# AI-Driven Web API Testing

Alberto Martin-Lopez
Universidad de Sevilla
Seville, Spain
amarlop@us.es

## ABSTRACT

Testing of web APIs is nowadays more critical than ever before, as they are the current standard for software integration. A bug in an organization's web API could have a huge impact both internally (services relying on that API) and externally (third-party applications and end users). Most existing tools and testing approaches require writing tests or instrumenting the system under test (SUT). The main aim of this dissertation is to take web API testing to an unprecedented level of automation and thoroughness. To this end, we plan to apply artificial intelligence (AI) techniques for the autonomous detection of software failures. Specifically, the idea is to develop intelligent programs (we call them "bots") capable of generating hundreds, thousands or even millions of test inputs and to evaluate whether the test outputs are correct based on: 1) patterns learned from previous executions of the SUT; and 2) knowledge gained from analyzing thousands of similar programs. Evaluation results of our initial prototype are promising, with bugs being automatically detected in some real-world APIs.

## CCS CONCEPTS

• **Information systems → Web services**; **RESTful web services**; • **Software and its engineering → Software testing and debugging**.

## KEYWORDS

Web API, RESTful API, testing framework, automated software testing, artificial intelligence

**ACM Reference Format:**
Alberto Martin-Lopez. 2020. AI-Driven Web API Testing. In *42nd International Conference on Software Engineering Companion (ICSE '20 Companion), May 23–29, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3377812.3381388

## 1 INTRODUCTION

Web APIs enable the consumption of services and data over the network, typically using web services, each of which exposes one or more endpoints. An *endpoint* accepts requests and returns responses, usually over HTTP, and allows to perform operations (e.g. translate a text) or access a particular piece of information (e.g. a photo). Modern web APIs generally adhere to the REpresentational State Transfer (REST) architectural style [24], being referred to as RESTful web APIs. *RESTful web APIs* are usually decomposed into multiple RESTful web services [31], each of which implements one or more create, read, update or delete (CRUD) operations over a specific resource (e.g. a song in the Spotify API).

As web APIs become the preferred choice for the seamless integration of heterogeneous software systems, their validation becomes more critical than ever before. A bug in an organization's API could have a huge impact both internally (services relying on that API) and externally (third-party applications and end users). Multiple works have addressed the problem of testing web services based on SOAP [1] or WSDL [2], technologies increasingly in disuse nowadays. However, few of them have addressed the unique characteristics of RESTful APIs, which are the current de facto standard. According to ProgrammableWeb [8], a popular API repository, more than 80% (14,903 out of 18,279) of the APIs recorded in its directory up to November 2017 are RESTful APIs. It is on this account that this dissertation revolves mainly around RESTful API testing. More specifically, we focus on APIs described with the OpenAPI Specification (OAS) [6], which is the industry standard for RESTful API design. Nevertheless, we aspire our contributions to be general enough to be applicable to other types of web services and API specification languages (e.g. RAML [12]).

Some of the current approaches of RESTful API testing involve the tester in the task of writing tests, thereby hindering automation. Other approaches require the instrumentation of the SUT, which may hinder their adoption. A number of techniques achieve a high degree of automation, but fail at generating test cases that are complex enough to exercise deep functionality of the program, which hinders thoroughness. Moreover, some papers focus on white-box testing, however, source code of web APIs is not always available, and so these techniques cannot always be applied. Lastly, to the best of our knowledge, no previous work has yet attempted to leverage AI techniques (other than search-based) in the context of web API testing. More and more testing companies such as Applitools [4] and test.ai [13] (the latter founded by Jason Arbon, ex lead tester at Google) are betting on AI as the next revolution for automating software tests (in these cases, UI and mobile testing, respectively).

Our proposal seeks to address all these issues, reaching an unprecedented level of automation and thoroughness. We plan to develop a specification-driven testing framework for the automated generation of complex test cases for web APIs. We intend to use AI for the autonomous detection of software failures. Specifically, our aim is to develop intelligent programs (called "bots") capable of generating hundreds, thousands or even millions of test inputs following a model-based approach, where the API specification will drive the generation of test data using a variety of testing techniques (e.g. combinatorial, search-based and metamorphic testing).


*ICSE '20 Companion, May 23–29, 2020, Seoul, Republic of Korea*

ACM ISBN 978-1-4503-7122-3/20/05…$15.00
https://doi.org/10.1145/3377812.3381388

The test outputs will also be automatically evaluated based on: 1) patterns learned from previous executions of the SUT; and 2) knowledge gained from analyzing thousands of similar programs. This testing framework could be offered as a service, opening a new range of business opportunities. For example, API providers could have their APIs automatically tested and pay depending on the type of service contracted (e.g. 24/7 testing or 100% test coverage).

## 2 RELATED WORK

Numerous authors have addressed the testing of Service-Oriented Architectures (SOA) in general and web services in particular. Surveys on the topic [17, 18] point out that this issue started to gain attention with the popularization of service-centric systems. However, most papers focus on SOAP web services and WSDL, technologies that have fallen in disuse and are nowadays being displaced by more modern alternatives such as REST or GraphQL [5].

Regarding RESTful web services testing, multiple tools and libraries are available in the market, including Postman [7], REST Assured [10], ReadyAPI [9] and API Fortress [3]. These tools allow to create and execute test cases as complex as desired, nevertheless, these must be written by the tester, which hinders automation.

From an academic point of view, RESTful API testing has received less attention than SOA testing, but interest has been continuously increasing over the last decade. Several works have proposed domain-specific languages [19, 30], formal notations [20] and frameworks [22] for the description of test cases or service specifications. Other approaches rely on manually defined models of the system or models of the test data format from which test cases can be automatically generated [16, 23, 25, 29]. However, such domain-specific tools and model-based approaches may suffer from usability problems and adoption in industry, since the tester must familiarize with them instead of using already-known standards. Furthermore, these and other techniques such as [14, 33] can still be improved in terms of automation, since the tester must usually perform some preliminary task or instrument the SUT prior to the generation and execution of test cases. On the other hand, some works leverage the OAS document of the API to automate the testing process. For instance, Ed-Douibi et al. [21] automatically generated test cases for 91 APIs and Atlidakis et al. [15] developed an autonomous fuzzer for OAS-based APIs. While both techniques succeeded in finding real faults, there is room for improvement in the way they generate input test data: the first approach mainly uses default values and in the second the user must manually define valid values for each data type. The lack of complex test data may lead to a lack of thoroughness in the automatically generated test suites. Lastly, a few approaches focus on white-box testing of RESTful APIs [14, 34]. Nevertheless, source code of web APIs is not always available, and so these techniques cannot always be applied.

## 3 PROBLEM

The hypothesis that motivates this PhD can be described as follows: *It is possible to apply AI techniques to detect software failures in web APIs in an autonomous way, increasing testing effectiveness and significantly reducing its cost, through less human intervention.* This hypothesis leads us to address the following research questions:

**RQ1: Is it possible to thoroughly test RESTful APIs just using their formal specification?** Current API design languages such as OAS or RAML provide a structured way to describe a RESTful API in a both human- and machine-readable way, allowing to automatically generate source code or documentation, for instance. However, it is unclear to what extent the specification can be used to automate the generation of complex test cases that exercise deep functionality of the API. We aim to further investigate this issue.

**RQ2: Which advanced testing techniques are best suited for testing web APIs?** Web APIs pose challenges in testing that no other systems do. For instance, in the context of search-based testing, if source code is not available, which fitness functions are more effective for driving the generation of test data? Furthermore, most techniques focus on the generation of test inputs and disregard the oracle problem, which is prominent in these kinds of systems. In this sense, metamorphic testing stands out as an alternative to approach this issue.

**RQ3: How can AI support web API testing?** The potential of AI in web API testing is very much unknown, as it has hardly been used in this context. AI could free human testers from several tasks, for instance: 1) multiple resources from the API such as the documentation or the issue tracking system could be used to generate more meaningful test cases; 2) natural language processing could be used to infer useful parameter values based on the parameter descriptions; 3) new realistic test cases could be created by analyzing real calls to the service. The potential applications of AI and their challenges will be one of the main research lines of this dissertation.

**RQ4: Is it possible to offer continuous Testing as a Service (TaaS) for web APIs?** The popularization of DevOps in recent years is fostering a shift towards the use of tools that provide continuous and autonomous building, testing and deployment. Furthermore, Software nowadays is predominantly provided as a Service (SaaS). In this scenario, we envision the need of a testing framework for web APIs that fulfills these two aspects. Ideally, the framework should offer 24/7 testing and the reporting of bugs and API coverage should be accessible as a service. Companies such as API Fortress [3] already offer a similar service, however, we aim to provide a higher level of automation, personalization, efficacy and efficiency.

## 4 OBJECTIVE

The main goal of this dissertation is to apply AI techniques for the autonomous detection of failures in web APIs, including input test data generation and output evaluation (oracle). We strongly believe that this work can improve or complement existing web API testing tools and approaches in several ways. Firstly, our proposal is the first to integrate a model-based approach with a variety of testing techniques; for example, there may be APIs where combinatorial testing is more suited, e.g. an API with a high number of parameters, and others where data perturbation may be more effective, if the API input is a complex JSON object. Secondly, most approaches focus on the generation of inputs, but that is only half of the problem; we plan to address the evaluation of outputs as well. Thirdly, the application of AI techniques will ideally allow to automate the full process, as opposed to most approaches that require human intervention. Finally, we aim to provide continuous TaaS with a much higher

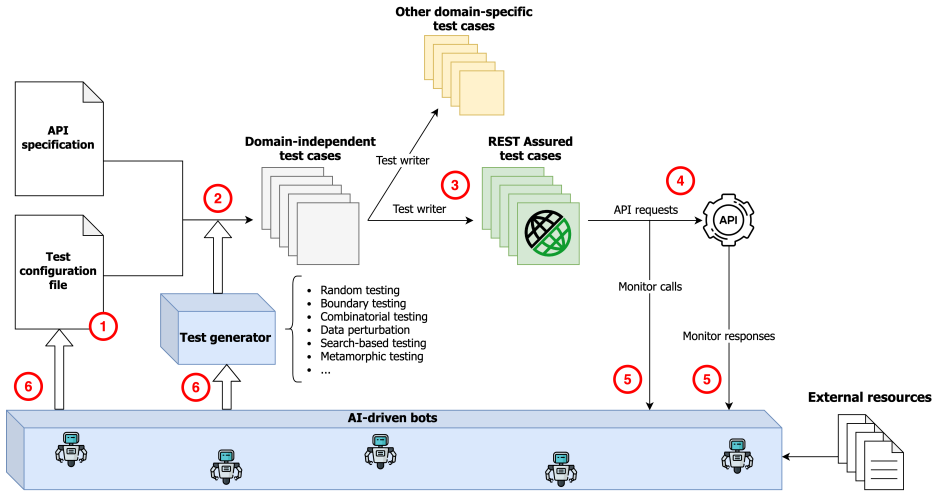


**Figure 1: High level architecture of the testing framework.**

degree of automation and personalization than the current offerings, which will open a new range of business opportunities.

## 5 WORKING PLAN AND PRELIMINARY RESULTS

We have set ourselves a long-term and ambitious goal. Therefore, we plan to divide it into four main milestones, each one closely related to one of the research questions previously posed. Each of the milestones is expected to materialize in the form of multiple publications and artifacts (e.g. tools and datasets). Figure 1 depicts the overall architecture of the testing framework we plan to implement, where we have divided the testing process into six phases. Each milestone will address one or more of these phases.

The research described in this paper started in October 2018 and is estimated to end in June 2022 (four years). At the time of writing this paper (October 2019), we find ourselves in the second year of the PhD, so we have fully addressed the first contribution and are working on the second one.

**Model-based specification-driven testing of RESTful APIs (1st year).** This contribution naturally results from answering RQ1. After some preliminary tests, we found that it is generally infeasible to create test cases only with the API specification, as it lacks key information such as authorization data (e.g. API keys). This led us to implement a model-based testing prototype [11], partly addressing steps 1 to 4 of Figure 1. Step 1 refers to the creation of a test configuration file where to include all key information needed to generate test cases. This includes authorization data, parameter values, inter-parameter dependencies [26] and parameter weights (i.e. what parameters must be tested more thoroughly). In step 2, test cases are automatically generated based on the API specification and the test configuration file, so far using random and boundary values for the input parameters. These test cases are domain-independent, i.e. they can be instantiated in any framework (step 3). In our case, we use REST Assured [10]. Finally, the test cases are run and the API responses are asserted (step 4); the framework provides a GUI for the visualization of bugs and API coverage.

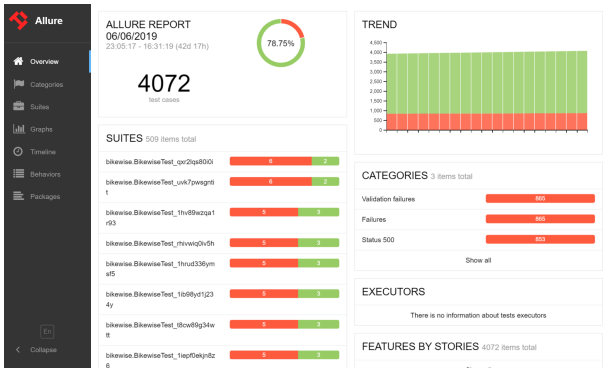


**Figure 2: GUI of the testing framework.**

Figure 2 shows a screenshot of the GUI, taken after continuously generating test cases for a real API (Bikewise)[1] during 42 days. More than 4,000 randomly generated test cases were executed, 865 of which made the system fail, showing the presence of bugs in 3 out of 4 paths in the API (errors 500 and 502).

Aiming to compare our testing framework to others, we noticed the lack in the literature of a standard model for the assessment and comparison of RESTful API testing techniques. For this reason, we created a catalogue of test coverage criteria for RESTful APIs and arranged them into multiple *coverage levels*, constituting what we call a Test Coverage Model (TCM). The TCM provides a common framework that authors can use to compare their work to others. This contribution was published at an international workshop [28].

When automatically generating test cases, we realized that many of them were invalid due to the presence of inter-parameter dependencies in some API operations, which cannot be formally specified in OAS[2]. We reviewed the state of practice on 40 real-world APIs and eventually found that this phenomenon is extremely common. We published this contribution at an international conference [26] as

[1] https://bikewise.org
[2] https://github.com/OAI/OpenAPI-Specification/issues/256

well as the resulting dataset [27]. We are now developing a domain-specific language (DSL) and a constraint programming-aided tool for the specification and automated analysis of these dependencies. This information will then be included in the test configuration file so that valid test cases can be automatically generated.

**Automatic generation of test cases (2nd year)**. This contribution seeks to answer RQ2 and is mainly concerned with phase 2 of Figure 1. The goal is to implement several techniques for the generation of input test data and oracles and evaluate them. Among others, we plan to integrate the following testing techniques in the framework: 1) search-based testing: the coverage criteria defined in our previous contribution [28] can be used as the fitness function; 2) data perturbation: we are currently defining JSON mutation operators with two purposes: a) input data can be generated by perturbing original JSON request bodies, and b) test suites can be evaluated by mutating the specification of the API, i.e. the JSON response bodies; 3) metamorphic testing: we have designed an approach for the automated generation of metamorphic relations based on the API specification and the patterns defined in a previous work from our research group [32].

**AI-driven testing of RESTful APIs (3rd year)**. This contribution seeks to answer RQ3. Once that test inputs and oracles are automatically generated, we plan to keep improving and automating other aspects of the testing process as much as possible. This refers to steps 5 and 6 in Figure 1.

In step 5, API requests and responses are monitored aiming to identify two main patterns: invariants and metamorphic relations. Invariants are request-response patterns that always hold, e.g. "when using parameter X in some operation, the response always contains the property Y". Metamorphic relations are relations between two or more API requests and responses, e.g. "when invoking some operation and then invoking the same operation using parameter X, the result of the second call must be a subset of the first one". The identification of these patterns will hopefully help detect previously unknown bugs.

In step 6, the knowledge gained by the bots is used to automate manual tasks, namely the tuning of the test configuration file and some aspects of the generation of test cases. For example, we envision that inter-parameter dependencies could be automatically inferred only by monitoring real calls to the service, based on what calls are successful and which are erroneous and why. Moreover, meaningful and realistic values for parameters could be generated by processing their description in natural language (extracted from the API documentation) or by perturbing original values from previous executions.

**Continuous Testing as a Service (TaaS) of RESTful APIs (4th year)**. This contribution seeks to answer RQ4. Once the process is fully automated, we intend to evaluate the effectiveness of the framework in a more realistic environment. To this end, we plan to select a number of real-world industrial APIs (e.g. YouTube and Spotify) and perform 24/7 testing on them with the aim of finding bugs in a completely autonomous way. Once we ascertain the validity of the framework, we aim to deploy it in a publicly accessible server and offer its capabilities as a service to other API providers, opening a new range of business opportunities.

## ACKNOWLEDGMENTS

This work has been partially supported by the European Commission (FEDER) and Spanish Government under projects BELI (TIN2015-70560-R) and HORATIO (RTI2018-101204-B-C21), and the FPU scholarship program, granted by the Spanish Ministry of Education and Vocational Training (FPU17/04077).

## REFERENCES

[1] 2007. *Simple Object Access Protocol (SOAP) Version 1.2.* https://www.w3.org/TR/soap12-part0/
[2] 2007. *Web Services Description Language (WSDL) Version 2.0.* https://www.w3.org/TR/wsdl20/
[3] 2019. *API Fortress.* https://apifortress.com
[4] 2019. Applitools. https://applitools.com
[5] 2019. *GraphQL.* https://graphql.org
[6] 2019. *OpenAPI Specification.* https://www.openapis.org
[7] 2019. *Postman.* https://getpostman.com
[8] 2019. *ProgrammableWeb.* https://www.programmableweb.com
[9] 2019. *ReadyAPI.* https://smartbear.com/product/ready-api/overview/
[10] 2019. *REST Assured.* https://rest-assured.io
[11] 2019. *RESTest.* https://github.com/isa-group/RESTest
[12] 2019. *RESTful API Modeling Language (RAML).* https://raml.org
[13] 2019. test.ai. https://test.ai
[14] A. Arcuri. 2019. RESTful API Automated Test Case Generation with EvoMaster. *ACM Trans. on Software Engineering and Methodology* 28, 1 (2019), 3.
[15] V. Atlidakis, P. Godefroid, and M. Polishchuk. 2019. RESTler: Stateful REST API Fuzzing. In *Intern. Conference on Software Engineering*. 748–758.
[16] C. Benac, L. Fredlund, A. Herranz, and J. Mariño. 2014. Jsongen: A QuickCheck Based Library for Testing JSON Web Services. In *ACM SIGPLAN Workshop on Erlang*. 33–41.
[17] M. Bozkurt, M. Harman, and Y. Hassoun. 2013. Testing and Verification in Service-Oriented Architecture: A Survey. *Software Testing, Verification and Reliability* 23, 4 (2013), 261–313.
[18] G. Canfora and M. Di Penta. 2006. Service-Oriented Architectures Testing: A Survey. In *Software Engineering*. Springer, 78–105.
[19] S. K. Chakrabarti and P. Kumar. 2009. Test-the-REST: An Approach to Testing RESTful Web-Services. In *Computation World: Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns*. 302–308.
[20] S. K. Chakrabarti and R. Rodriquez. 2010. Connectedness Testing of RESTful Web-Services. In *India Software Engineering Conference*. 143–152.
[21] H. Ed-douibi, J.L.C. Izquierdo, and J. Cabot. 2018. Automatic Generation of Test Cases for REST APIs: A Specification-Based Approach. In *IEEE 22nd Intern. Enterprise Distributed Object Computing Conference*. 181–190.
[22] J. Edstrom and E. Tilevich. 2015. Improving the Survivability of RESTful Web Applications via Declarative Fault Tolerance. *Concurrency and Computation: Practice and Experience* 27, 12 (2015), 3108–3125.
[23] T. Fertig and P. Braun. 2015. Model-Driven Testing of RESTful APIs. In *Intern. Conference on World Wide Web*. 1497–1502.
[24] R. T. Fielding. 2000. *Architectural Styles and the Design of Network-based Software Architectures.* Ph.D. Dissertation.
[25] P. Lamela, H. Li, and S. Thompson. 2013. Towards Property-Based Testing of RESTful Web Services. In *ACM SIGPLAN Workshop on Erlang*. 33–41.
[26] A. Martin-Lopez, S. Segura, and A. Ruiz-Cortés. 2019. A Catalogue of Inter-Parameter Dependencies in RESTful Web APIs. In *Intern. Conference on Service-Oriented Computing*. 399–414.
[27] A. Martin-Lopez, S. Segura, and A. Ruiz-Cortés. 2019. Inter-Parameter Dependencies in RESTful APIs [Dataset]. https://bit.ly/2wvv1m1
[28] A. Martin-Lopez, S. Segura, and A. Ruiz-Cortés. 2019. Test Coverage Criteria for RESTful Web APIs. In *ACM SIGSOFT Intern. Workshop on Automating TEST Case Design, Selection, and Evaluation*. 15–21.
[29] P. V. P. Pinheiro, A. T. Endo, and A. Simao. 2013. Model-Based Testing of RESTful Web Services Using UML Protocol State Machines. In *Brazilian Workshop on Systematic and Automated Software Testing*. 1–10.
[30] H. Reza and D. Van Gilst. 2010. A Framework for Testing RESTful Web Services. In *Intern. Conference on Information Technology: New Generations*. 216–221.
[31] L. Richardson, M. Amundsen, and S. Ruby. 2013. *RESTful Web APIs.* O'Reilly Media, Inc.
[32] S. Segura, A. Durán, J. Troya, and A. Ruiz-Cortés. 2019. Metamorphic Relation Patterns for Query-based Systems. In *Workshop on Metamorphic Testing*. 24–31.
[33] S. Segura, J.A. Parejo, J. Troya, and A. Ruiz-Cortés. 2018. Metamorphic Testing of RESTful Web APIs. *IEEE Trans. on Software Engineering* 44, 11 (2018), 1083–1099.
[34] M. Zhang, B. Marculescu, and A. Arcuri. 2019. Resource-Based Test Case Generation for RESTful Web Services. In *Genetic and Evolutionary Computation Conference*. 1426–1434.