of outliers, especially in Infection, Mutant, PIT and StrykerJS. Thus, to simplify the analysis

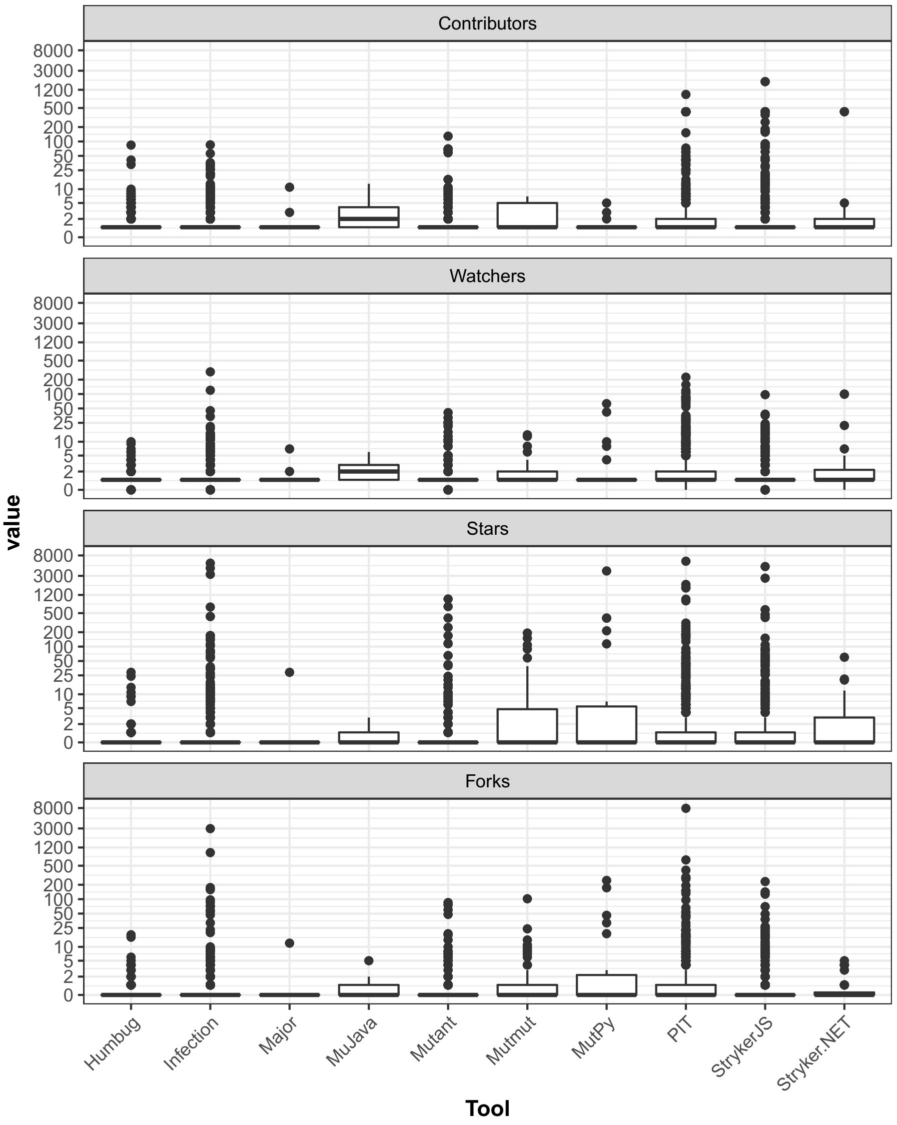


Fig. 10 Contributors, watchers, stars and forks (*y-axis* on logarithmic scale)

of the results, outliers over 50 in contributors, watchers, stars, or forks will be deemed as highly popular or relevant projects.

Regarding contributors, all the mutation tools are imported in repositories with 10 or fewer contributors —excluding outliers—, with MuJava being the only tool with a median value over one (2). This clearly shows that most of these repositories were generated with individualistic purposes (e.g., to perform a lab activity, to develop basic or ad hoc applications, or to store personal resources), while only a few of them stem from the joint collaboration of a large group of developers in a community, public entity or company. Regarding the latter, interestingly, 74 repositories integrating PIT count with 418 contributors, all from the same owner (HM Courts & Tribunals Service, UK government). Apart from those, half of the repositories with over 50 contributors import StrykerJS (18 out of 36). Most repositories have 5 watchers or fewer, again with MuJava having the highest median value (2). Among the set of repositories that are watched by 50 or more GitHub users, PIT is the tool with more associated repositories (17 out of 23). With regard to stars and forks, the median value is 0 for all the tools. Altogether, the low number of watchers, stars and forks in general, reflects that most of these repositories contain toy, small-sized or still incomplete projects that are not meant to be distributed globally. Contrarily, some of them may have not received much attention from the community despite that being the original intention (as it can be grasped from the details of installation and use in some of the readme files). Using 50 as a threshold, PIT (26 repositories), StrykerJS (23) and Infection (20) are the tools more frequently integrated into the most starred repositories. Similarly, PIT (14 repositories), Infection (10) and StrykerJS (4) are the tools more commonly imported in highly forked repositories. Among the most relevant projects based on these metrics, we can cite the main repository of Checkstyle[17] (using PIT), with 6,072 stars and 7,894 forks, and phpMyAdmin (using Infection), with 291 watchers.

Summary of answers to **RQ4**:

1. Two thirds of the repositories importing mutation testing tools received 50 commits or fewer, which reflects a limited overall activity.
2. Most of the projects with more than 500 commits are classified as development (162 out of 205, 79%). PIT, Infection and StrykerJS are the main mutation testing tools used by these repositories.
3. Repositories classified as extensions are by far the most active ones. Learning projects appear as the least active.
4. Commits to repositories importing Mutmut, StrykerJS, Mutant, Stryker.NET and PIT are quite more numerous than to those referencing Humbug or Infection. However, Infection is currently in use in particularly active projects.
5. Judging by the median of contributors (1), watchers (1), stars (0) and forks (0), we can say that most of these repositories have limited popularity. However, we did find a number of highly popular repositories with at least 50 contributors (110 repositories), 50 watchers (23), 50 stars (88) or 50 forks (34).
6. PIT and, to a lesser extent, Infection and StrykerJS, are the most commonly imported tools in that set of highly popular repositories in terms of watchers, stars and forks. StrykerJS is, however, the most frequently integrated tool in repositories developed by groups of 50 or more contributors.

[17] https://github.com/checkstyle/checkstyle

5 Action Points

The results of our study open new promising research directions. Among others, we identify the following action points to complement our findings and delve into their causes.

1. *Further investigation of mutation tools*. Our results show that some mutation testing tools are clearly more popular than others and that the popularity of a tool in the research arena and in practice does not usually go hand in hand. The results of our study, and in particular our dataset, may be an excellent starting point for investigating the causes behind these findings, for example, by conducting a survey among the developers of the most popular projects using each mutation tool.
2. *Understanding the use of mutation testing in practice*. Our study investigates what mutation testing tools are used in practice, but not *how*. Such investigation would be of great interest to better understand the strengths and drawbacks of the technique in practice. For example, are all mutation operators equally used? is mutating testing used during the whole development process or only at certain phases? is it integrated into CI/CD pipelines? Answering these and other questions would certainly require interacting with developers and, again, our dataset could act as a useful source to identify promising subject projects.
3. *Collecting the opinions of users*. Related to the previous point, it would be very helpful to collect the opinions of the actual users of the technique regarding its benefits and limitations. Such insights would be of interest to foster the development of tools and techniques driven by the actual needs of practitioners. Again, our work could be a good starting point to identify popular and mature projects where the different mutation testing tools are currently in use.
4. *Comparison of mutation testing tools*. We identified some of the most popular mutation tools in practice, including their target artifacts and programming languages, but we did not perform a rigorous comparison of their main features. Such comparison would be of great interest to better understand the potential of current tool support and, perhaps, to explain why some tools are more popular than others. Also, such comparison could serve as the starting point for the definition of guidelines for the development of better mutation testing tools.
5. *Conducting further repository mining studies*. The results of our study show the potential of repository mining to better understand the current state of practice on mutation testing, but it only scratches the surface. Hence, for example, our work could be complemented with similar studies in other platforms such as Bitbucket or GitLab. Also, it would be nice to increase the degree of automation of the analysis of the repositories. Ideally, this would serve as an effective mechanism to monitor the use of mutation testing in practice. Last, but not least, similar studies could be conducted with different objectives such as understanding the evolution of projects using mutation testing or the performance of the technique when used at scale.

6 Related Work

In this section, we discuss the related work on mutation testing and repository mining.

6.1 Mutation Testing

Mutation testing has been extensively analyzed and studied in the literature. Several systematic literature reviews (Jia and Harman 2011; Offutt and Untch 2001; Papadakis et al. 2019) addressed mutation testing from a general point of view or focused on more particular issues within the topic, such as the techniques to reduce its cost (Pizzoleto et al. 2019; Usaola and Mateo 2010). Some other studies examine various mutation testing tools and compare them following a variety of perspectives. A recent work assesses the effectiveness of the three mutation tools for Java (PIT, MuJava and Major) in the detection of faults (Kintis et al. 2018). The experiments conducted demonstrate that, while none of these mutation tools completely subsumes the others, an improved version of PIT with research purposes is the more effective tool at inducing test cases that could reveal real faults. Another group of papers deepens on technical aspects regarding the usage of mutation testing tools, such as the efficiency, controllability or the compatibility and integration with a test environment. Although most of these studies focus on Java tools (Delahaye and du Bousquet 2015; Márki and Lindström 2017), some other programming languages have been considered too, like C# (Uzunbayir and Kurtel 2019). The study by Delahaye and du Bousquet (2015) identifies three different profiles that can influence the election of a mutation tool: teaching, research and industry. They conclude that PIT is a good choice for the industry and the teaching profile, where tools should be easy to apply and, in the particular case of the industry, should have a good balance between efficiency and meaningfulness of results. According to the survey by Papadakis et al. (2019), PIT, MuJava and Major for Java, and Proteum for C are the mutation tools more frequently used in experimental studies. Some of these results are reflected in our study mining GitHub. MuJava mostly appears in research projects and PIT is mainly imported in development and teaching repositories, but it is also the tool with more repositories dedicated to research.

As for the application of mutation tools in the development of software projects, some works have shown the possible benefits of transferring mutation testing concepts from academia to industry, carrying out empirical studies with open-source applications (Just et al. 2014) and industrial projects (Delgado-Pérez et al. 2018). In fact, a recent study by Petrović et al. (2021a) reveals that mutation testing has positive long-term effects on the testing practices of developers. Also, some recent studies analyze the use of the technique in large companies. Google (Petrovic et al. 2021b) implements its own mutation system for seven programming languages and applies a diff-based probabilistic approach to reduce the prohibitive computational expenses of traditional mutation analysis. In the study by Beller et al. (2021), more than half of the mutants generated –based on some error-inducing patterns– were not detected by Facebook's rigorous tests.

This study complements previous work by providing a detailed and updated picture of the state of mutation testing in practice by looking into GitHub. Rather than focusing on specific companies or tools, we provide a general overview of the use of the technique in the wild providing helpful insights and trends that will hopefully contribute to narrow the gap between research and practice.

6.2 Repository Mining

Data mining is defined as the process of exploring and analyzing large data sets (e.g., from databases or information repositories) in order to discover meaningful patterns and rules (Aggarwal and Zhai 2012; Pujari 2001; Witten et al. 2017). Data mining has become a highly-demanded task with wide applications, which has attracted many researchers and developers. Among the most commonly mined data sources in recent times, we find GitHub, which is the platform of reference when mining open-source repositories to learn from past experiences (Borges et al. 2016; Cosentino et al. 2016; Gonzalez et al. 2020; Gousios and Spinellis 2017; Kalliamvakou et al. 2014; Xiong et al. 2017).

We found some recent studies on mining data from repositories in GitHub (Borges et al. 2016; Kalliamvakou et al. 2014). Borges et al. (2016) proposed a study on the popularity of software systems hosted at GitHub. This study reveals that repositories owned by organizations are more popular than those owned by individuals. They also reported a strong correlation between stars and forks, and the importance of a large number of contributors to the success of open-source software. In the experience by Kalliamvakou et al. (Kalliamvakou et al. 2014), the best way to identify the software projects with more activity is to consider those projects that have a good balance of number of commits and pull requests, and have a number of committers and authors larger than 2. The number of issues can also be used as an indicator, but not all active projects use GitHub's issue tracker.

There exist different alternatives to retrieve automatically information from GitHub. In (Mombach1 and Valente 2018), the authors made a comparative study among the three most used methods to mining data from GitHub up to 2018: GitHub REST API, GHTorrent and GitHub Archive. They concluded that REST API was more appropriate when it is important to retrieve the most recent data up to date. GitHub Archive collected data since 2011 and GHTorrent from 2012 to June 2019, when part of its functionality was no longer supported (Cosentino et al. 2016; Gousios 2013; Kalliamvakou et al. 2014; Mombach1 and Valente 2018). Recently, Brito et al. (Brito and Valente 2020) published an experiment comparing REST API vs GraphQL, the most popular tools for mining GitHub nowadays. The authors found that GraphQL requires less effort to implement API queries than REST API, and it provides GraphiQL, a web app to test GraphQL queries with autocomplete options.

Compared to previous work, in this paper we provide a novel application of repository mining in GitHub: studying the use of mutation testing in practice.

7 Threats to Validity

The factors that could have influenced our study are summarized in the following internal and external validity threats.

Internal validity. This refers to whether there is sufficient evidence to support the conclusions and the sources of bias that could compromise those conclusions. Our work is based on the assumption that software projects importing a mutation testing tool (e.g., library) most likely use or have used that tool at some point, although this might not be always true. This threat is minimized by the number of manually reviewed repositories, over 3.5K, which dilutes the potential effect of these unlikely cases where the mutation tool is imported, but not used.

The data collection method could also threaten the validity of our work. To mitigate this threat, we resorted to automated methods whenever it was possible, including the use of web scraping and automated queries on the GraphQL API of GitHub. We managed to define search strings for 55 out of the 83 mutation tools with a name (this could not be done for some tools as we did not find a website that provided the required information). Despite our best efforts analyzing individually each tool, we may have missed some search strings. However, the fact that we found more than 6K repositories with traces of 44 different tools makes us confident that the procedure followed has been adequate. We manually checked the validity of the data at each step, for example, by randomly comparing the data returned by the scrapper and the web API with those observed in the GitHub web platform.

The manual review and classification of the repositories could also threaten the validity of the results, since some repositories could have been misclassified. To minimize this threat, the four authors of the paper participated actively in the review process following a common review procedure. We found some repositories that were hard to classify with confidence. For instance, sometimes it was difficult to discern whether a repository with some toy examples had been created for a teaching/tutorial session or by a novice programmer that sought to learn a particular technology. These repositories were reviewed by all the authors in several working sessions until reaching a consensus. Regarding the classification of repositories into industry, academia or public institution, we based on the profile of the repository's owner; however, whether this information was conveniently updated by the owners on the platform (e.g., when changing from academia to industry or vice-versa) cannot be ascertained. Similarly, we cannot determine whether a project was developed in a moment when the owner had a different role than the one currently stated in the profile. To mitigate this threat, most of the analyses were performed manually checking all the information available in the corresponding repositories, including the links to external resources and, exceptionally, even the source code. This threat is also partially mitigated by the number of reviewed repositories, over 3.5K, which minimizes the effect of possible misclassified repositories.

Finally, our work is focused on the study of public repositories only, but surely there exists private repositories importing the mutation testing tools under study. Unfortunately, the analysis of such repositories is beyond our reach.

External validity. This refers to the generalizability of the conclusions. Widening the scope of our work to other platforms beyond GitHub (e.g., Bitbucket (https://bitbucket.org/)) could have yielded different results. However, the size and the popularity of GitHub in related mining studies make us confident in the validity of the reported trends.

Also, our study focuses on mutation tools leaving traces of their use in GitHub, e.g., libraries. However, there exist other ways of using mutation tools that do not usually leave traces in code repositories like, for example, those released as an IDE plugin, as an executable file, or as a docker image. A more comprehensive analysis of such uses would require other experimental means (e.g., surveys) and is beyond the scope of this article.

Finally, we focused on active projects (i.e., those with at least one commit in the last 12 months) due to the search restrictions of GitHub and to make our work affordable. This also allowed us to study the current use of mutation testing in practice. Removing this restriction, however, would certainly throw different results.

8 Conclusions

In this paper, we report the findings of a study on the use of mutation testing in practice by looking into GitHub. Specifically, we systematically searched for GitHub repositories including traces of the use of 127 different mutation testing tools released in the last two decades. Then, we focused on the top ten more widely used tools and manually revised the active repositories importing them, over 3.5K. The results show a notable upturn in interest and activity in recent years, mostly focused on a small set of highly popular tools. The impact of the technique transcends specific programming languages with the ten most widely used tools targeting Java, JavaScript, PHP, Python, Ruby, and .Net. The predominant use of mutation testing is development, followed by teaching and learning, and research. We found notable users of the technique including public institutions and relevant open-source projects. Interestingly, some of the most widely used tools in GitHub are rarely found in research papers, this is the case of Infection and Humbug for PHP, and StrykerJS for JavaScript. Our results open new promising research directions including new empirical studies, both quantitative and qualitative, that complement our finding and delve into their causes. Overall, our work provides a novel perspective on the use of mutation testing in the wild and show the potential of repository mining studies to close the gap between research and practice.

Appendix: Mutation Tools

Table 3 List of mutation tools. References to the tools with source "Papadakis et al." can be found in the review by Papadakis et al. (2019); "Literature" refers to tools found between 2018 and 2020 in our review of the literature; Tools labeled as "GitHub" were found in our GitHub search of tools with more stars

Source	Name	Year	Domain
Papadakis et al.	mutate	N/A	C/C++
Papadakis et al.	Jester	2001	Java
Papadakis et al.	Proteum	2001	C/C++
Papadakis et al.	mutgen	2003	C/C++
Papadakis et al.	MuJava	2004	Java
Papadakis et al.	ByteME	2006	Java
Papadakis et al.	SQLMutation	2006	SQL
Papadakis et al.	Jumble	2007	Java
Papadakis et al.	ESTP	2008	C/C++
Papadakis et al.	Not Named	2008	Others programming languages (Sulu)
Papadakis et al.	Milu	2008	C/C++
Papadakis et al.	Not Named	2008	Model/Specification (NuSMV models)
Papadakis et al.	Not Named	2008	Simulink
Papadakis et al.	Not Named	2008	Security (Security policies)
Papadakis et al.	Not Named	2008	Model/Specification (LOTOS)
Papadakis et al.	Not Named	2008	AspectJ
Papadakis et al.	Javalanche	2009	Java
Papadakis et al.	JDama	2009	SQL
Papadakis et al.	AjMutator	2009	AspectJ
Papadakis et al.	GAmera	2009	Web (WS-BPEL)
Papadakis et al.	Not Named	2009	Others (Boolean logic)
Papadakis et al.	PASTE	2009	Model/Specification (TFSM)
Papadakis et al.	Not Named	2009	Model/Specification (Z)
Papadakis et al.	Not Named	2009	Others (GCC-XML)
Papadakis et al.	Not Named	2009	Others programming languages (Lustre)
Papadakis et al.	Not Named	2009	Java
Papadakis et al.	PIT	2010	Java

Table 3 (continued)

Source	Name	Year	Domain
Papadakis et al.	MutMut	2010	Java
Papadakis et al.	GenMutants	2010	.Net
Papadakis et al.	Judy	2010	Java
Papadakis et al.	webMuJava	2010	Web (HTML/JSP)
Papadakis et al.	Bacterio	2010	Java
Papadakis et al.	Not Named	2010	N/A
Papadakis et al.	Major	2011	Java
Papadakis et al.	Paraμ	2011	Java
Papadakis et al.	ILMutator	2011	C#
Papadakis et al.	SMutant	2011	Others programming lang. (Smalltalk)
Papadakis et al.	MuBPEL	2011	Web (WS-BPEL)
Papadakis et al.	jMuHLPSL	2011	Security (HLPSL)
Papadakis et al.	Not Named	2011	Others (SPADE)
Papadakis et al.	Not Named	2011	Others (Aglets)
Papadakis et al.	Not Named	2011	Java
Papadakis et al.	SMT-C	2012	C/C++
Papadakis et al.	mutant (muRuby)	2012	Ruby
Papadakis et al.	Not Named	2012	Security (Obligation policies)
Papadakis et al.	Not Named	2012	N/A
Papadakis et al.	CCMUTATOR	2013	C/C++
Papadakis et al.	Comutation	2013	Java
Papadakis et al.	SchemaAnalyst	2013	SQL
Papadakis et al.	XACMUT	2013	Security (XACML)
Papadakis et al.	Mutandis	2013	JavaScript
Papadakis et al.	Not Named	2013	Web (Web service compositions)
Papadakis et al.	Not Named	2013	Security (Security policies)
Papadakis et al.	Not Named	2013	Model/Specification (Feature models)
Papadakis et al.	MutPy	2014	Python
Papadakis et al.	MuCheck	2014	Others programming languages (Haskell)
Papadakis et al.	HOMAJ	2014	Java
Papadakis et al.	Not Named	2014	Web (HTML/CSS)
Papadakis et al.	Not Named	2014	Model/Specification (EFSM)
Papadakis et al.	Not Named	2014	Others (Data flow languages)
Papadakis et al.	MutaLog	2014	Others (Logic Mutation)
Papadakis et al.	REDECHECK	2015	Web (HTML/CSS)
Papadakis et al.	Not Named	2015	Others (Spreadsheets)
Papadakis et al.	Not Named	2015	Model/Specification (FSM)
Papadakis et al.	Not Named	2015	Model/Specification (Sequence diagrams)
Papadakis et al.	Not Named	2015	Web (HTML /JavaScript)
Papadakis et al.	Not Named	2015	C/C++
Papadakis et al.	Not Named	2015	Android
Papadakis et al.	MoMut	2015	Model/Specification (UML models)
Papadakis et al.	MuVM	2016	C/C++

Table 3 (continued)

Source	Name	Year	Domain
Papadakis et al.	Not Named	2016	Others programming languages (FBD)
Papadakis et al.	Not Named	2016	Simulink
Papadakis et al.	Not Named	2016	C/C++
Papadakis et al.	Not Named	2016	C/C++
Papadakis et al.	Not Named	2016	C/C++
Papadakis et al.	Not Named	2016	N/A
Papadakis et al.	Vibes	2016	Model/Specification (Transition syst.)
Papadakis et al.	µDroid	2017	Android
Papadakis et al.	MDroid+	2017	Android
Papadakis et al.	Not named	2017	Others (Source code)
Papadakis et al.	LittleDarwin	2017	Java
Papadakis et al.	MuCPP	2017	C/C++
Papadakis et al.	MutRex	2017	Others (Regular Expressions)
Papadakis et al.	BacterioWeb	2017	Android
Papadakis et al.	Not Named	2017	C/C++
Papadakis et al.	Not Named	2017	C/C++
Papadakis et al.	Not Named	2017	Java
Literature	WODEL (Gómez-Abajo et al. 2018)(Gómez-Abajo et al. 2018)	2018	Model/Specification
Literature	MuTomVo (Cañizares et al. 2018)	2018	C/C++
Literature	µUTA (Siavashi et al. 2018)	2018	Model/Specification
Literature	MUSIC (Phan et al. 2018)	2018	C/C++
Literature	Mull (Denisov and Pankevich 2018)	2018	C/C++
Literature	Mutode (Rodríguez-Baquero and Linares-Vásquez 2018)	2018	JavaScript
Literature	universalmutator (Groce et al. 2018)	2018	Others (General)
Literature	MuAlloy (Wang et al. 2018)	2018	Others programming languages (Alloy)
Literature	DeepMutation (Ma et al. 2018; Hu et al. 2019)	2018	Python
Literature	Not Named (Bashir and Nadeem 2018)	2018	Java
Literature	MUTWEB (Suguna Mallika and Rajya Lakshmi 2019)	2019	Web (Java/Servlet)
Literature	Deviant (Chapman et al. 2019)	2019	Solidity
Literature	SRCIROR (Hariri et al. 2019)	2019	Others (LLVM-IR)
Literature	Edroid (Luna and Ariss 2018)	2019	Android
Literature	MuEPL (Gutiárrez-Madroñal et al. 2019)	2019	Others (EPL)
Literature	Not Named (Efremidis et al. 2018)	2019	Others programming languages (Prolog)
Literature	Not Named (Ngambenchawong and Suwannasart 2019)	2019	Model/Specification (BPMN)
Literature	Not Named (Momigliano and Ornaghi 2019)	2019	Others programming languages (aProlog)
Literature	eMuJava (Bashir and Nadeem 2019)	2019	Java

Table 3 (continued)

Source	Name	Year	Domain
Literature	MutAPK (Escobar-Velásquez et al. 2019; Escobar-Velásquez et al. 2020)	2019	Android
Literature	MuSC (Li et al. 2019)	2019	Solidity
Literature	OSWN (Sadath and Nair 2019)	2019	N/A
Literature	Mart (Chekam et al. 2019)	2019	Others (LLVM-IR)
Literature	Not Named (Van Phol and Binh 2020)	2020	Others programming languages (Lustre)
Literature	MuHyb (Ahmed et al. 2020)	2020	Web (Angular)
Literature	Not Named (Rodrigues et al. 2020)	2020	Java
Literature	Styx (Liu et al. 2020)	2020	Others (Training data)
Literature	Mutation tool for annotations (Pinheiro et al. 2020)	2020	Others (Java/C# annotations)
Literature	TECAMU (Jovanovikj et al. 2020)	2020	Others (EMSL)
Literature	RegularMutator (Ivanova and Khritankov 2020)	2020	Solidity
Literature	Muteria (Chekam et al. 2020)	2020	Others (General)
GitHub	Bamsurgeon (https://github.com/adamewing/bamsurgeon)	2012	Others (.bam files)
GitHub	Humbug (https://github.com/humbug/humbug)	2014	PHP
GitHub	Go-Mutesting (https://github.com/zimmski/go-mutesting)	2014	Others programming languages (Go)
GitHub	Cosmic-Ray (https://cosmic-ray.readthedocs.io/en/latest/index.html)	2015	Python
GitHub	StrykerJS (https://stryker-mutator.io/docs/stryker-js/introduction)	2016	JavaScript
GitHub	Mutmut (https://github.com/boxed/mutmut)	2016	Python
GitHub	Infection (https://infection.github.io/)	2017	PHP
GitHub	Stryker-NET (https://stryker-mutator.io/docs/stryker-net/Introduction)	2018	Others programming languages (.Net)
GitHub	Muter (https://github.com/muter-mutation-testing/muter)	2018	Others programming languages (Swift)

Supplementary Information (SI) For the sake of replicability, we provide a supplementary package containing the data related to our study, available at reference (https://doi.org/10.5281/zenodo.5713585).

Acknowledgements This work has been partially supported by the European Commission (ERDF) and Junta de Andalucía under projects APOLO (US-1264651) and EKIPMENT-PLUS (P18-FR-2895), and by the Spanish Government under projects HORATIO (RTI2018-101204-B-C21) and FAME (RTI2018-093608-BC33).

Funding Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature.

References

Acree AT, Budd TA, DeMillo RA, Lipton RJ, Sayward FG (1979) Mutation analysis. techreport GIT-ICS-79/08, Georgia Institute of Technology, Atlanta, Georgia

Aggarwal CC, Zhai C (2012) Mining text data. Springer Science & Business Media

Ahmed S, Taj-Eddin IATF, Ismail MA (2020) MuHyb: A proposed mutation testing tool for hybrid mobile applications. In: Proceedings of the 2020 9th International Conference on Software and Information Engineering (ICSIE). ICSIE 2020. Association for Computing Machinery, New York, NY, USA, pp 67–72. https://doi.org/10.1145/3436829.3436848

Apache Maven Project. https://maven.apache.org/. accessed in October 2021

Arcuri A (2018) An experience report on applying software testing academic results in industry: we need usable automated test generation. Empir Softw Eng 23(4):1959–1981. https://doi.org/10.1007/s10664-017-9570-9

Bamsurgeon. https://github.com/adamewing/bamsurgeon

Bashir MB, Nadeem A (2018) An experimental tool for search-based mutation testing. In: 2018 international conference on frontiers of information technology (FIT), pp 30–34. https://doi.org/10.1109/FIT.2018.00013

Bashir MB, Nadeem A (2019) An evolutionary mutation testing system for Java programs: eMuJava. In: Intelligent computing-proceedings of the computing conference. Springer, pp 847–865

Beller M, Wong C, Bader J, Scott A, Machalica M, Chandra S, Meijer E (2021) What it would take to use mutation testing in industry – a study at Facebook. In: 2021 IEEE/ACM 43rd international conference on software engineering: software engineering in practice (ICSE-SEIP). IEEE Computer Society, Los Alamitos, CA, USA, pp 268–277. https://doi.org/10.1109/ICSE-SEIP52600.2021.00036

Bitbucket. https://bitbucket.org/. accessed in October 2021

Borges H, Hora A, Valente MT (2016) Understanding the factors that impact the popularity of GitHub repositories. In: 2016 IEEE international conference on software maintenance and evolution (ICSME), pp 334–344. https://doi.org/10.1109/ICSME.2016.31

Brito G, Valente MT (2020) REST vs GraphQL: A controlled experiment. In: 2020 IEEE international conference on software architecture, ICSA 2020, Salvador, Brazil, March 16-20, 2020. IEEE, pp 81–91. https://doi.org/10.1109/ICSA47634.2020.00016

Cañizares PC, Núñez A, Merayo MG (2018) Mutomvo: Mutation testing framework for simulated cloud and HPC environments. J Syst Softw 143:187–207. https://doi.org/10.1016/j.jss.2018.05.010

Chapman P, Xu D, Deng L, Xiong Y (2019) Deviant: A mutation testing tool for Solidity Smart Contracts. In: 2019 IEEE International Conference on Blockchain (Blockchain), pp 319–324. https://doi.org/10.1109/Blockchain.2019.00050

Chekam TT, Papadakis M, Le Traon Y (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. ESEC/FSE 2019. Association for Computing Machinery, New York, NY, USA, pp 1080–1084. https://doi.org/10.1145/3338906.3341180

Chekam TT, Papadakis M, Traon YL (2020) Muteria: An extensible and flexible multi-criteria software testing framework. In: Proceedings of the IEEE/ACM 1st international conference on automation of software test. AST '20. Association for Computing Machinery, New York, NY, USA, pp 97–100. https://doi.org/10.1145/3387903.3389316

Coles H, Laurent T, Henard C, Papadakis M, Ventresque A (2016) PIT: A practical mutation testing tool for Java (demo). In: Proceedings of the 25th international symposium on software testing and analysis. ISSTA 2016. Association for Computing Machinery, New York, NY, USA, pp 449–452. https://doi.org/10.1145/2931037.2948707

Cosentino V, Izquierdo JLC, Cabot J (2016) Findings from GitHub: Methods, datasets and limitations. In: 2016 IEEE/ACM 13th working conference on mining software repositories (MSR), pp 137–141

Cosmic-Ray. https://cosmic-ray.readthedocs.io/en/latest/index.html

Delahaye M, du Bousquet L (2015) Selecting a software engineering tool: lessons learnt from mutation analysis. Software: Practice and Experience 45(7):875–891. https://doi.org/10.1002/spe.2312

Delamaro ME, Maldonado JC, Vincenzi AMR (2001) Proteum/im 2.0: An integrated mutation testing environment. In: Wong WE (ed) Mutation Testing for the New Century. Springer US, Boston, MA, pp 91–101. https://doi.org/10.1007/978-1-4757-5939-6_17

Delgado-Pérez P, Habli I, Gregory S, Alexander R, Clark J, Medina-Bulo I (2018) Evaluation of mutation testing in a nuclear industry case study. IEEE Trans Reliab 67(4):1406–1419. https://doi.org/10.1109/TR.2018.2864678

DeMillo RA, Lipton RJ, Sayward FG (1978) Hints on test data selection: Help for the practicing programmer. Computer 11(4):34–41. https://doi.org/10.1109/C-M.1978.218136

DeMillo RA, Lipton RJ, Sayward FG (1979) Program mutation: A new approach to program testing. In: Infotech State of the Art Report, Software Testing, pp 107–126

Denisov A, Pankevich S (2018) Mull it over: Mutation testing based on LLVM. In: 2018 IEEE international conference on software testing, verification and validation workshops (ICSTW), pp 25–31. https://doi.org/10.1109/ICSTW.2018.00024

Derezińska A, Hałas K (2014) Analysis of mutation operators for the Python language. In: Proceedings of the Ninth international conference on dependability and complex systems DepCoS-RELCOMEX. June 30–July 4, 2014, Brunów, Poland. Springer, pp 155–164. https://doi.org/10.1007/978-3-319-07013-1_15

Efremidis A, Schmidt J, Krings S, Körner P (2018) Measuring coverage of Prolog programs using mutation testing. In: International workshop on functional and constraint logic programming. Springer, pp 39–55

Escobar-Velásquez C, Riveros D, Linares-Vásquez M (2020) MutAPK 2.0: A tool for reducing mutation testing effort of Android apps. In: Proceedings of the 28th ACM joint meeting on european software engineering conference and symposium on the foundations of software engineering. ESEC/FSE 2020. Association for Computing Machinery, New York, NY, USA, pp 1611–1615. https://doi.org/10.1145/3368089.3417942

Escobar-Velásquez C, Osorio-Riaño M, Linares-Vásquez M (2019) MutAPK: Source-codeless mutant generation for Android apps. In: 2019 34th IEEE/ACM international conference on automated software engineering (ASE), pp 1090–1093. https://doi.org/10.1109/ASE.2019.00109

GitHub GraphQL API. https://docs.github.com/en/graphql. accessed in October 2021

Gómez-Abajo P, Guerra E, de Lara J, Merayo MG (2018) A tool for domain-independent model mutation. Sci Comput Program 163:85–92. https://doi.org/10.1016/j.scico.2018.01.008

Go-Mutesting. https://github.com/zimmski/go-mutesting

Gómez-Abajo P, Guerra E, de Lara J, Merayo MG (2018) Towards a model-driven engineering solution for language independent mutation testing. In: JISBD2018. SISTEDES. http://hdl.handle.net/11705/JISBD/2018/052

Gonzalez D, Zimmermann T, Nagappan N (2020) The state of the ML-Universe: 10 years of artificial intelligence & machine learning software development on GitHub. In: Proceedings of the 17th international conference on mining software repositories. MSR '20. Association for Computing Machinery, New York, NY, USA, pp 431–442. https://doi.org/10.1145/3379597.3387473

Gousios G, Spinellis D (2017) Mining software engineering data from GitHub. In: 2017 IEEE/ACM 39th international conference on software engineering companion (ICSE-C), pp 501–502. https://doi.org/10.1109/ICSE-C.2017.164

Gousios G (2013) The GHTorent dataset and tool suite. In: Proceedings of the 10th working conference on mining software repositories. MSR '13. IEEE Press, pp 233–236

Gradle Build Tool. https://gradle.org/. accessed in October 2021

Groce A, Holmes J, Marinov D, Shi A, Zhang L (2018) An extensible, regular-expression-based tool for multi-language mutant generation. In: Proceedings of the 40th international conference on software engineering: companion proceeedings. ICSE '18. Association for Computing Machinery, New York, NY, USA, pp 25–28, https://doi.org/10.1145/3183440.3183485

Gutiárrez-Madroñal L, Medina-Bulo I, Domínguez-Jiménez JJ (2019) Evaluation of EPL mutation operators with the MuEPL mutation system. Expert Syst Appl 116:78–95. https://doi.org/10.1016/j.eswa.2018.09.003

Hariri F, Shi A, Fernando V, Mahmood S, Marinov D (2019) Comparing mutation testing at the levels of source code and compiler intermediate representation. In: 2019 12th IEEE conference on software testing, validation and verification (ICST), pp 114–124. https://doi.org/10.1109/ICST.2019.00021

Hu Q, Ma L, Xie X, Yu B, Liu Y, Zhao J (2019) Deepmutation++: A mutation testing framework for deep learning systems. In: Proceedings of the 34th IEEE/ACM international conference on automated software engineering. ASE '19. IEEE Press, pp 1158–1161. https://doi.org/10.1109/ASE.2019.00126

Humbug. https://github.com/humbug/humbug

Infection. https://infection.github.io/

Irvine SA, Pavlinic T, Trigg L, Cleary JG, Inglis S, Utting M (2007) Jumble Java byte code to measure the effectiveness of unit tests. In: Testing: Academic and industrial conference practice and research techniques - MUTATION (TAICPART-MUTATION 2007), pp 169–175. https://doi.org/10.1109/TAIC.PART.2007.38

Ivanova Y, Khritankov A (2020) RegularMutator: A mutation testing tool for Solidity Smart Contracts. Procedia Computer Science 178:75–83. https://doi.org/10.1016/j.procs.2020.11.009. 9th International Young Scientists Conference in Computational Science, YSC2020, 05-12 September 2020

Jester. http://jester.sourceforge.net/

Jia Y, Harman M (2011) An analysis and survey of the development of mutation testing. IEEE Trans Softw Eng 37(5):649–678. https://doi.org/10.1109/TSE.2010.62

Jovanovikj I, Weidmann N, Yigitbas E, Anjorin A, Sauer S, Engels G (2020) A model-driven mutation framework for validation of test case migration. In: Babur O, Denil J, Vogel-Heuser B (eds) Proceedings of the First international conference on systems modelling and management, ICSMM 2020. Springer. https://doi.org/10.1007/978-3-030-58167-1_2

Just R, Krinke J, Li N, Rojas JM (2019) Special issue on mutation testing and analysis. Software Testing, Verification and Reliability 29(1-2):e1697. https://doi.org/10.1002/stvr.1697

Just R (2014) The Major mutation framework: Efficient and scalable mutation analysis for Java. In: Proceedings of the 2014 international symposium on software testing and analysis. ISSTA 2014. Association for Computing Machinery, New York, NY, USA, pp 433–436. https://doi.org/10.1145/2610384.2628053

Just R, Jalali D, Inozemtseva L, Ernst MD, Holmes R, Fraser G (2014) Are mutants a valid substitute for real faults in software testing?. FSE 2014. Association for Computing Machinery, New York, NY, USA, pp 654–665. https://doi.org/10.1145/2635868.2635929

Kalliamvakou E, Gousios G, Blincoe K, Singer L, German DM, Damian D (2014) The promises and perils of mining GitHub. In: Proceedings of the 11th working conference on mining software repositories. MSR 2014. Association for Computing Machinery, New York, NY, USA, pp 92–101. https://doi.org/10.1145/2597073.2597074

Kintis M, Papadakis M, Papadopoulos A, Valvis E, Malevris N, Le Traon Y (2018) How effective are mutation testing tools? An empirical analysis of java mutation testing tools with manual analysis and real faults. Empir Softw Eng 23(4):2426–2463. https://doi.org/10.1007/s10664-017-9582-5

Li Z, Wu H, Xu J, Wang X, Zhang L, Chen Z (2019) Musc: A tool for mutation testing of Ethereum smart contract. In: 2019 34th IEEE/ACM international conference on automated software engineering (ASE), pp 1198–1201. https://doi.org/10.1109/ASE.2019.00136

Liu M, Hong W, Pan W, Feng C, Chen Z, Wang J (2020) Styx: A data-oriented mutation framework to improve the robustness of DNN. In: Proceedings of the 35th IEEE/ACM international conference on automated software engineering. ASE ’20. Association for Computing Machinery, New York, NY, USA, pp 1260–1261. https://doi.org/10.1145/3324884.3418903

Luna E, Ariss OE (2018) Edroid: A mutation tool for Android apps. In: 2018 6th international conference in software engineering research and innovation (CONISOFT), pp 99–108. https://doi.org/10.1109/CONISOFT.2018.8645883

Ma L, Zhang F, Sun J, Xue M, Li B, Juefei-Xu F, Xie C, Li L, Liu Y, Zhao J, Wang Y (2018) Deepmutation: Mutation testing of deep learning systems. In: 2018 IEEE 29th international symposium on software reliability engineering (ISSRE), pp 100–111. https://doi.org/10.1109/ISSRE.2018.00021

Ma Y-S, Offutt J, Kwon YR (2005) MuJava: an automated class mutation system. Software Testing, Verification and Reliability 15(2):97–133. https://doi.org/10.1002/stvr.308

Márki A, Lindström B (2017) Mutation tools for Java. In: Proceedings of the symposium on applied computing. SAC ’17. Association for Computing Machinery, New York, NY, USA, pp 1364–1415. https://doi.org/10.1145/3019612.3019825

Mombach1 T, Valente MT (2018) GitHub REST API vs GHTorrent vs GitHub Archive: A comparative study. https://homepages.dcc.ufmg.br/~mtov/pub/2018-vem-thais.pdf

Momigliano A, Ornaghi M (2019) The blame game for property-based testing. In: CILC, pp 4–13

HtmlUnit. https://htmlunit.sourceforge.io. accessed in October 2021

MuJava download. https://cs.gmu.edu/~offutt/mujava/#Links. accessed in October 2021

MuJava. https://github.com/jeffoffutt/muJava. accessed in October 2021

Mull releases, https://github.com/mull-project/mull/releases. accessed in October 2021

Mutant. https://github.com/mbj/mutant. accessed in October 2021

Muter. https://github.com/muter-mutation-testing/muter

Mutmut. https://github.com/boxed/mutmut. accessed in October 2021

MutPy. https://github.com/mutpy/mutpy. accessed in October 2021

Ngambenchawong C, Suwannasart T (2019) A weak mutation testing framework for BPMN. In: Proceedings of the international multiconference of engineers and computer scientists, IMECS 2019, pp 13–15

Offutt AJ, Untch RH (2001) Mutation 2000: Uniting the orthogonal. Kluwer Academic Publishers, USA, pp 34–44

Papadakis M, Just R (2017) Special issue on mutation testing. Inf Softw Technol 81(C):1–2. https://doi.org/10.1016/j.infsof.2016.08.003

Papadakis M, Kintis M, Zhang J, Jia Y, Traon YL, Harman M (2019) Chapter six - mutation testing advances: An analysis and survey. Elsevier, pp 275–378. https://doi.org/10.1016/bs.adcom.2018.03.015

Petrović G, Ivanković M, Fraser G, Just R (2021) Does mutation testing improve testing practices? In: 2021 IEEE/ACM 43rd international conference on software engineering (ICSE). IEEE Computer Society, Los Alamitos, CA, USA, pp 910–921. https://doi.org/10.1109/ICSE43902.2021.00087

Petrovic G, Ivankovic M, Fraser G, Just R (2021) Practical mutation testing at scale: A view from Google. IEEE Trans Softw Eng, pp 1–1. https://doi.org/10.1109/TSE.2021.3107634

Phan DL, Kim Y, Kim M (2018) Music: Mutation analysis tool with high configurability and extensibility. In: 2018 IEEE international conference on software testing, verification and validation workshops (ICSTW), pp 40–46. https://doi.org/10.1109/ICSTW.2018.00026

Pinheiro P, Viana JC, Ribeiro M, Fernandes L, Ferrari F, Gheyi R, Fonseca B (2020) Mutating code annotations: An empirical evaluation on Java and C# programs. Sci Comput Program 191:102418. https://doi.org/10.1016/j.scico.2020.102418

PIT: installation with Maven and Gradle. https://pitest.org/quickstart/maven/, https://gradle-pitest-plugin.solidsoft.info/. accessed in October 2021

PIT. https://github.com/hcoles/pitest. accessed in October 2021

Pizzoleto AV, Ferrari FC, Offutt J, Fernandes L, Ribeiro M (2019) A systematic literature review of techniques and metrics to reduce the cost of mutation testing. J Syst Softw 157:110388. https://doi.org/10.1016/j.jss.2019.07.100

Pujari AK (2001) Data mining techniques. Universities press

Rodrigues E, Montecchi L, Ceccarelli A (2020) Model-driven fault injection in Java source code. In: 2020 IEEE 31st international symposium on software reliability engineering (ISSRE), pp 414–425. https://doi.org/10.1109/ISSRE5003.2020.00046

Rodríguez-Baquero D, Linares-Vásquez M (2018) Mutode: Generic JavaScript and Node.Js mutation testing tool. In: Proceedings of the 27th ACM SIGSOFT international symposium on software testing and analysis. ISSTA 2018. Association for Computing Machinery, New York, NY, USA, pp 372–375. https://doi.org/10.1145/3213846.3229504

Sadath L, Nair R (2019) Oncological inspired techniques for intelligent software testing. In: 2019 amity international conference on artificial intelligence (AICAI), pp 327–333. https://doi.org/10.1109/AICAI.2019.8701251

Siavashi F, Truscan D, Vain J (2018) Vulnerability assessment of web services with model-based mutation testing. In: 2018 IEEE international conference on software quality, reliability and security (QRS), pp 301–312. https://doi.org/10.1109/QRS.2018.00043

Stryker-NET. https://stryker-mutator.io/docs/stryker-net/Introduction

StrykerJS. https://stryker-mutator.io/docs/stryker-js/introduction

Suguna Mallika S, Rajya Lakshmi D (2019) MUTWEB-A testing tool for performing mutation testing of Java and servlet based web applications. International Journal of Innovative Technology and Exploring Engineering 8(12):5406–5413

“Supplementary material of the paper”? [Online]. Available at https://doi.org/10.5281/zenodo.5713585

The 17th International Workshop on Mutation Analysis - MUTATION 2022. https://icst2022.vrain.upv.es/home/mutation-2022/

Usaola MP, Mateo PR (2010) Mutation testing cost reduction techniques: A survey. IEEE Softw 27(3):80–86. https://doi.org/10.1109/MS.2010.79

Uzunbayir S, Kurtel K (2019) An analysis on mutation testing tools for C# programming language. In: 2019 4th international conference on computer science and engineering (UBMK), pp 439–443. https://doi.org/10.1109/UBMK.2019.8907222

Van Phol L, Binh NT (2020) Optimizing mutant generation for Lustre programs with multi-threading. In: 2020 5th international conference on innovative technologies in intelligent systems and industrial applications (CITISIA), pp 1–5. https://doi.org/10.1109/CITISIA50690.2020.9397490

Wang K, Sullivan A, Khurshid S (2018) MuAlloy: A mutation testing framework for Alloy. In: Proceedings of the 40th international conference on software engineering: companion proceeedings. ICSE ’18. Association for Computing Machinery, New York, NY, USA, pp 29–32. https://doi.org/10.1145/3183440.3183488

Witten IH, Frank E, Hall MA, Pal CJ (eds) (2017) Data mining (fourth edition), 4th Edn. Morgan Kaufmann, Burlington. https://doi.org/10.1016/B978-0-12-804291-5.00014-3

WODEL: Get started. https://github.com/gomezabajo/Wodel/wiki/Get-Started. accessed in October 2021

Xiong Y, Meng Z, Shen B, Yin W (2017) Mining developer behavior across GitHub and StackOverflow. In: The 29th international conference on software engineering and knowledge engineering, pp 578–583. https://doi.org/10.18293/SEKE2017-062