

Widening The Adoption of Web API Fuzzing: Docker, GitHub Action and Python Support for EvoMaster

Andrea Arcuri

Kristiania University of Applied Sciences and Oslo
Metropolitan University
Oslo, Norway
andrea.arcuri@kristiania.no

Juan P. Galeotti

University of Buenos Aires and CONICET
Buenos Aires, Argentina
Kristiania University of Applied Sciences
Oslo, Norway

Philip Garrett

Kristiania University of Applied Sciences
Oslo, Norway

Man Zhang

Beihang University
Beijing, China

Abstract

Web APIs, like for example REST, GraphQL and RPC APIs, are widely used in industry. Due to their importance, a lot of research work has been carried out in the recent years on test automation in this domain. Several fuzzers have been developed and evaluated in the research literature. EvoMASTER is a state-of-the-art, search-based fuzzer for Web APIs. It is in active development, since 2016, and it is used in several enterprises around the world, like for example at Meituan and Volkswagen. In this paper, we present and discuss its most recent technical features, aimed at widening its adoption in industry. In particular, we discuss its release on Docker, its use in Continuous Integration environments such as GitHub Actions, and the support for test case outputs in Python. This enables us a better technology transfer from academic results to industrial practice. A video showcasing these new features is currently available at: <https://youtu.be/l1ybs7SjvcA>

CCS Concepts

• **Software and its engineering** → **Software verification and validation**; Search-based software engineering.

Keywords

REST API, Fuzzing, Search-based Software Testing, Test Case Generation

ACM Reference Format:

Andrea Arcuri, Philip Garrett, Juan P. Galeotti, and Man Zhang. 2025. Widening The Adoption of Web API Fuzzing: Docker, GitHub Action and Python Support for EvoMaster. In *33rd ACM International Conference on the Foundations of Software Engineering (FSE Companion '25)*, June 23–28, 2025, Trondheim, Norway. ACM, New York, NY, USA, 5 pages. <https://doi.org/10.1145/3696630.3728586>



This work is licensed under a Creative Commons Attribution 4.0 International License. FSE Companion '25, Trondheim, Norway
© 2025 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-1276-0/2025/06
<https://doi.org/10.1145/3696630.3728586>

1 Introduction

Enterprise applications rely on web services to function. These are widely used in industry, especially when dealing with microservice architectures [15, 20, 26, 29, 33]. REST APIs are currently the most common type of web service.

Due to their importance in industry, a lot of research work has been carried out to design novel techniques to automatically test this kind of web services [18]. Several fuzzers have been presented in the research literature, like, for example, in alphabetic order: ARAT-RL [21], bBOXRT [23], DeepREST [17], EvoMASTER [2], Mor-est [24], ResTest [25], RestCT [34], Restler [14], RestTestGen [32], and Schemathesis [19].

Among those, EvoMASTER [2, 3, 8, 13] was the first to be presented (open-sourced since 2016), and the one used in most scientific studies [18]. It is a mature open-source tool, downloaded thousands of times [13], and actively used in large Fortune 500 enterprises such as Meituan [37–39] and Volkswagen [9, 28]. In *independent* comparisons conducted by other researchers, EvoMASTER consistently ranked among the top tools for both code coverage and the number of detected failures [22, 30]. Furthermore, among these cited tools, EvoMASTER is the *only one that currently supports white-box testing* [4]. Based on advanced white-box heuristics [7, 11], and special handling of databases [6] and external services [31], *white-box* EvoMASTER can provide significantly better results than all other *black-box* fuzzers [36].

Building a research prototype that can be widely used in industry is no easy task [10]. There are several technical features that are needed to be able to have an effective technology transfer from academic research results to industrial practice. In this paper, we present and discuss three recent technical features added to EvoMASTER, to widen its use among practitioners in industry:

- **Docker:** enable to run EvoMASTER via Docker.¹ This makes trying out EvoMASTER easier, as a potential user can just copy&paste a command on a terminal to try EvoMASTER out without needing to do any installation or manual download first.
- **GitHub Action:** automated test generation in Continuous Integration (CI) can simplify the use of fuzzers [5, 16, 27, 38].

¹<https://www.docker.com/>

Currently, GitHub Actions² is one of the most common CI provider. We implemented a new “action” to enable running EvoMASTER and analyze its results on such CI.

- **Python:** the output of a fuzzer should include an executable test suite. This is essential to be able to reproduce any found fault, needed for the debugging and eventually fix of such newly found fault. EvoMASTER originally supported output formats in Java and Kotlin, using JUnit. But, this is not ideal for the general black-box testing use of EvoMASTER, when the tested APIs are written in different programming languages. A more general, test-engineer (and not software-developer) friendly output type was needed. For this goal, we chose and implemented the support for the scripting language Python.

What is presented in this paper is now available in the most recent version of EvoMASTER, namely 3.4.0. EvoMASTER is open-source on GitHub,³ with each release automatically uploaded to Zenodo for long-term storage (e.g., [12]). These features enable the scientific results of the work done on EvoMASTER (e.g., resource-dependency handling [41] and adaptive hyper-mutation [35]) to get a wider application in industry.

2 Docker Support

EvoMASTER is written in Kotlin, and it is released as an executable JAR file. For example, it can be run with:

```
java -jar evomaster.jar <args>
```

Doing this requires two steps: downloading the `evomaster.jar` file from the release page⁴ of EvoMASTER, and install a JDK. This latter step might not be necessary if the user is a Java/Kotlin developer, as likely they would already have a JDK installed. However, for test and quality assurance engineers, they might have nothing to do with the JDK. This is especially the case when they test APIs within an organization that does not use Java/Kotlin as programming language for their backends. Furthermore, even if a JDK is installed, then a user would need to make sure to run the `java` program from the folder in which the `evomaster.jar` file is saved into, or provide each time its absolute path (which might be inconvenient).

A first step to alleviate such an issue was to employ the JPack utility (provided bundled with the JDK since JDK 17) to build installers (including JDK and `evomaster.jar`) for the different main operating systems (e.g., `.msi` for Windows and `.dmg` for macOS). These installers are available since EvoMASTER version 1.2.0, but it still requires to manually download such files from the release page of EvoMASTER, and to manually update the `PATH` environment variable to point to the newly installed executable files (unfortunately, JPack does not seem to have yet any user-friendly option to setup this `PATH` configuration automatically as part of the installer).

Still, we considered this was not enough. A user that knows what benefits EvoMASTER can bring would have no particular problem in downloading and installing a new EvoMASTER version from its release page. However, potential users who wish to evaluate the tool’s functionality may be discouraged by the requirement to install new software. It would be much simpler, and possibly more

```
1 # Calls:
2 # (200) POST:/v2/store/order
3 @timeout_decorator.timeout(60)
4 def test_1(self):
5
6     headers = {}
7     headers["content-type"] = "application/json"
8     body = {}
9     body = " { " + \
10         " \"id\": -6916732944634204757, " + \
11         " \"petId\": 850, " + \
12         " \"shipDate\": \"2042-02-18T05
13         :03:34.527-09:11\", " + \
14         " \"status\": \"placed\" " + \
15         " } "
16     headers['Accept'] = "application/json"
17     res_0 = requests \
18         .post(self.baseUrlOfSut + "/v2/store/order",
19              headers=headers, data=body)
20
21     assert res_0.status_code == 200
22     assert "application/json" in res_0.headers["content-
23     type"]
24     assert res_0.json()["petId"] == 850.0
25     assert res_0.json()["quantity"] == 0.0
26     assert res_0.json()["shipDate"] == "2042-02-18T14
27     :14:34.527+0000"
28     assert res_0.json()["status"] == "placed"
29     assert res_0.json()["complete"] == False
```

Figure 1: Example of generated test, in Python.

effective, to copy&paste a console command from the *readme* page of the tool,³ without any installation, like for example:

```
docker run -v "$(pwd)/generated_tests":/generated_tests
webfuzzing/evomaster --blackBox true --maxTime 30s
--ratePerMinute 60 --bbSwaggerUrl https://petstore
.swagger.io/v2/swagger.json
```

Note, if run in a MSYS (Minimal SYStem) shell on Windows like *Git Bash*, there is the need of an extra `/` before the `$`.

This command runs EvoMASTER via Docker, configured for black-box fuzzing of the *PetClinic* example API of Swagger, for 30 seconds. After the command is completed, all generated tests are available under the folder `generated_tests`. An example of such a generated test can be found in Figure 1. Note that re-running such command might likely lead to different generated tests, for two main reasons: (1) EvoMASTER is non-deterministic and (2) that API online on internet might have changed since we ran that command (which was done in January 2025).

Running with Docker has its own (minor) issues though. First, there is the need to sync a folder to be able to access the generated tests (e.g., see the command `-v "$(pwd)/generated_tests":/generated_tests` in the previous example). Also, references to `localhost` would not work, as those would point to the Docker network, and not the host machine. For example, running an API locally and trying to fuzz it via something like `-bbSwaggerUrl https://localhost:8080/v2/swagger.json` would fail. However, this can be fixed by using `host.docker.internal` instead of `localhost` as the hostname. As this creates an extra layer of needed configurations, that might not be too common to know how to handle unless one has significant experience with Docker, this is

²<https://github.com/features/actions>

³<https://github.com/WebFuzzing/EvoMaster>

⁴<https://github.com/WebFuzzing/EvoMaster/releases>

explained in details in our documentation.⁵ Furthermore, when EvoMASTER is run in Docker, info messages are printed in the logs to explain these details (including a link to the documentation).

Enabling the running of EvoMASTER via Docker required to prepare a (simple) Dockerfile.⁶ Then, at each new release of EvoMASTER (e.g., the latest 3.4.0), it is automatically uploaded to Docker Hub under the name webfuzzing/evomaster.⁷ This is done as part of the automated release process of EvoMASTER on GitHub, with the following configuration:⁸

```
1 - name: Docker meta
2   id: meta
3   uses: docker/metadata-action@v5
4   with:
5     images: webfuzzing/evomaster
6 - name: Login to DockerHub
7   uses: docker/login-action@v3
8   with:
9     username: ${ secrets.DOCKERHUB_USERNAME }
10    password: ${ secrets.DOCKERHUB_PASSWORD }
11 - name: Build and push Docker
12   uses: docker/build-push-action@v5
13   with:
14     context: .
15     push: true
16     tags: ${ steps.meta.outputs.tags }
17     labels: ${ steps.meta.outputs.labels }
```

3 GitHub Action

Continuous Integration (CI) is an essential component in modern software development processes. In a nutshell, at each new committed code change, the entire application is rebuilt, and all regression tests are run, to see if the new changes break any existing functionality. Due to its importance, and wide-spread use in industry, there has been work in how to integrate automated test generation in CI. An example is EvoSuite, with plugin for example for Jenkins CI [5, 16]. The integration of fuzzers in CI is an important feature that is often requested by practitioners [27, 38].

To widen the adoption of EvoMASTER, we have created a new open-source, custom GitHub Action for it, called webfuzzing/evomaster-action.⁹ GitHub Actions is one of the most popular CI platform. The idea here is that we can run EvoMASTER at each code commit (or pull requests, or when commits are done on a specific branch), and possibly fail the build if any new fault is detected in the tested API. Then, the test suites generated by EvoMASTER can be downloaded for debugging. Figure 2 shows an example of YAML configuration for GitHub Actions to enable running EvoMASTER as part of the CI build.

A working example can found in our rest-api-example repository.¹⁰ There, a simple REST API is built with Java and the Spring framework. At each code commit, EvoMASTER is run on it via our new custom GitHub Action. The build fails, as six faults are detected by EvoMASTER in that API, as it can be seen in the screenshot in Figure 3.

```
1 - name: Build the Application
2   run: ... # this will depend on your application
3
4 - name: Start Docker Compose
5   run: docker compose up -d --build
6
7 - uses: webfuzzing/evomaster-action@v1
8   with:
9     args: >-
10      --writeWFCReport true
11      --blackBox true
12      --bbSwaggerUrl http://localhost:8080/v3/api-docs
13      --maxTime 20s
14      --showProgress false
15      failOnErrors: "true"
16
17 - name: Upload Generated Files
18   if: always()
19   uses: actions/upload-artifact@v4
20   with:
21     name: results
22     path: ./generated_tests
23
24 - name: Stop Docker Compose
25   if: always()
26   run: docker compose down
```

Figure 2: Example of YAML configuration for GitHub Actions.

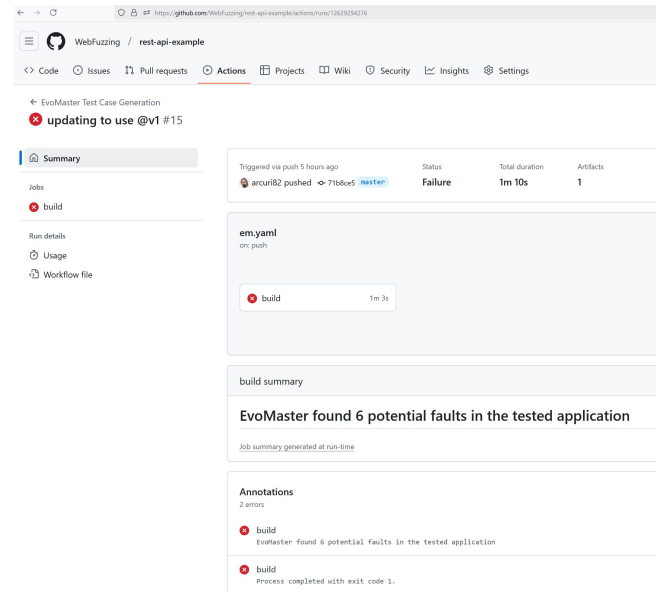


Figure 3: Screenshot of failed build due to EvoMASTER finding faults.

4 Python Test Output

For a fuzzer, it is essential to be able to output useful and executable test cases. The final goal is not just to *find* faults (if any is present), but to be able to *fix* them. To reach such goal, once a fault is detected, then the generated test revealing such fault can be used to start a

⁵<https://github.com/WebFuzzing/EvoMaster/blob/master/docs/docker.md>

⁶<https://github.com/WebFuzzing/EvoMaster/blob/master/Dockerfile>

⁷<https://hub.docker.com/r/webfuzzing/evomaster>

⁸<https://github.com/WebFuzzing/EvoMaster/blob/master/.github/workflows/release.yml>

⁹<https://github.com/WebFuzzing/evomaster-action>

¹⁰<https://github.com/WebFuzzing/rest-api-example>

debugging phase. Therefore, how a generated test suite can be used plays a major role in its effectiveness and applicability.

Since its inception in 2016, EvoMASTER has been mainly a *white-box*, search-based test generator tool [1], targeting the JVM. It has been able to output test suites in either Java or Kotlin, using the JUnit library, and RestAssured for making HTTP calls. This has been reasonable for white-box testing: if targeting APIs developed in programming language X, then use X to write the generated output test suite. This is particularly the case when the user of a fuzzer is a software developer, and not a test engineer. For the same reason, when we attempted to support white-box testing for JavaScript NodeJS APIs [40] (effort that is now no longer supported), we added JavaScript as a possible test output type.

If this is reasonable for white-box testing for a specific programming language, it might not be ideal when dealing with *black-box* testing. Assume for example a test engineer that, as part of a quality assurance team, uses EvoMASTER to test an API from a development team, written for example in Go. Generating tests in Kotlin or Java might not make much sense, if those programming languages are not used in that enterprise. Therefore, for black-box testing, there is the need of a more widespread, general option. Even if concentrating on only the major, most popular programming languages, supporting several language outputs might not be viable in an academic context. This is because it would be just “technical” work, and not a novel research contribution.

As testing output, targeting a domain specific language (DSL) of a popular tool, used to write manual tests for APIs, might sound reasonable. These are for example Postman,¹¹ Insomnia,¹² Hoppscotch¹³ and Bruno.¹⁴ However, the popularity of these commercial tools come and go. Also, choosing one of them would preclude the use of EvoMASTER in enterprises that use a different tool. Supporting more than one as test output might not be viable, unfortunately.

For all these reasons, for black-box testing we decided to support Python as default test output, besides the previous Java, Kotlin and JavaScript. Python is a programming language used for scripting and, in the grey literature, it seems quite in use in industry for test automation by test engineers. Recall Figure 1, which shows a generated test by EvoMASTER in Python. These Python tests use the library unittest, and make HTTP calls with the Requests library.

With four possible different test outputs (i.e., Java, Kotlin, JavaScript and now Python), how to make sure they work correctly is essential. It would be unfortunate to run EvoMASTER for 24 hours (for example), and then getting unusable test suites due to some faults in the output routine (e.g., generating invalid test code). As for any piece of software, EvoMASTER itself can have faults. For this reason, it is extensively tested with hundreds of unit, integration and E2E (End-to-End) tests [10].

But, testing outputs in programming languages that do not run on the JVM is not trivial. For this goal, we had to spend considerable time in extending the E2E infrastructure of EvoMASTER to handle these cases. In particular, we have created E2E JUnit tests in which we start artificial APIs on the JVM (e.g., using the Spring framework), run EvoMASTER on them in the same process (as both

can run on the JVM), and then, in the same JUnit test, we spawn a process which compiles and runs the generated tests (e.g., using NodeJS for JavaScript, and Python interpreter for Python outputs). These generated tests will connect to the API running on the JVM of our own E2E JUnit test for EvoMASTER. Once they are executed, in the JUnit E2E we verify, via static probes, what has been executed. This process is automatically run for each artificial API we created, using JUnit parameterized tests (i.e., using the annotation `@ParameterizedTest`). Here the “parameter” is the test output language. Different artificial APIs have been designed for these E2E tests, to verify how EvoMASTER can successfully handled different aspects of fuzzing REST APIs. For reasons of space, unfortunately we cannot go in more details about these advanced E2E tests. We refer the interested reader to the implementation of EvoMASTER [12], for example with the E2E tests in the `spring-rest-bb` module, such as for example `BBInputsEMTest`.

5 Conclusion

In this paper, we have presented and discussed three new important features to widen the applicability of the fuzzer EvoMASTER in industry. These are Docker, GitHub Actions and Python support.

EvoMASTER is a mature research tool, used to investigate several research questions in the field of software testing in tens of research articles in the last eight years. Future work will aim at balancing the effort between scientific novelty and new technical features, these latter aimed at a more effective technology transfer from academic research in software engineering to industrial practice.

A video showcasing these new features in EvoMASTER is currently available at: <https://youtu.be/l1ybs7SjvcA>

Acknowledgments

This work is funded by the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (EAST project, grant agreement No. 864972), and partially funded by UBACYT-2020 20020190100233BA, PICT-2019-01793. Man Zhang is supported by State Key Laboratory of Complex & Critical Software Environment (SKLCCSE, grant No. CCSE-2024ZX-01) and the Fundamental Research Funds for the Central Universities.

Data Availability

EvoMASTER and its new GitHub Action presented in this paper are open-source on GitHub,³ with each release automatically uploaded to Zenodo for long-term storage (e.g., [12]).

References

- [1] Andrea Arcuri. 2017. RESTful API Automated Test Case Generation. In *IEEE International Conference on Software Quality, Reliability and Security (QRS)*. IEEE, 9–20.
- [2] Andrea Arcuri. 2018. EvoMaster: Evolutionary Multi-context Automated System Test Generation. In *IEEE International Conference on Software Testing, Verification and Validation (ICST)*. IEEE.
- [3] Andrea Arcuri. 2019. RESTful API Automated Test Case Generation with EvoMaster. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 28, 1 (2019), 3.
- [4] Andrea Arcuri. 2020. Automated Black-and White-Box Testing of RESTful APIs With EvoMaster. *IEEE Software* 38, 3 (2020), 72–78.
- [5] Andrea Arcuri, José Campos, and Gordon Fraser. 2016. Unit Test Generation During Software Development: EvoSuite Plugins for Maven, IntelliJ and Jenkins. In *IEEE International Conference on Software Testing, Verification and Validation (ICST)*.

¹¹<https://www.postman.com>

¹²<https://insomnia.rest>

¹³<https://hoppscotch.io>

¹⁴<https://www.usebruno.com/>

- [6] Andrea Arcuri and Juan P Galeotti. 2020. Handling SQL databases in automated system test generation. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 29, 4 (2020), 1–31.
- [7] Andrea Arcuri and Juan P Galeotti. 2021. Enhancing Search-based Testing with Testability Transformations for Existing APIs. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 31, 1 (2021), 1–34.
- [8] Andrea Arcuri, Juan Pablo Galeotti, Bogdan Marculescu, and Man Zhang. 2021. EvoMaster: A Search-Based System Test Generation Tool. *Journal of Open Source Software* 6, 57 (2021), 2153.
- [9] A. Arcuri, A. Poth, and O. Rrjolli. 2025. Introducing Black-Box Fuzz Testing for REST APIs in Industry: Challenges and Solutions. In *IEEE International Conference on Software Testing, Verification and Validation (ICST)*.
- [10] Andrea Arcuri, Man Zhang, Asma Belhadi, Bogdan Marculescu, Amid Golmohammadi, Juan Pablo Galeotti, and Susruthan Seran. 2023. Building an open-source system test generation tool: lessons learned and empirical analyses with EvoMaster. *Software Quality Journal* (2023), 1–44.
- [11] Andrea Arcuri, Man Zhang, and Juan Pablo Galeotti. 2024. Advanced White-Box Heuristics for Search-Based Fuzzing of REST APIs. *ACM Transactions on Software Engineering and Methodology (TOSEM)* (2024). doi:10.1145/3652157
- [12] Andrea Arcuri, Man Zhang, Susruthan Seran, Asma Belhadi, Juan Pablo Galeotti, Bogdan, Amid Golmohammadi, Onur Duman, Agustina Aldasoro, Philip, Alberto Martín López, Hernan Ghianni, Ömür Şahin, Annibale Panichella, Kyle Niemeyer, and Marcello Maugeri. 2025. *WebFuzzing/EvoMaster: v3.4.0*. doi:10.5281/zenodo.14597412
- [13] Andrea Arcuri, Man Zhang, Susruthan Seran, Juan Pablo Galeotti, Amid Golmohammadi, Onur Duman, Agustina Aldasoro, and Hernan Ghianni. 2025. Tool report: EvoMaster—black and white box search-based fuzzing for REST, GraphQL and RPC APIs. *Automated Software Engineering* 32, 1 (2025), 1–11.
- [14] Vaggelis Atlidakis, Patrice Godefroid, and Marina Polishchuk. 2019. RESTler: Stateful REST API Fuzzing. In *ACM/IEEE International Conference on Software Engineering (ICSE)*. 748–758.
- [15] Adam Bellemare. 2020. *Building Event-Driven Microservices*. " O'Reilly Media, Inc."
- [16] José Campos, Andrea Arcuri, Gordon Fraser, and Rui Abreu. 2014. Continuous test generation: enhancing continuous integration with automated test generation. In *IEEE/ACM Int. Conference on Automated Software Engineering (ASE)*. ACM, 55–66.
- [17] Davide Corradini, Zeno Montolli, Michele Pasqua, and Mariano Ceccato. 2024. DeepREST: Automated Test Case Generation for REST APIs Exploiting Deep Reinforcement Learning. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*. 1383–1394.
- [18] Amid Golmohammadi, Man Zhang, and Andrea Arcuri. 2023. Testing RESTful APIs: A Survey. *ACM Transactions on Software Engineering and Methodology* (aug 2023). doi:10.1145/3617175
- [19] Zac Hatfield-Dodds and Dmitry Dygalo. 2022. Deriving Semantics-Aware Fuzzers from Web API Schemas. In *2022 IEEE/ACM 44th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. IEEE, 345–346.
- [20] Pooyan Jamshidi, Claus Pahl, Nabor C Mendonça, James Lewis, and Stefan Tilkov. 2018. Microservices: The journey so far and challenges ahead. *IEEE Software* 35, 3 (2018), 24–35.
- [21] Myeongsoo Kim, Saurabh Sinha, and Alessandro Orso. 2023. Adaptive rest api testing with reinforcement learning. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 446–458.
- [22] Myeongsoo Kim, Qi Xin, Saurabh Sinha, and Alessandro Orso. 2022. Automated Test Generation for REST APIs: No Time to Rest Yet. In *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis (Virtual, South Korea) (ISSTA 2022)*. Association for Computing Machinery, New York, NY, USA, 289–301. doi:10.1145/3533767.3534401
- [23] Nuno Laranjeiro, João Agnelo, and Jorge Bernardino. 2021. A black box tool for robustness testing of REST services. *IEEE Access* 9 (2021), 24738–24754.
- [24] Yi Liu, Yuekang Li, Gelei Deng, Yang Liu, Ruiyuan Wan, Runchao Wu, Dandan Ji, Shiheng Xu, and Minli Bao. 2022. Morest: Model-based RESTful API Testing with Execution Feedback. In *ACM/IEEE International Conference on Software Engineering (ICSE)*.
- [25] Alberto Martín-López, Sergio Segura, and Antonio Ruiz-Cortés. 2021. RESTTest: Automated Black-Box Testing of RESTful Web APIs. In *ACM Int. Symposium on Software Testing and Analysis (ISSTA)*. ACM, 682–685.
- [26] Sam Newman. 2021. *Building microservices*. " O'Reilly Media, Inc."
- [27] Olivier Nourry, GABRIELE BAVOTA, MICHELE LANZA, and YASUTAKA KAMEI. 2023. The Human Side of Fuzzing: Challenges Faced by Developers During Fuzzing Activities. *ACM Transactions on Software Engineering and Methodology* (2023).
- [28] Alexander Poth, Olsi Rrjolli, and Andrea Arcuri. 2025. Technology adoption performance evaluation applied to testing industrial REST APIs. *Automated Software Engineering* 32, 1 (2025), 5.
- [29] RV Rajesh. 2016. *Spring Microservices*. Packt Publishing Ltd.
- [30] Hassan Sartaj, Shaikat Ali, and Julie Marie Gjølby. 2024. REST API Testing in DevOps: A Study on an Evolving Healthcare IoT Application. arXiv:2410.12547 [cs.SE] <https://arxiv.org/abs/2410.12547>
- [31] Susruthan Seran, Man Zhang, and Andrea Arcuri. 2023. Search-Based Mock Generation of External Web Service Interactions. In *International Symposium on Search Based Software Engineering (SSBSE)*. Springer.
- [32] Emanuele Viglianisi, Michael Dallago, and Mariano Ceccato. 2020. RESTTEST-GEN: Automated Black-Box Testing of RESTful APIs. In *IEEE International Conference on Software Testing, Verification and Validation (ICST)*. IEEE.
- [33] Muhammad Waseem, Peng Liang, Mojtaba Shahin, Amleto Di Salle, and Gastón Márquez. 2021. Design, monitoring, and testing of microservices systems: The practitioners' perspective. *Journal of Systems and Software* 182 (2021), 111061.
- [34] Huayao Wu, Lixin Xu, Xintao Niu, and Changhai Nie. 2022. Combinatorial Testing of RESTful APIs. In *ACM/IEEE International Conference on Software Engineering (ICSE)*.
- [35] Man Zhang and Andrea Arcuri. 2021. Adaptive Hypermutation for Search-Based System Test Generation: A Study on REST APIs with EvoMaster. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 31, 1 (2021).
- [36] Man Zhang and Andrea Arcuri. 2023. Open Problems in Fuzzing RESTful APIs: A Comparison of Tools. *ACM Transactions on Software Engineering and Methodology (TOSEM)* (may 2023). doi:10.1145/3597205
- [37] Man Zhang, Andrea Arcuri, Yonggang Li, Yang Liu, and Kaiming Xue. 2023. White-Box Fuzzing RPC-Based APIs with EvoMaster: An Industrial Case Study. *ACM Transactions on Software Engineering and Methodology* 32, 5 (2023), 1–38.
- [38] Man Zhang, Andrea Arcuri, Yonggang Li, Yang Liu, Kaiming Xue, Zhao Wang, Jian Huo, and Weiwei Huang. 2024. Fuzzing Microservices: A Series of User Studies in Industry on Industrial Systems with EvoMaster. arXiv:2208.03988 [cs.SE] <https://arxiv.org/abs/2208.03988>
- [39] Man Zhang, Andrea Arcuri, Piyun Teng, Kaiming Xue, and Wenhao Wang. 2024. Seeding and Mocking in White-Box Fuzzing Enterprise RPC APIs: An Industrial Case Study. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*. 2024–2034.
- [40] Man Zhang, Asma Belhadi, and Andrea Arcuri. 2023. JavaScript SBST Heuristics To Enable Effective Fuzzing of NodeJS Web APIs. *ACM Transactions on Software Engineering and Methodology* (2023).
- [41] Man Zhang, Bogdan Marculescu, and Andrea Arcuri. 2021. Resource and dependency based test case generation for RESTful Web services. *Empirical Software Engineering* 26, 4 (2021), 1–61.