# International Summer School on Search- and Machine Learning-based Software Engineering (SMILESENG)

June 22-24, 2022

Córdoba, Spain

Aurora Ramírez and
José Raúl Romero (Eds.)
University of Córdoba

UCOPress
Ediciones Universidad de Córdoba

# Proceedings of the International Summer School on Search- and Machine Learning-based Software Engineering (SMILESENG)

June 22-24, 2022
Córdoba, Spain

**Editors:**
Aurora Ramírez
José Raúl Romero

Proceedings of the International Summer School on Search- and Machine Learning-based Software Engineering - Córdoba: UCOPress. Editorial Universidad de Córdoba, 2022

21x29.7 cm; 74 p.; il. color.
Edited by Aurora Ramírez and José Raúl Romero

PROCEEDINGS OF THE INTERNATIONAL SUMMER SCHOOL ON SEARCH- AND MACHINE LEARNING-BASED SOFTWARE ENGINEERING

© Several authors

© Edited by: Aurora Ramírez and José Raúl Romero

© UCOPress. Editorial Universidad de Córdoba, 2022
Campus de Rabanales. Ctra. Nac. IV, km. 396, 14071 – Córdoba, Spain
`http://www.uco.es/ucopress/` — ucopress@uco.es
Phone: (+34) 957 21 21 65

Design and layout: Aurora Ramírez

**ISBN**: 978-84-9927-704-2

# Welcome Note

Optimization and search algorithms applied to software engineering led to the emergence of the area of search-based software engineering (SBSE), which aims to automate and optimize solutions to complex problems in the daily work of a software engineer. More recently, machine learning (ML) algorithms have also made it possible to explore and extract knowledge from various data sources commonly used in the software development process, such as source code hosted in repositories, developers' forums or technical documentation. Whatever the technique applied, the objective of using Artificial Intelligence (AI) to these problems is to facilitate the work of developers, analysts, testers or software engineers, improving the quality of the resulting products while reducing the costs of their production.

This International Summer School on Search- and Machine learning-based Software Engineering (SMILESENG) is the third edition that continues the path of the successful previous schools, organized by the Spanish Network of Excellence SEBASENet in 2016 and 2017. The areas evolve, so SMILESENG is opened to new ways of learning and improving the software development process. Thanks to the support of the Second SEBASENet Research Network, the School of Engineering Sciences (EPSC), and other sponsors, I am proud to be able to organize this event, aimed at undergraduates and postgraduates (especially PhD students), as well as senior researchers and professionals interested in the application of AI techniques to software engineering.

The SMILESENG scientific program includes four seminars by renowned researchers that will surely be a source of inspiration for all of us. In addition, participants will present 22 talks related to search-based software testing, machine learning to support code development, and methodological aspects, among other topics. This is an event open to the community, in which I hope we will have the opportunity to open interesting debates and hear challenging ideas about ongoing works in this area. All of this, why not, while we enjoy the magnificent city of Córdoba.

Welcome to the SMILESENG International Summer School and on behalf of all the organizing team, I wish you an enjoyable experience and see you in Córdoba!

**José Raúl Romero**
**General and Local Chair**

# Organizing Committee

| | |
|---|---|
| **General and Local Chair** | |
| José Raúl Romero | University of Córdoba, Spain |
| **General Chair** | |
| Sebastián Ventura | University of Córdoba, Spain |
| **Program Chair** | |
| Aurora Ramírez | University of Córdoba, Spain |
| **Registration and Financial Chair** | |
| Carlos García-Martínez | University of Córdoba, Spain |
| **Web Chair** | |
| Rafael Barbudo | University of Córdoba, Spain |
| **Steering Committee** | |
| José Francisco Chicano | University of Málaga, Spain |
| Inmaculada Medina-Bulo | University of Cádiz, Spain |
| José Raúl Romero | University of Córdoba, Spain |
| **Local Committee** | |
| Eduardo Almeda | University of Córdoba, Spain |
| Álvaro Espejo | University of Córdoba, Spain |
| Aurora Esteban | University of Córdoba, Spain |
| Eva Gibaja | University of Córdoba, Spain |
| José María Luna | University of Córdoba, Spain |
| María Luque | University of Córdoba, Spain |
| Antonio Rafael Moya | University of Córdoba, Spain |
| José María Moyano | University of Córdoba, Spain |
| Cristóbal Romero | University of Córdoba, Spain |
| Amelia Zafra | University of Córdoba, Spain |

**Student Volunteers**

| | |
|---|---|
| Mario Berrios | University of Córdoba, Spain |
| Aitana Delgado | University of Córdoba, Spain |
| José Manuel Flores | University of Córdoba, Spain |
| Pedro Pablo García | University of Córdoba, Spain |
| Fernando Herrera | University of Córdoba, Spain |

## Financial Support and Sponsors

ESCUELA POLITÉCNICA SUPERIOR DE CÓRDOBA
Universidad de Córdoba
EPSC

UNIVERSIDAD Đ CÓRDOBA

SEBASE.Net
RED2018-102472-T

GOBIERNO DE ESPAÑA
MINISTERIO DE CIENCIA E INNOVACIÓN

Sistedes
Sociedad de Ingeniería de Software y Tecnologías de Desarrollo de Software

# Contents

# Seminars

## Bayesian Analysis of Software Engineering Data

Robert Feldt

Chalmers University of Technology

**Abstract.** Many other scientific fields that rely heavily on analyzing empirical data, e.g. medicine, psychology, and economics, are in a sort of replication crisis. One underlying reason is inadequate statistical practices. There is reason to believe software engineering might not be much better off. In this seminar I will briefly summarize key principles for Bayesian data analysis as well as show examples of how one can apply it for analysis of software engineering data. I will also argue what the benefits are and how it can be one step towards both making our results more robust and help create chains of evidence between multiple studies.

**Biography.** Dr. Robert Feldt is a Professor of Software Engineering at Chalmers University of Technology in Gothenburg, where he is part of the Software Engineering division at the Department of Computer Science and Engineering. He is also a part-time Professor of Software Engineering at Blekinge Institute of Technology in Karlskrona, Sweden. He is co-Editor in Chief of Empirical Software Engineering (EMSE) Journal. He is interested in Software Engineering but with a particular focus on software testing, requirements engineering, psychological and social aspects as well as agile development methods/practices. He is "one of the pioneers" in the search-based software engineering field (according to an ACM Computing Survey of SBSE) and has a general interest in applying Artificial Intelligence and Machine Learning both during software development and, in general, within software systems. Based on his studies in Psychology he also tries to get more research focused on human aspects; an area we have proposed to call Behavioral Software Engineering.

## Intelligent Recommender Systems in Software Development

Davide Di Ruscio

University of L'Aquila

**Abstract.** Recommender systems are a crucial component of several online shopping systems, allowing business owners to offer personalized products to customers. Recommender systems in software engineering (RSSE) have been conceptualized on a comparable basis, i.e., they assist developers in navigating large information spaces and getting instant recommendations that are helpful to solve a particular development task. In this sense, RSSE provides developers with valuable suggestions, which may consist of different items, such as code examples, topics, third-party components, documentation, to name a few. However, developing RSSE is a complex task; technical choices must be taken to overcome issues related to several aspects, including the lack of baselines, limited data availability, decisions about the performance measures, and evaluation approaches. This seminar makes an overview of RSSE and describes the challenges that have been encountered in developing different RSSE in the context of the EU CROSSMINER project. Specific attention will be devoted to presenting the intricacies related to the development and evaluation techniques that have been employed to conceive and evaluate the CROSSMINER recommender systems. Moreover, the lessons that have been learned while working on the project will also be discussed.

**Biography.** Dr. Davide Di Ruscio is Associate Professor at the Department of Information Engineering Computer Science and Mathematics of the University of L'Aquila. His main research interests are related to several aspects of Software Engineering, Open Source Software, and Model Driven Engineering (MDE) including domain specific modeling languages, model transformation, model differencing, coupled evolution, and recommendation systems. He has published more than 140 papers in various journals, conferences and workshops on such topics. He has been co-guest editor of a number of special issues. He has been in the PC and involved in the organization of several workshops and conferences, and reviewer of many journals like IEEE Transactions on Software Engineering, Science of Computer Programming, Software and Systems Modeling, and Journal of Systems and Software. He is member of the steering committee of the International Conference on Model Transformation (ICMT), of the Software Language Engineering (SLE) conference, of the Seminar Series on Advanced Techniques & Tools for Software Evolution (SATTOSE), of the Workshop on Modeling in Software Engineering at ICSE (MiSE) and of the International Workshop on Robotics Software Engineering (RoSE). Davide is in the editorial board of the International Journal on Software and Systems Modeling (SoSyM), of IEEE Software, of the Journal of Object Technology, and of the IET Software journal.

## Testing with Fewer Resources: Toward Adaptive Approaches for Cost-effective Test Generation and Selection

Sebastiano Panichella and Christian Birchler

Zurich University of Applied Science

**Abstract.** After a very brief introduction to the basic concepts of SBST (search-based software testing), we will go into adaptive approaches to cost-effecting test generation for Java systems, which will be demonstrated in a short demo. The seminar will continue with the basics of self-driving cars development and testing, and will end with a more detailed discussion on test regression (particularly selection strategies) for self-driving cars software. A demo will also be given on the latter topic, giving the seminar a practical character.

**Biography.** Dr. Sebastiano Panichella is a passionate Computer Science Researcher at the Zurich University of Applied Science (ZHAW). He received the PhD in Computer Science from the University of Sannio (Department of Engineering) in 2014 defending the thesis entitled "Supporting Newcomers in Open Source Software Development Projects". His main research goal is to conduct industrial research, involving both industrial and academic collaborations, to sustain the Internet of Things (IoT) vision, where future "smart cities" will be characterized by millions of smart systems (e.g., cyber-physical systems such as drones, and other autonomous vehicles) connected over the internet, composed by AI-components, and/or controlled by complex embedded software implemented for the cloud. His research interests are in the domain of Software Engineering (SE), cloud computing (CC), and Data Science (DS): DevOps (e.g., Continuous Delivery, Continuous integration), Machine learning applied to SE, Software maintenance and evolution (with particular focus on Cloud, mobile, AI-based, and Cyber-physical applications), Mobile Computing. He is Review Board member of the EMSE journal.

**Biography.** Mr. Christian Birchler is a Research Assistant at the Zurich University of Applied Sciences where he is working on the EU Horizon project COSMOS ("DevOps for Complex Cyber-physical Systems"). He studied at the University of Zurich Software Systems with Applied Probability and Statistics as a minor subject. Currently, he is pursuing a master's degree in Software Systems with Data Science as a minor subject at the University of Zurich. During his studies, he mainly focused on software testing. His research interests are search-based software testing and fuzzing. In his ongoing work, he is investigating the area of software testing and testing in virtual environments combined with the development of tools to solve the problems in his research area. A prominent example is SDC-Scissor (github.com/ChristianBirchler/sdc-scissor), which is a tool that leverages the test selection part of the regression testing process for self-driving cars software. His vision is to provide a regression testing framework that also includes test prioritization for simulation-based testing of cyber-physical systems.

## Data Mining Algorithms Using/Used-by Optimizers: a DUO Approach to Software Engineering

Leandro Minku

University of Birmingham

**Abstract.** The fields of Software Analytics and Search-Based Software Engineering have evolved mostly as separate fields over the past decades. Both have achieved a great level of maturity, finding their way into Software Engineering Practice. However, their achievements are limited by their isolated focus on either data mining / machine learning or search-based optimization. What could Software Analytics achieve when using search-based optimization? And what could Search-Based Software Engineering achieve when using data mining? This talk will discuss recent advancements and future directions of the emerging field of DUO — Data mining Using/Used-by Optimizers for empirical studies in software engineering.

**Biography.** Dr. Leandro L. Minku is an Associate Professor at the School of Computer Science, University of Birmingham (UK). Prior to that, he was a Lecturer at the University of Leicester (UK), and a Research Fellow at the University of Birmingham (UK). He received the PhD degree in Computer Science from the University of Birmingham (UK) in 2010. Dr. Minku's main research interests include machine learning for software engineering, machine learning for non-stationary environments / data stream mining, class imbalance learning and search-based software engineering. Among other roles, Dr. Minku is Associate Editor-in-Chief for Neurocomputing, and Associate Editor for IEEE Transactions on Neural Networks and Learning Systems, Empirical Software Engineering journal and Journal of Systems and Software. He was the general chair for the International Conference on Predictive Models and Data Analytics in Software Engineering (PROMISE 2019-2020), co-chair for the Artifacts Evaluation Track of the International Conference on Software Engineering (ICSE 2020), and program committee member for prestigious conferences both in the fields of machine learning and software engineering, such as IJCAI, AAAI and ICSE.

# Student talks

# Sustainability in Open Source: Bots to the Rescue

Adem Ait
IN3 - UOC
Barcelona, Spain
aait_mimoune@uoc.edu

Javier Luis Cánovas Izquierdo
IN3 - UOC
Barcelona, Spain
jcanovasi@uoc.edu

Jordi Cabot
IN3 - UOC, ICREA
Barcelona, Spain
jordi.cabot@icrea.cat

***Abstract*—Nowadays, most critical software is built as Open Source Software (OSS) or heavily relies on it. Despite OSS bringing many benefits -better quality, more flexibility or lower cost- it suffers from problems such as the "tragedy of the commons": everybody uses Open Source, but very few contribute back. Moreover, a key piece to the sustainability of OSS is the non-coding contributions, helping to maintain and report different threats to the project development. Usually not acknowledged, the OSS community is crucial for the continuous maintenance of OSS projects. Thereby, we propose the use of a self-guiding swarm of smart software bots to assist project owners and developers but also occasional contributors and community members in all their software-related tasks. Bots will be trained using a variety of artificial intelligence techniques, including machine learning models derived from a curated collection of software project data in social code hosting platforms like GitHub. Thus, the source of information is extracted as collaboration graphs, which help to represent the relationship between the project contributors. Using machine learning techniques, we aim to learn from these collaboration graphs to further configuration and adaptation of the bots. With the ability to self-adapt, bots can answer to the threats that an OSS project is facing. Furthermore, we intend to create a software framework to model, generate, personalize, combine and coordinate smart software bots to help in all phases of software development and maintenance.**

## I. Introduction

The Open Source Initiative[1] describes the Open Source Software (OSS) development process as "a method that harnesses the power of distributed peer review and transparency... The promise of OSS is better quality, higher reliability, more flexibility and lower cost". This promise is achieved by developing OSS in a collaborative manner, where the whole community involved in an OSS project (i.e., project owners, external contributors and end-users) can participate.

A key characteristic of OSS projects is their global nature and reach. Modern online code hosting platforms such as GitHub have enabled geographically dispersed developers to launch, maintain, and deploy OSS projects. These platforms provide source control management systems to track changes in the code, but also include collaboration features like issue-tracking systems or the pull-request mechanism [2] to manage the different development tasks in the project. These platforms are implicitly standardizing a development process based on open and free access to the code and the development tasks, which aim at promoting a collaboration model among the community of developers and users of the software.

In theory, this collaboration model should be able to quickly evolve the software codebase to respond to the users' needs thanks to this higher community involvement. Unfortunately, this is hardly ever what we see in practice. Using the metaphor presented in the well-known article titled "The Cathedral and the Bazaar" [3], what we typically find is an attempt to reach a "Bazaar" model but poorly executed, where only few community members are willing to participate and those who do want to contribute lack the required support to collaborate effectively. This situation is known as the "tragedy of the commons": everybody wants to benefit from the software, but they all hope others will chip in. As a result, there is a grossly disproportionate imbalance between those consuming the software and those able to participate in building the software, thus creating important sustainability issues.

Existing solutions to correct this situation are mostly focused on code aspects of OSS projects, which is not enough to enable a real and effective collaboration model. OSS projects are much more than their codebase: they are shaped according to their community. For an OSS project to succeed, we need to consider OSS as a global social endeavor where community and source code dimensions must be first-class elements in the evaluation, monitoring and progress of any project to guarantee its long-term sustainability.

To tackle this situation we propose to develop a bot-driven framework where a self-guiding swarm of smart software bots would assist projects owners, developers (including occasional contributors) and community members in all their OSS developing tasks going all the way up from simple tasks like welcoming new project members or helping them to write useful bug reports to automatically reject/fix contributions that violate the project guidelines or code of conduct. The application of bots for OSS can leverage on the open availability of project assets (i.e., source code but also discussions, issues or comments), which will be used to train the bots using a variety of artificial intelligence techniques, including Machine Learning (ML) models derived from a curated collection of software project data in code hosting platforms like GitHub.

In the rest of this paper, we present a proposal for our approach, describing a tentative architectural description and discussing the main challenges to address.

[1]https://opensource.org/

## II. Proposal & Challenges

Transforming OSS engineering into a successful and sustainable process requires being able to enroll as much additional help as possible to manage the software and the surrounding community. We propose to leverage on the help of a swarm of smart software bots to tackle the diverse tasks required to address the sustainability challenges on OSS.

Figure 1 shows the process we devised to create the bots. As can be seen, we define a process composed of four phases: (1) data collection, where project assets are gathered and curated (code but also other collaboration assets such as issues, discussions or comments, among others); (2) graph generation and metric calculation, where collaboration graphs will be generated and used to analyze OSS projects; (3) bot configuration and training, where graphs and metrics will be employed to train bots to perform specific tasks; and (4) bot deployment, where bots will be deployed in the OSS project and used to perform specific tasks.

We have identified several challenges to develop our proposal, which we present in the following.

**Mining collaboration graphs.** Our proposal leverages on collaboration graphs to represent OSS projects, thus enabling the defining of metrics to evaluate relationships between assets of the project. A collaboration graph is a directed graph where nodes represent project assets (e.g., commits, contributors, issues, etc.) and edges represent relationships between them (e.g., the author of a comment, the contributor of a commit, the issue that was assigned to a contributor, etc.). Figure 2 shows an example of a collaboration graph. However, the creation of collaboration graphs is not an easy task, as the extraction and curation of the data recovered from repositories from online code hosting platforms usually require an intricate process to digest the data. Existing solutions such as Cauldron[2] or Augur[3] are able to extract tabulated data from repositories, which could be used to generate our graph. To the best of our knowledge, only SourceCred[4] is able to generate graphs from repositories, but the extracted data is very limited.

**Graph-specific metrics.** Once the data is extracted in the form of a collaboration graph, the next challenge is to extract significant metrics to describe behaviors, patterns, or identification of roles. The definition and calculation of graph-specific metrics has usually been covered in the field of Social Network Analysis, which investigates social structures through the use of networks and graph theory (e.g., [1], [4]). In our framework, we propose to explore the application of graph-specific metrics to evaluate the collaboration in OSS projects.

**Graph ML Methods.** Recently, Graph Neural Networks (GNN) have received a lot of attention due to its ability to analyze graph structural data, which is difficult to analyze as it does not exist in a Euclidean space, does not have a fixed form and usually is hard to visualize for human interpretation.

[2]https://cauldron.io/
[3]https://github.com/chaoss/augur
[4]https://sourcecred.io/

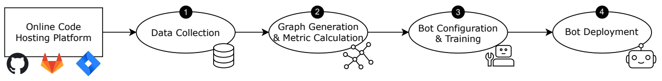

Online Code Hosting Platform
1
Data Collection
2
Graph Generation & Metric Calculation
3
Bot Configuration & Training
4
Bot Deployment

Fig. 1. Process to create the bots.

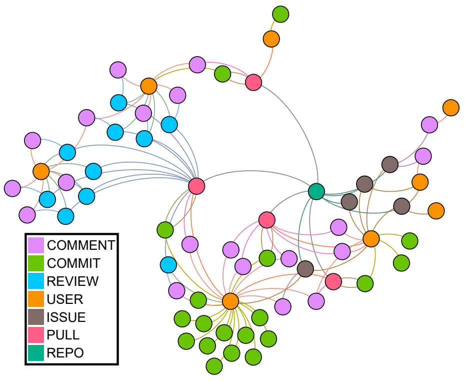


Fig. 2. Example of a collaboration graph of a GitHub repository.

A GNN is a neural network that can directly be applied to graphs and provides a convenient way for node-level, edge-level, and graph-level prediction tasks. We propose to use GNNs to analyze collaboration in our graphs and to configure the behavior of bots for OSS projects, as we describe below.

**Bot definition and configuration.** Our approach aims at providing bots specifically tailored for OSS projects. We propose to define languages to build a smart bots infrastructure able to monitor OSS projects, define and enforce rules, and communicate with the user. Furthermore, to support the deployment of a swarm of bots, we would need to define mechanisms to enable effective bot collaboration protocols.

## III. Conclusion

In this paper we have described our proposal for the development of a bot-driven framework to tackle the diverse tasks required to address the sustainability challenges on OSS. The framework leverages on mining software repository techniques to extract collaboration graphs from the repositories of OSS projects, which are then used to configure and train bots for OSS projects. As next steps, we plan to explore the different challenges commented in Section II.

## Acknowledgment

This work has been partially funded by the Spanish government (LOCOSS project - PID2020-114615RB-I00).

## References

[1] Mohammad Y. Allaho and Wang-Chien Lee. Analyzing the Social Ties and Structure of Contributors in Open Source Software Community. pages 56–60, 2013.
[2] Georgios Gousios, Martin Pinzger, and Arie van Deursen. An Exploratory Study of the Pull-based Software Development Model. In *Int. Conf. on Software Engineering*, pages 345–355, 2014.
[3] Eric S. Raymond and Tim O'Reilly. *The Cathedral and the Bazaar*. 1999.
[4] Wen Zhang, Ye Yang, and Qing Wang. An Empirical Study on Identifying Core Developers Using Network Analysis. In *Int. Workshop on Evidential assessment of software technologies*, pages 43–48, 12.

# Automated Generation of Test Oracles for RESTful APIs

Juan C. Alonso
SCORE Lab, I3US Institute,
University of Seville,
Seville, Spain
Email: javalenzuela@us.es

Sergio Segura
SCORE Lab, I3US Institute,
University of Seville,
Seville, Spain
Email: sergiosegura@us.es

Antonio Ruiz-Cortés
SCORE Lab, I3US Institute,
University of Seville,
Seville, Spain
Email: aruiz@us.es

***Abstract*—Web APIs following the REST architectural style (frequently known as RESTful APIs) have become the de-facto standard for web integration. In recent years, a large number of tools for automatically testing this type of API have emerged. However, all these tools fall short when it comes to their fault-detection capabilities, which are limited to unexpected failures (i.e., 5XX code responses) and disconformities with the API specification. This article describes our ongoing work for automatically generating test oracles for RESTful APIs. Specifically, we propose to automatically infer likely invariants from sets of inputs and outputs that can later be used as test oracles by leveraging an extended version of Daikon, a tool that detects likely invariants by processing a program execution. A preliminary evaluation with 8 operations from 6 industrial APIs shows the effectiveness of our approach for automatically generating test oracles, detecting reproducible faults in two of them (GitHub and OMDb).**

## I. Introduction

RESTful APIs are the cornerstone of software integration, allowing systems to interact with each other over the network by exchanging messages in JSON or XML format through the HTTP protocol. Web services usually provide RESTful APIs for different clients to access their functionality. This is one of the main reasons why testing these systems is vital, since a fault in an API could compromise hundreds or thousands of other systems consuming it.

RESTful APIs are commonly described using languages such as the OpenAPI Specification (OAS) [1], which provides a machine-readable description of the API functionality that is used by different tools to automatically generate test cases [2]. All these tools are limited by the types of errors they can detect, such as disconformities with the OAS specification and server errors.

An apparently successful response returned by an API (i.e., a 2XX status code that conforms to the API specification) does not guarantee that the system is fulfilling its intended functionality of behavior. This is a classical problem in Software testing known as the *oracle problem* [3], that can be expressed as the challenge of, given an input for a system, distinguish the expected behavior from a potentially incorrect one. For example, when performing a search for songs in Spotify establishing a maximum number of results to return (`limit` parameter), the size of property of the response body whose value is an array of songs (`items`) should be less or equal than the value set for this parameter (`input.limit >= size(return.items)`). Although it is possible to generate these oracles manually, this is a time-consuming task that requires domain knowledge.

An invariant is a property that holds at a certain point or points of the execution of a program, such as its input parameters and responses in the context of black box testing of RESTful APIs. Currently, there are several systems available for the automated detection of likely invariants, with Daikon [4] being one of the most popular. Daikon detects likely invariants by processing an instrumented version of a program, this instrumentation process is performed by an instrumenter, a software that converts a program structure into a format that can be analyzed by Daikon. There are several Daikon instrumenters available, most of them are designed for specific programming languages (i.e., they detect invariants on white box contexts) such as Java or Perl.

In this article, we propose an approach for automatically generating test oracles for RESTful APIs from a set of valid API requests by modelling the generation of oracles as a problem of extracting likely invariants. For this purpose, we created an instrumenter that takes as input an OAS specification and a set of test cases and returns a set of files that can be used as inputs for Daikon (i.e., it works on a black box context). A preliminary evaluation with a set of 8 operations from 6 commercial APIs shows the potential of this approach for automatically generating hundreds of test oracles, detecting real errors in two of the systems under test, namely GitHub and OMDb.

## II. Automated generation of test oracles for RESTful APIs

This section describes our approach for the automated generation of test oracles for RESTful APIs from an OAS specification and a set of test cases (i.e., values of the input parameters and the response body). Our instrumenter receives these two files as input, returning a `decls` file that describes the structure of the program (in our case, the input and the possible outputs), and a `dtrace` file specifying, for each test composing the test suite, the values assigned to each part of the structure defined by the `decls` file.

Specifically, we have modified Daikon by adding a total of 22 new invariants and suppressing 25 invariants that do would

TABLE I
ORACLES GENERATED. TP=TRUE POSITIVES, FP=FALSE POSITIVES

| API | Operation | # oracles | Precision (%) | TP | FP | Inconsistency/Bug |
|---|---|---|---|---|---|---|
| Amadeus Hotel | Find hotel offers | 93 | 60.2 | 56 | 37 | 0 |
| GitHub | List organization repositories | 106 | 70.1 | 68 | 29 | 9 |
| OMDb | By ID or Title | 20 | 70 | 14 | 6 | 0 |
| OMDb | By Search | 7 | 100 | 5 | 0 | 2 |
| Spotify | Create Playlist | 28 | 100 | 28 | 0 | 0 |
| Spotify | Get Album tracks | 51 | 76.5 | 39 | 12 | 0 |
| Yelp | Search businesses | 25 | 32 | 8 | 17 | 0 |
| YouTube | List videos | 171 | 59.1 | 101 | 70 | 0 |
| **Total** | | 501 | 65.1 | 319 | 171 | 11 |

not reveal any relevant information in our current context, resulting in the generation of redundant information or false positives. These new invariants are based on a previous publication of the authors in which an evaluation was performed on a dataset of 48 real world APIs [5]. This version of Daikon supports a total of 140 invariants that can be classified into one of the following categories:

- **Arithmetic relationships.** They specify comparisons between the values of numeric properties. For example, when searching for albums on Spotify, the track number of a song must be greater than or equal to 1 (`return.track_number >= 1`).
- **Array properties:** These invariants indicate that an array has certain characteristics. For example, when searching for hotels by id in Amadeus, the id of each hotel returned must be contained in the array of ids used as the input parameter (`return.hotelId in input.hotelIds[]`). Also, the size of the array property containing the results (`data`) must be less than or equal to the size of the list of ids used as a parameter (`size(input.hotelIds[]) >= size(return.data[])`).
- **Specific values:** They specify that a property always has a fixed value or set of values. For example, in the GitHub API, a repository can be public or private (`return.visibility one of "private", "public"`).
- **Specific formatting:** These invariants indicate that a string field always follows a specific format, such as URLs, dates or emails. For example, the OMDb's text field "Poster" must always be of type URL (`return.Poster is Url`).

## III. PRELIMINARY EVALUATION

For our evaluation, we selected a total of 8 operations from 6 industrial RESTful APIs. For each one of these operations, we automatically generated 50 valid API requests (2XX codes) using the black box framework for automated testing of RESTful APIs RESTest [6]. These requests were used as inputs for our approach, resulting in a set of likely invariants (oracles) for the operation.

These invariants are manually classified as true positives, false positives or as invariants that reveal the existence of a bug or inconsistency. The results of our evaluation (Table I) show the potential of our approach for automatically generating oracles for complex real-world systems, achieving a total precision of 65.1% and detecting errors and inconsistencies in the documentation and implementation of systems with millions of users such as GitHub or OMDb.

In the GitHub API, our proposal automatically detected that one of the fields in the response, `template_repository`, was not present in any of the repositories returned, even in cases in which they had a template repository. This bug has been confirmed by the API developers, who have created an internal issue to update the documentation.

According to their official documentation, "By Search" operation of the OMDb API allows to search for titles filtered by type ("movie", "series" or "episode"). However, our proposal detected not only that the API never returns results of type "episode", but that it returns results of a fourth type not specified in the documentation ("game"), which can also be used as the value of the parameter used to filter the search.

## IV. CONCLUSION

Our future work includes using the generated oracles for the automatic creation of assertions to evaluate the validity of API responses, as well as developing a method to prioritise among the generated oracles and detect false positives.

## REFERENCES

[1] "OpenAPI Specification," https://www.openapis.org, accessed May 2022.
[2] S. S. Myeongsoo Kim, Qi Xin and A. Orso, "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*, 2022.
[3] E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, and S. Yoo, "The oracle problem in software testing: A survey," *IEEE transactions on software engineering*, vol. 41, no. 5, pp. 507–525, 2014.
[4] M. D. Ernst, J. H. Perkins, P. J. Guo, S. McCamant, C. Pacheco, M. S. Tschantz, and C. Xiao, "The daikon system for dynamic detection of likely invariants," *Science of Computer Programming*, vol. 69, no. 1, pp. 35–45, 2007, special issue on Experimental Software and Toolkits. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S016764230700161X
[5] J. C. Alonso, A. Martin-Lopez, S. Segura, J. M. Garcia, and A. Ruiz-Cortes, "ARTE: Automated Generation of Realistic Test Inputs for Web APIs," *IEEE Transactions on Software Engineering*, 2022.
[6] A. Martin-Lopez, S. Segura, and A. Ruiz-Cortés, "RESTest: Black-Box Constraint-Based Testing of RESTful Web APIs," in *International Conference on Service-Oriented Computing*, 2020, pp. 459–475.

# Resource Optimization in End-to-End Testing

Cristian Augusto
Department of Computing
University of Oviedo
Gijón, Spain
augustocristian@uniovi.es

Jesus Morán
Department of Computing
University of Oviedo
Gijón, Spain
moranjesus@uniovi.es

Claudio de la Riva
Department of Computing
University of Oviedo
Gijón, Spain
claudio@uniovi.es

Javier Tuya
Department of Computing
University of Oviedo
Gijón, Spain
tuya@uniovi.es

***Abstract*—End-to-end (E2E) test suite execution is expensive due to the number of complex resources required. When E2E test suites are executed frequently into a continuous integration system, the total amount of resources required may be prohibitive, moreover when the tests are run in the Cloud with different billing strategies. Traditional techniques to optimize the test suites, such as test suite reduction, minimization, or prioritization, are limited in E2E due to the fact that reordering or selecting a subset of test cases also requires deploying the same expensive system. The current Ph.D. thesis aims to achieve an efficient E2E test execution for large systems in the Cloud. This is done through a smart characterization of the resources required by the test cases, grouping and scheduling them according to their resource usage to avoid unnecessary redeployments and reduce execution time, and finally, executing them into a combination of Cloud infrastructure (i.e., containers, virtual machines, and services) to optimize the costs employed in executing the test suite. Based on the scheduled test cases, we elaborate a cost model for selecting the most cost-effective infrastructure of those available in the Cloud, considering both the cost of the resources required by the test cases and the oversubscription cost (cost incurred in infrastructure contracted and not used during the test suite execution).**

## I. Introduction

The execution of End-to-end (E2E) test suites is costly in resource terms due to the expensive and complex systems required. When the test suites are put into a continuous integration environment executed with every repository change, the huge number of resources required can be prohibitive for some companies, particularly when they are run in the Cloud with pay-per-usage billing. The traditional approaches used for optimizing test suites, such as test prioritization, minimization, or reduction, are limited in E2E because selecting a subset or executing the test cases in a fixed order to preserve the effectiveness, may require fewer but also expensive resources. Moving E2E testing to the Cloud brings the opportunity for a better cost but is challenging because of the number of different infrastructures with different costs available to deploy the same resource. Often, the selected infrastructure combination does not match the test suite requirements, which leads to oversubscription. Oversubscription is the difference between the infrastructure contracted and used and causes that part of the cost is not invested in testing tasks, incurring in an extra cost budget [1]. In this thesis, we aim to achieve efficient E2E test suite execution of large systems in the Cloud. For this purpose, we focus on the resources: the physical, logical, or computational entities that are required by the test cases during their execution. We elaborate an orchestration process that, through the information provided by the characterization, schedules the test cases to reduce time and unnecessary resource redeployments and execute the test suite into a combination of Cloud infrastructure. To choose the most cost-effective combination we elaborate a model which considers the cost of the (contracted) infrastructure, as well as the cost of the resources required by the test cases and the cost of oversubscription during the execution.

## II. Research Hypothesis

Working in the Cloud has become complex and heterogeneous and with it, selecting the most suitable alternative to carrying out the E2E testing, has become increasingly more challenging. The organization and management of the E2E test cases could derive from different execution times and, depending on how they are deployed in the Cloud, the costs might change. To determine the specific objectives of the research we consider the following four hypotheses:

**H1:** The execution of End-to-End (E2E) test suites can require large amounts of physical-logical resources, limiting the effectiveness of state-of-the-art techniques such as test case minimization, reduction, or parallelization.

**H2:** The scheduling of E2E test suites can be optimized by analyzing the dependencies between test cases and looking for concurrency in the use of resources (e.g., web servers, databases, hardware devices, and files, among others).

**H3:** A smart resource characterization, complemented by grouping and scheduling according to the resources used by the test cases, could save time and avoid unnecessary redeployments in the execution of the test suite.

**H4:** Considering the cost of the resources and oversubscription during the deployment of the test suite in the Cloud could help to select the most cost-effective combination among the available infrastructure.

## III. Objectives of the Work

The general objective of this thesis is to optimize end-to-end test suite execution. This objective is particularized in the following sub-objectives:

- **O1** Analyze the challenges and issues arising from the execution of the E2E Test Suites on large software developments in the Cloud.

**O2** Characterize the resources employed during the E2E testing by considering different attributes and how the resources are used by the test cases.

**O3** Improve the E2E test suite execution by considering: (1) relationships between resources, (2) how test cases access the resources, and (3) the resource attributes.

**O4** Select a cost-effective infrastructure to execute the test suite in the Cloud, considering both the cost of the resources employed in E2E and oversubscription.

**O5** Experimentation and validation of the approach, executing test suites of real-world scenarios.

## IV. Proposed Methodology

The methodologies used to achieve the prior objectives are the following:

1) **Review the state of the art and literature:** we intend to review the state-of-the-art works such as test suite minimization, reduction, prioritization, cloud economics, or cost models in the cloud, among others.
2) **Research-action:** we are collaborating with the *Institute of Information Science and Technologies "Alessandro Faedo"* which provides us with real-world problems faced in European projects such as *Elastest* [2]. Specifically, we intend to validate the research results with several test suites in the industrial field.
3) **Incremental development:** we develop the support tool based on agile methodologies [3] . Each new feature added is a research result according to its relevance

## V. First Results

To accomplish the second objective (**O2**), we carried out a characterization of the resources employed in the E2E testing, consisting of several attributes that represent the different resource features and their relationship with the test cases (e.g., available resources, access mode performed by test cases, or if the resource is shared). We propose a four-phase orchestration process called **RETORCH** (acronym of ***R**esource aware **E**nd-to-end **T**est **ORCH**estation*) to accomplish the first three objectives (**O1**, **O2**, and **O3**). **RETORCH** was published at the *QUATIC19* conference [4] and extended to the *Software Quality Journal* [5]. The work was also presented in the *ACM Research competition* in the *ICSE20* [6] and won an award in the *5th edition of the SISTEDES-Everis Awards* [7]. **RETORCH** uses the information from the characterization (***resource identification***) to generate sets of test cases with compatible usage of resources (***grouping***). The sets are split into subsets and scheduled sequentially or in parallel to reduce the execution time and avoid unnecessary redeployments (***scheduling***) and are finally deployed into a continuous integration environment (***orchestration***). The approach was validated (**O5**) with a real-world example of an educational application called *FullTeaching* [8], achieving reductions in the execution time (*61% less than the non-orchestrated test suite*) and fewer resources employed (*due to resource sharing between test cases*). To accomplish the first and fourth objectives (**O1** and **O4**), we develop a cost model focused on the cost of the resources employed in E2E test suites. The model estimates the cost invested in executing the test suite and the oversubscription cost. These two costs with the infrastructure cost (contracted) support the tester's decision-making to choose the most cost-effective infrastructure among those available in the Cloud. This model has also been submitted to the *JISBD22* [9] conference.

## VI. Conclusions and Future Work

The current thesis addresses the upcoming issue of optimizing the execution of E2E test suites in the Cloud. Its first results have proven that using a smart characterization of the resources employed on end-to-end test suites, grouping the test cases according to the resource usage, scheduling, and orchestrating them, savings in terms of resources and time can be achieved. The cost model that considers both the cost of executing the test suite and the cost incurred in oversubscription is a work in progress and we expect that it will help in selecting the most cost-effective Cloud infrastructure and obtain a better execution cost. As future work, we want to validate **RETORCH** in more real-world end-to-end test suites. We are exploring how a smarter cost model could improve the efficient E2E test execution. Specifically, we aim to integrate the cost model in an infrastructure advisor engine that analyses all three costs. The purpose of the advisor is to make suggestions about Cloud infrastructure changes that lead to a more cost-effective test suite execution (e.g., new infrastructures available, or changes in those selected that reduce one of the costs).

## Ackowledgements

The authors would like to thank **Dra. Antonia Bertolino** (*ISTI-CNR, Pisa Italia*) for her contributions to the line of research. This work was supported in part by the *Spanish Ministry of Economy and digital transformation under **TestBUS (PID2019-105455GB-C32)*** and *SEBASENet 2.0 **(RED2018-102472-T)***".

## References

[1] K. Inçki, I. Ari, and H. Sözer, "A survey of software testing in the cloud," *Proceedings of the 2012 IEEE 6th International Conference on Software Security and Reliability Companion, SERE-C 2012*, pp. 18–23, 2012.

[2] URJC, FOKUS, TUB, INSTI-CNR, IMDEA, and ATOS, "Elastest." [Online]. Available: https://elastest.eu/

[3] P. Deemer, G. Benefield, C. Larman, and B. Vodde, *The Scrum Primer: A Lightweight Guide to the Theory and Practice of Scrum.* InfoQ, 2012. [Online]. Available: www.odd-e.com

[4] C. Augusto, J. Morán, A. Bertolino, C. de la Riva, and J. Tuya, "Retorch: Resource-aware end-to-end test orchestration," *Communications in Computer and Information Science*, vol. 1010, pp. 297–310, 9 2019.

[5] ——, "Retorch: an approach for resource-aware orchestration of end-to-end test cases," *Software Quality Journal*, 2020.

[6] C. Augusto, "Efficient test execution in end to end testing," in *Proceedings - 2020 ACM/IEEE 42nd International Conference on Software Engineering: Companion, ICSE-Companion 2020*, 2020, pp. 152–154.

[7] C. Augusto and C. de la Riva, "Optimización de recursos en pruebas de sistema," *5th edition of the SISTEDES-Everis Award*, 2021.

[8] URJC and P. F. Pérez, "Fullteaching - elastest repository," 2019. [Online]. Available: https://github.com/elastest/full-teaching

[9] C. Augusto, J. Morán, C. de la Riva, and J. Tuya, "Modelo de costes para el despliegue de pruebas e2e en entornos cloud," 2022, unpublished Manuscript.

# Third-party Library Recommendations for Python Developers using Software Analytics Techniques

Pedro P. García-Pozo
Dept. Computer Science
University of Córdoba
i82gapop@uco.es

Aurora Ramírez
Dept. Computer Science
University of Córdoba
aramirez@uco.es

José Raúl Romero
Dept. Computer Science
University of Córdoba
jrromero@uco.es

***Abstract*—This talk provides an overview of our ongoing research into the design of intelligent assistants to support Python developers. In the context of a bachelor thesis, we are taking the first steps towards this long-term research objective. In this sense, this short paper presents the motivation and research objectives of our work, as well as our first results focused on the analysis of the Python library ecosystem using software analytics techniques.**

## I. Introduction

Python has become one of the most widely used programming languages among developers due to its low learning curve, its portability and the large amount of available resources within the community. The availability of third-party libraries is clearly one of the key features of Python, which promotes code reuse while reducing development effort. PyPi, one of the package managers for Python, currently hosts more than 375,000 projects.[1] In 2016, the Python ecosystem was already recognized as one of the most extensive and with higher growth prospects [1].

In such a vast and dynamic ecosystem, selecting the most suitable third-party library becomes a hard task. Several libraries might meet the functional requirements, so programmers need to consider other factors like its development support, dependencies and compatibility with other libraries, etc. Furthermore, they should decide how the library functionalities are better integrated in their current program, and whether it actually provides better performance than their own code.

The problem of library recommendation has been studied in the recent literature for Java systems [2], [3], [4]. However, the recommendations are mostly based on the idea of finding similar projects to the one under evaluation, then choosing the libraries that appear in the related projects but have not been used in the new one. These recommender systems explore a large set of code repositories, using collaborative filtering, pattern mining or clustering to discover similarities among them. Recently, some authors have focused on how to support library migration, taking a new temporal perspective of the problem by means of deep learning [5].

In this bachelor thesis, we want to take the first steps towards providing intelligent recommendations about third-party libraries oriented to Python developers. Our research vision is that current recommender systems do not exploit all the potential knowledge hidden in software repositories, and still require additional steps to support developers in the effective integration of the recommended libraries.

[1]https://pypi.org/ (Accessed: 31/05/2022)

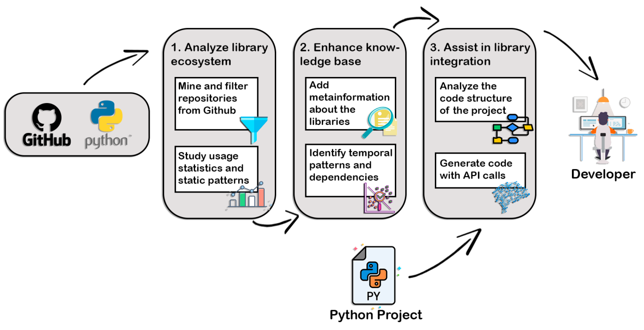
## II. Research objectives

In the long-term, we have identified three research objectives towards the design of more effective intelligent development assistants specifically oriented to Python:

1) Analyze the use of Python library in repositories to extract hidden knowledge to make recommendations.
2) Enhance the code knowledge base to be able to adapt the recommendations to the project context.
3) Assist the developers regarding how their code should be combined or replaced by API calls to the library.

## III. Overview of the approach

Figure 1 shows a high-level view of the proposed approach to develop an intelligent assistant. We first need to analyze the Python library ecosystem, adopting mining software repositories (MSR) best practices to extract data about library usage by a large number of repositories. Filtering and statistically studying the collected dataset is necessary at this stage to ensure that the information is representative and useful to perform next steps. In the second phase, we will enhance the knowledge base with additional information from the selected set of libraries (popularity, version compatibility or update). Library usage trends, e.g., whether some libraries are replaced in favor of others or they become obsolete, will be analyzed using temporal pattern mining techniques over the commit histories. Also, we will study dependencies among libraries to discover patterns of libraries frequently used together, or groups of libraries with related functionalities. Rule mining and clustering techniques will be applied to such purpose. As a final step, we will make use of all the available information to help developers integrate the library in his/her project. This implies the study of the project to detect pieces of code likely to be changed by API calls to the library. Understanding how other repositories use specific libraries, and whether the current code has similar dependencies is necessary to chose a library satisfying the developer's needs and expertise. For this step, we hypothesize that code structural analysis can be combined with classification rules learned from other repositories as done



GitHub
python
1. Analyze library ecosystem
Mine and filter repositories from Github
Study usage statistics and static patterns
2. Enhance knowledge base
Add metainformation about the libraries
Identify temporal patterns and dependencies
3. Assist in library integration
Analyze the code structure of the project
Generate code with API calls
Developer
PY
Python Project

Fig. 1. Proposed approach to build intelligent assistants for Python developers.

to detect design pattern implementations [6]. . Finally, we seek to explore neural code transformers to assist the developer in the effective integration of the library functionalities, as they have shown great potential in code generation tasks like code completion [7].

## IV. First Results

We are currently working on the first research objective, using MSR and pattern mining techniques to extract knowledge about the usage of Python libraries in GitHub. Following Vidoni's guidelines for conducting MSR studies [8], we have sampled public repositories[2] using filters to ensure that the selected repositories are relevant and active. The following criteria were established: minimum number of commits (1,000), minimum number of stars (100), and update (last commit after 1st January 2021). As a result, we have obtained a list of 3,347 repositories. PyGitHub[3] and Requirements Parser[4] were used to access the repository content and extract the libraries specified in the `requirements.txt` file, respectively. Our procedure returned 802 repositories with a valid file, resulting in a set of 3,330 different libraries in use.

An initial statistical analysis reveals some interesting insights. The maximum number of libraries used by a single repository is 232, with a median equal to 8. The average (15.5) and standard deviation (24.6) clearly indicate that the usage of libraries across repositories presents a skewed distribution: many repositories use a small number of libraries, and only a small sample of repositories import many of them. Even though the initial number of libraries could be considered high, we have found that more than half of them only appear in one repository. This seems to suggest that some libraries were conceived by the repository contributors without a clear intention of being reused in other projects. Only 32 of the 3,300 libraries (0.97%) appear in 50 or more repositories, although it is worth noting that 10 of them (`requests`, `numpy`, `sphinx`, `sphinx`, `six`, `pyyaml`, `scipy`, `python-dateutil`, `jinja2`, `matplotlib` and `pytz`) are included in 100 repositories or more.

To discover library pattern usages, we have applied a pattern mining algorithm called DCI_Closed [9] that returns subsets of libraries frequently appearing together. Notice that for this part of the study we keep only those libraries appearing in two or more repositories, i.e., 1,237 libraries in 787 repositories. Despite the fact that few repositories contain large sets of libraries, we have found pairs of libraries that appear together in between 10% and 14% of the repositories. Some frequent combinations are: {`numpy`, `scipy`}, {`numpy`, `matplotlib`}, {`requests`, `six`} and {`requests`, `pyyaml`}.

[2]For this step we used GitHub Search: https://seart-ghs.si.usi.ch/

[3]https://pygithub.readthedocs.io/ (Accessed: 31/05/2022)

[4]https://requirements-parser.readthedocs.io/ (Accessed: 31/05/2022)

## V. Conclusion and Future Work

Our initial results have provided us with useful knowledge regarding the current adoption of Python libraries in GitHub repositories. Our next steps will be directed towards enlarging the dataset and applying other unsupervised techniques to find more patterns and dependencies. In particular, we think the DBSCAN clustering algorithm will be useful to isolate libraries with dense usage from those used more sparsely. Then, we will continue our roadmap, focusing on exploding library metadata (e.g., from PyPi) and temporal patterns. All the collected knowledge will feed our intelligent assistant, which will be implemented as an extension or plug-in for IDEs (VSCode, Eclipse or Pycharm). Such an assistant is expected to give real-time recommendations to the developer with the aim of improving his/her code, automatically generating code lines to efficiently operate with the recommended libraries.

## Acknowledgment

Work supported by the Spanish Ministry of Science and Innovation (RED2018-102472-T, PID2020-115832GB-I00) and the Andalusian Regional Government (DOC_00944).

## References

[1] M. Valiev, B. Vasilescu, and J. Herbsleb, "Ecosystem-Level Determinants of Sustained Activity in Open-Source Projects: A Case Study of the PyPI Ecosystem," in *ACM Joint Meeting on European Softw. Eng. Conf. and Symposium on the Foundations of Softw. Eng.*, 2018, p. 644–655.

[2] F. Thung, D. Lo, and J. Lawall, "Automated library recommendation," in *20th Working Conf. Reverse Engineering*, 2013, pp. 182–191.

[3] M. A. Saied, A. Ouni, H. Sahraoui, R. G. Kula, K. Inoue, and D. Lo, "Improving reusability of software libraries through usage pattern mining," *J. Syst. Softw.*, vol. 145, pp. 164–179, 2018.

[4] P. T. Nguyen, J. Di Rocco, D. Di Ruscio, and M. Di Penta, "CrossRec: Supporting software developers by recommending third-party libraries," *J. Syst. Softw.*, vol. 161, p. 110460, 2020.

[5] P. T. Nguyen, J. Di Rocco, R. Rubei, C. Di Sipio, and D. Di Ruscio, "DeepLib: Machine translation techniques to recommend upgrades for third-party libraries," *Expert Syst. Appl.*, vol. 202, p. 117267, 2022.

[6] R. Barbudo, A. Ramírez, F. Servant, and J. R. Romero, "GEML: A grammar-based evolutionary machine learning approach for design-pattern detection," *J. Syst. Softw.*, vol. 175, p. 110919, 2021.

[7] N. Chirkova and S. Troshin, "Empirical Study of Transformers for Source Code," in *ACM Joint Meeting on European Softw. Eng. Conf. and Symposium on the Foundations of Softw. Eng.*, 2021, p. 703–715.

[8] M. Vidoni, "A systematic process for mining software repositories: Results from a systematic literature review," *Inf. Softw. Technol.*, vol. 144, p. 106791, 2022.

[9] C. Lucchese, S. Orlando, and R. Perego, "DCI_Closed: A Fast and Memory Efficient Algorithm to Mine Frequent Closed Itemsets," in *ICDM Workshop on Frequent Itemset Mining Implementations*, 2004, pp. 1–9.

# Green IN Artificial Intelligence: Energy Impact of Machine Learning Models

María Gutiérrez
Institute of Technology
and Information Systems
University of Castilla-La Mancha
Ciudad Real, Spain
Email: maria.ggutierrez@uclm.es

Félix García
Institute of Technology
and Information Systems
University of Castilla-La Mancha
Ciudad Real, Spain
Email: felix.garcia@uclm.es

***Abstract*—Nowadays, artificial intelligence (AI) algorithms are used in a wide range of applications, and as their use becomes more common, considering their environmental impact becomes an increasingly urgent topic. We present some empirical cases to illustrate different approaches for energy-efficient AI models: the choice of model and the choice of "real" data or synthetic data for the training dataset. These cases can be used as examples of how a green-driven approach can make contributions to the lifecycle of AI models, helping developers to take energy efficiency requirements into consideration when developing their programs and negotiate a suitable trade-off between energy efficiency and model reliability and performance.**

## I. Introduction

The use of artificial intelligence has been increasing over the past years, which has also generated a greater interest in researching the energy consumption of AI systems, and especially of those that include some form of machine learning or neural network. However, attempts at improving the energy efficiency of a model have traditionally been made by designing computer architecture that is particularly suited to run machine learning (ML) models, while the study of the models themselves has been overlooked.

However, there are reasons to research the energy consumption of the models themselves. Some studies already point out how the computing cost for ML models has increased exponentially over the last few years [5], and the lack of standardized methods for measuring the energy consumption of these kind of programs [2] hinders our abilities to adequately measure and understand their energetic behavior. In turn, this makes all the harder to design AI systems that take energy efficiency into consideration, since it is not clear which kind of design choices could contribute to its energy efficiency without compromising its performance.

In this proposal, we propose two empirical cases that illustrate possible design choices to develop and train ML models with an eye on their energy consumption: the choice of using "real" training data versus synthetic training data and the choice of the model itself.

## II. Proposed cases

### *A. Synthetic training data or real training data*

The most energetically expensive part of the development of a machine learning model often turns out to be its training, not only because it is a task with high computational demands, but also because finding the optimal combination of hyperparameters is a difficult process that requires a lot of experimentation.

For this case, we studied if the choice of training data could affect the overall consumption of a ML model during its training. Many systems are trained using "real" datasets, constructed from data extracted from the real world (such as hospital records, grocery store tickets, network traffic, etc.). However, training models on synthetic datasets is also an extended practice, since real data is not always available, or presents in an unsuitable format. These datasets are created by synthetic data generators, which produce data according to a particular distribution and are able to generate any amount of data on demand, on a format that requires little to no preprocessing.

To properly compare the energy consumption of training with a real dataset versus a synthetic one, we prepared a set of algorithms to perform a binary classification task. We set the task to be run on two different environments: first on MOA [1] and then on WEKA [4]. We chose four algorithms that are available on both platforms and are typical for this kind of task: naïve Bayes, Hoeffding tree, support vector machine (SVM) and logistic regression. For the datasets, we used a real dataset with data for credit card fraud detection, and then we created a synthetic version of that dataset with one of WEKA's synthetic data generators, trying to "recreate" the original dataset as closely as possible.

Once the tasks were set up, they were ran on a computer without any special capabilities, and the energy consumption of each algorithm while training on each dataset was measured using GSMP, FEETINGS' energy measuring methodology [3]. The analysis of the measurements showed that the models that were trained on the synthetic dataset consistently consumed less energy than the models trained on real data, both when they were run on MOA and on WEKA. The percentage of correct predictions for the models trained on the synthetic data was less than 10% lower than for the models trained with real data.

### *B. Choice of model*

It is known that some ML models are more suited to perform certain tasks, while being less apt for others. For this case, we propose that the choice of a ML model that is adequate for the task at hand is also a relevant choice for its energy efficiency, and not just for its performance.

## References

[1] Albert Bifet et al. "MOA: Massive Online Analysis". In: *The Journal of Machine Learning Research* 11 (Aug. 2010), pp. 1601–1604. ISSN: 1532-4435.

[2] Eva García-Martín et al. "Estimation of energy consumption in machine learning". en. In: *Journal of Parallel and Distributed Computing* 134 (Dec. 2019), pp. 75–88. ISSN: 07437315. DOI: 10.1016/j.jpdc.2019.07.007. URL: https://linkinghub.elsevier.com/retrieve/pii/S0743731518308773 (visited on 01/26/2022).

[3] Javier Mancebo et al. "FEETINGS: Framework for Energy Efficiency Testing to Improve Environmental Goal of the Software". en. In: *Sustainable Computing: Informatics and Systems* 30 (June 2021), p. 100558. ISSN: 2210-5379. DOI: 10.1016/j.suscom.2021.100558. URL: https://www.sciencedirect.com/science/article/pii/S2210537921000494 (visited on 02/04/2022).

[4] *Parallel File System Products — WEKA*. URL: https://www.weka.io/parallel-file-system/ (visited on 06/10/2022).

[5] Roy Schwartz et al. "Green AI". en. In: *Communications of the ACM* 63.12 (Nov. 2020), pp. 54–63. ISSN: 0001-0782, 1557-7317. DOI: 10.1145/3381831. URL: https://dl.acm.org/doi/10.1145/3381831 (visited on 01/25/2022).

# Discovery Service Federation in the Web of Things

Juan Alberto Llopis
*Applied Computing Group (TIC-211)*,
University of Almería
Almería, Spain
jalbertollopis@ual.es

Javier Criado
*Applied Computing Group (TIC-211)*,
University of Almería
Almería, Spain
javi.criado@ual.es

Luis Iribarne
*Applied Computing Group (TIC-211)*,
University of Almería
Almería, Spain
luis.iribarne@ual.es

***Abstract*—As the number of technological devices in our environment increases, the need for mechanisms to facilitate their use becomes more evident. A large part of these devices are Internet of Things (IoT) devices, devices that are accessible through a wide set of technologies, protocols and applications. To homogenize the representation, communication and management of IoT systems, web technology offers a series of mechanisms that facilitates the construction of heterogeneous systems. Inspired by this idea, the Web of Things (WoT) concept is an IoT approach that uses web technology to create a representation of physical entities in digital space. To perform the integration between both spaces, it is necessary to identify the devices or physical entities deployed and to establish a communication between the physical and digital space. Therefore, there is a need for a mechanism that allows applications and entities to obtain or discover services that meet their requirements. In this PhD thesis, we propose a federated discovery service for the WoT, capable of finding and registering WoT devices within reach of the discovery service or available in other discovery services. In a federated discovery service, when searching for devices, a device can be located in the same discovery service or other discovery services. Therefore, a criterion has to be established to select between (a) devices in a discovery service, (b) devices in different discovery services; and (c) discovery services. The proposed discovery service uses a recommender system to suggest devices that best suit the user's requirements. This work can reduce the workload of the development of Smart solutions by facilitating the discovery of WoT devices and by reducing the use of energy due to device response calls.**

## I. Introduction

With the increase in the number of deployed Internet of Things (IoT) devices and with the integration of the physical and digital space to represent physical entities on the web, searching for devices has become more difficult [1]. The Web of Things (WoT) is an approach that homogenizes the representation, communication and management of IoT systems, facilitating the interaction between devices and external entities [2]. However, there is still a problem when looking for devices, due to the increasing number of devices, the use of techniques to improve the search process is desired.

When searching for services, discovery services were developed to facilitate the search for services that meet certain requirements among a large number of services offered [3]. Discovery services make use of a repository or directory that stores the descriptions of the set of services it manages, avoiding the need to define them multiple times, and facilitating their access and use by interested applications or entities [4]. This reduces the number of requests made directly on devices, reducing the energy usage of the devices and increasing their usage time [5]. Discovery services in the field of service-oriented architectures (SOA) are a mechanism widely used by the community [6], [7]. In the IoT domain, there are also approaches that provide some middleware and Content Distribution Networks (CDNs) for service discovery [8], [9]. However, there are still no discovery services in the WoT domain that take into account needed features in these systems such as interoperability, security or service heterogeneity, among other possible examples [10], [11].

This work has been funded by the EU ERDF and the Andalusian Government (Spain) under the project I+D UrbanITA, ref. PY20_00809. Juan Alberto Llopis has been funded by a grant of the Spanish Government FPU19/0072.

In this PhD thesis, we propose a discovery service federation for the WoT, capable of delegating and extending queries to other discovery services when a discovery service is not able to answer a query. In this way, the proposed discovery service will not only allow identification, classification and use of the available entities but will also allow the discovery of these entities through other discovery services. In addition, a discovery service can be ineffective if the queries are not constructed correctly. Therefore, the research analyzes techniques to perform query mutation, so that a query that does not obtain results can be automatically modified by creating variants of that query to discover devices.

Finally, based on a request sent by a client, and taking into account the matching conditions, a discovery service returns a set of devices that meet the client's requirements. The returned set of devices can be ordered, prioritized, and/or filtered to allow the selection of the best alternative. Such selection may be based on criteria that look at various features, such as user preferences, usage history, or other information of interest to the application domain. For this reason, this PhD thesis will incorporate the application of recommender systems to be able to suggest the best alternatives during service discovery [12]. Figure 1 shows the proposed linkage between federated discovery systems and recommender systems. The recommender system will use Artificial Intelligence (AI) to recommend devices from queries in the form of a natural language sentence.

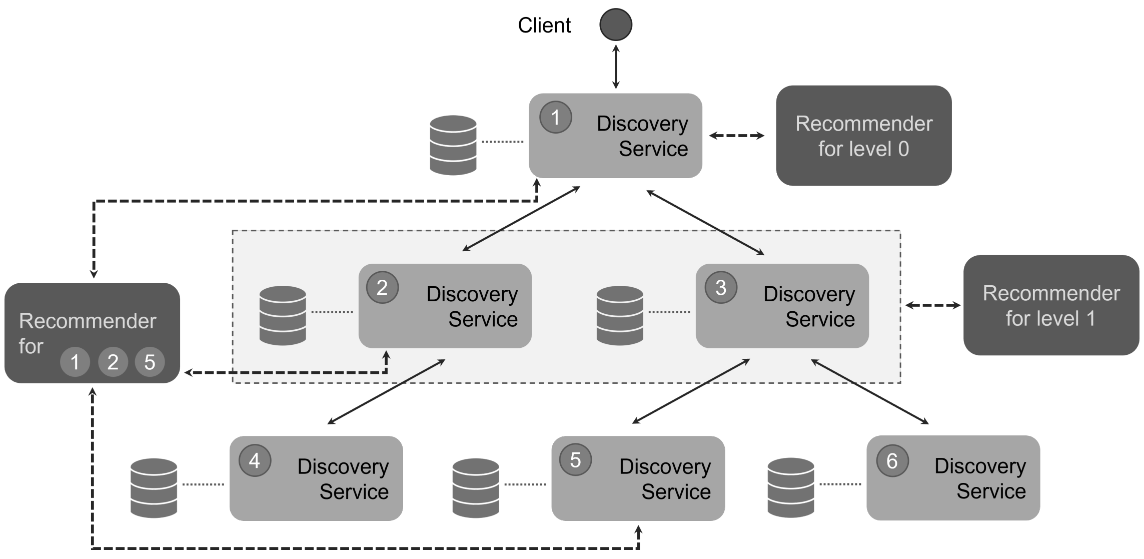


Fig. 1. Federated discovery services and recommender systems for WoT resources.

## II. Hypothesis and objectives

This thesis aims to study the possibility of implementing a federated discovery service that solves the identification of services, taking into account features such as interoperability, security and heterogeneity of services. Furthermore, the discovery service makes use of other discovery services to search for devices deployed in different locations. Finally, the system must be able to build, simulate and validate devices using their Thing Description, providing a configuration generation for deploying IoT devices in smart buildings. The following are the specific objectives, all of them in the Web of Things domain:

(a) To analyze and experiment with protocols and ontologies of service identification and classification.
(b) To experiment with the identification of services using other discovery services and provide a solution for the communication between discovery services.
(c) Establish a federated system in the discovery service and study the use of the compatibility rate in service discovery.
(d) Provide a repository solution and define the configuration generation system based on the Discovered Services.
(e) Integrate concepts of recommender systems and discovery services.
(f) Study particular aspects such as query mutation, data quality, ontologies, interoperability and security.

Objectives (a), (d) and (f) are partially completed with publications in national and international conferences. We expect to finish objectives (a) and (d) before the end of the year. Regarding objectives (b), (c) and (e); currently, we are working on (e), which we expect to finish this year with publications in journals. Finally, objectives about creating a federated discovery service, (b) and (c), will be solved in 2023 and 2024 as the ending of the thesis.

## References

[1] N. K. Tran, Q. Z. Sheng, M. A. Babar, and L. Yao, “Searching the web of things: State of the art, challenges, and solutions,” *ACM Comput. Surv.*, vol. 50, aug 2017.
[2] D. Guinard, V. Trifa, and E. Wilde, “A resource oriented architecture for the web of things,” in *2010 Internet of Things (IOT)*, pp. 1–8, 2010.
[3] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, “Service-oriented computing: State of the art and research challenges,” *Computer*, vol. 40, no. 11, pp. 38–45, 2007.
[4] S. Cirani, L. Davoli, G. Ferrari, R. Léone, P. Medagliani, M. Picone, and L. Veltri, “A scalable and self-configuring architecture for service discovery in the internet of things,” *IEEE Internet of Things Journal*, vol. 1, no. 5, pp. 508–521, 2014.
[5] G. Tanganelli, C. Vallati, and E. Mingozzi, “Edge-centric distributed discovery and access in the internet of things,” *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 425–438, 2018.
[6] M. Crasso, A. Zunino, and M. Campo, “A survey of approaches to web service discovery in service-oriented architectures,” *Journal of Database Management*, vol. 22, no. 1, pp. 102–132, 2011.
[7] S. Dasgupta, A. Aroor, F. Shen, and Y. Lee, “Smartspace: Multiagent based distributed platform for semantic service discovery,” *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 44, no. 7, pp. 805–821, 2014.
[8] A. H. Ngu, M. Gutierrez, V. Metsis, S. Nepal, and Q. Z. Sheng, “Iot middleware: A survey on issues and enabling technologies,” *IEEE Internet of Things Journal*, vol. 4, no. 1, pp. 1–20, 2017.
[9] A. Forestiero, “A smart discovery service in internet of things using swarm intelligence,” in *Theory and Practice of Natural Computing* (C. Martín-Vide, R. Neruda, and M. A. Vega-Rodríguez, eds.), (Cham), pp. 75–86, Springer International Publishing, 2017.
[10] M. Aziez, S. Benharzallah, and H. Bennoui, “Service discovery for the internet of things: Comparison study of the approaches,” in *2017 4th International Conference on Control, Decision and Information Technologies (CoDIT)*, pp. 0599–0604, 2017.
[11] B. Pourghebleh, V. Hayyolalam, and A. Aghaei Anvigh, “Service discovery in the internet of things: review of current trends and research challenges,” *Wireless Networks*, vol. 26, pp. 5371–5391, OCT 2020.
[12] N. N. Chan, W. Gaaloul, and S. Tata, “A recommender system based on historical usage data for web service discovery,” *Service Oriented Computing and Applications*, vol. 6, pp. 51–63, MAR 2013.

# *SmartTLC*: Towards Smart Traffic Light Systems

José R. Lozano-Pinilla
QSEG, Universidad de Extremadura, Spain
Email: joserralp@unex.es

C. Vicente-Chicote
QSEG, Universidad de Extremadura, Spain
Email: cristinav@unex.es

***Abstract*—The increasing number of fuel-based vehicles has several negative impacts on the environment, the economy and the citizen's daily life, being the largest contributor to Green-House Gas (GHG) emissions, mostly due to traffic congestion. Actually, more and more cities are deploying ICT-based infrastructures to monitor the traffic and its environmental impact (air pollution, noise, etc.). In this line, this paper describes *SmartTLC*: a software framework, aimed at enabling the simulation and comparison of different traffic light adaptive control algorithms based on traffic data (either historical, real-time or both). This framework allows designers to select the best traffic light control strategy for different situations, indicating which one achieves better results in terms of reducing traffic congestion. The experimental results obtained so far demonstrate that the adoption of a context-aware adaptive approach significantly improves traffic fluidity, reducing vehicle waiting time, in particular, in roads with a higher traffic demand.**

## I. Introduction

The increasingly growing number of vehicles, nowadays mostly fuel-based, has important negative impacts not only in the environment and in the personal and global economy, but also in our daily life [1]. In 2019, the transportation sector was the largest contributor to *Green-House Gas* (GHG) emissions, accounting up to 23% in Europe (29% in the USA) of the total GHG emissions. Up to 82% of these emissions were produced by light-duty vehicles (58%) and medium and heavy trucks (24%) [2], although these are not homogeneously distributed across all continents, countries and regions [3].

According to the *International Energy Outlook 2021* [4], electric vehicles currently make up only 30% of the 1.446 billion cars estimated on Earth in 2022 [3]. Furthermore, the report seriously warns that emissions from the transportation sector are expected to increase through 2050 unless world leaders establish legal and regulatory changes. In this context, the implementation of new mobility management policies becomes an urgent must.

## II. Hypothesis

In this work we propose a smart traffic light control system aimed at helping designers analyze how different traffic light strategies behave in different situations and select the one that provides better results in terms of alleviating traffic congestion. Addressing this issue is a highly complex challenge that involves (1) context-awareness, enabling real-time mobility monitoring (road demand, average waiting time, GHG emissions, etc.); (2) the identification of (eventually changing) traffic patterns; and (3) the adequacy of the road infrastructures and of the traffic lights that control the vehicular (and pedestrian) flows in urban environments.

Appropriately processing the context data provided by road-located *IoT* sensors can be useful (1) to predict future road demands and adequately plan and design the required infrastructures; (2) to identify relevant traffic patterns (e.g., daily peak hours) in order to schedule the best (predefined) traffic light control policies; and (3) to react to unforeseen situations (e.g., an unexpected increase of road demand), enabling the dynamic adaptation of the traffic light control strategy.

## III. Objectives

The ***SmartTLC*** framework is targeted at helping designers simulate different traffic conditions and analyze which traffic light control algorithm performs better in each of them. To achieve this goal, ***SmartTLC*** can use either (1) the information about the real-time traffic conditions provided by (real or simulated) IoT sensors; (2) predictions based on the traffic patterns learnt from a historical traffic dataset; or (3) both. Based on this information, the framework allows designers to simulate different traffic light control algorithms to find out which one achieves a higher vehicle waiting time reduction. It is worth noting that reducing vehicle waiting time can also indirectly improve other metrics such as air and noise pollution, fuel consumption, etc.

In this work, both the traffic conditions and the results of applying four different traffic light control algorithms were simulated using the SUMO framework [5].

## IV. Scenario

The scenarios defined to test the ***SmartTLC*** framework are based on four different concepts: (1) the network topology, representing roads, lanes, signals and traffic regulations; (2) the vehicular traffic types, representing different traffic intensities (vehicles per hour), along with a numeric range to support traffic uncertainty; (3) the traffic light algorithms, where the green phase can be calculated in different ways; and (4) the traffic time patterns, representing how the traffic evolves during a day. The scenario defined for this preliminary study considered: (1) a topology with a single junction, where vehicles could only travel north-south (NS) or east-west (EW), i.e., turns were not supported; (2) four different traffic types, each one defining a range of vehicles per hour from 1-5 (very low) to 350-650 (high); (3) four different traffic light control approaches, including a basic fix-cycle one and three adaptive algorithms based, in turn, on historical traffic data, real-time traffic data and a combination of both; and (4) 10 traffic patterns covering regular week days, weekends, bank holidays and summer holidays, among others.

## V. Learning Traffic Patterns

Based on the 10 traffic patterns previously mentioned, a 1-year traffic dataset was generated. For each day, depending on its traffic pattern (week day, weekend day, etc.), the traffic was randomly generated according to the different traffic intensities established per hour. In order to add some more randomness and realism to the data, a "noise" policy consisting in swapping the traffic patterns of a few (randomly selected days) and, within a few days, the traffic intensities defined by the corresponding pattern, was implemented.

The generated traffic dataset was used to train the "*Traffic Light Predictor*" component of the ***SmartTLC*** framework so that it could predict the traffic pattern at any given date and hour. Five machine learning models were trained using this data: (1) Naive Gaussian Bayes (NGB); (2) Support Vector Machines (SVM) both linear and polynomial; (3) K-Nearest Neighbors (KNN); (4) Decision Trees (DT); and (5) Random Forest (RF). These models were trained both using only time-related information (hour, day, month and year) and extending it with traffic-related information (actual vehicles passing per hour). An hyperparameter tuning approach was used in order to achieve even better predictions.

The complete results achieved using the five different models, in terms of elapsed time and F1 score, are available in [6](section 5.2.3). As expected, using both time- and traffic-related information achieves much better results. The best model using only time-related information was DT (F1 score 0.6924), while the best one using both time- and traffic-related information was KNN (F1 score 0.9998). The fastest one was DT in both cases (0.0010 seconds).

## VI. Results

We have considered four different traffic light control algorithms. The **"no-adaptation"** approach, based on a fixed-cycle with green and red phases of identical duration (common approach in most current traffic lights), was developed to be used as a reference baseline. Then, the other three algorithms implemented an adaptive approach consisting in proportionally increasing the duration of the green phase for the direction with a higher traffic intensity. The main difference among these three algorithms is how they identify the current traffic demand: (1) the **"date-based"** approach predicts the current traffic intensity based only on historical data (e.g., if today it is Monday and it is 14:00, the traffic intensities NS and EW are likely to be *high* and *medium*, respectively. Thus, increase the duration of the green phase for NS). As expected, this approach performs much better than the no-adaptation approach when the traffic behaves more or less as predicted (hopefully, most of the time). However, if it doesn't, it may eventually worse traffic congestion. The main benefit of this approach is that it does not require real-time traffic monitoring (which can be economically quite expensive), although it requires a dataset generation and a traffic pattern learning process (both computationally expensive); conversely, (2) the **"real-time"** approach adapts the traffic lights based only on real-time data. Obviously, this approach performs much better than the two previous ones as it takes into account actual (*vs* predicted) traffic conditions and, thus, can instantly react to changes. In this case, benefits and limitations are opposite to those described for (2). Finally, (3) the **"combined"** approach uses both historical and real-time data to adapt the traffic light phases. The results achieved in this case are almost identical to those obtained by (2). However, this approach allows adapting the real-time monitoring frequency, relaxing it when the traffic behaves more or less as predicted, and increasing it when not.

## VII. Conclusion and Future Works

The results achieved in this preliminary research demonstrate that, even when no real-time data is available (either because there is no monitoring infrastructure or because it eventually fails), it is possible to reduce traffic congestion using historical data and learning traffic patterns from it. Furthermore, when real-time data is available, it is possible to use predicted traffic patterns to lower the computational load required to monitor the traffic conditions and dynamically adapt the traffic lights accordingly. It is worth mentioning that the ***SmartTLC*** framework, developed as part of this work, has been designed to be easily extended, supporting arbitrary complex topologies, new traffic patterns, and traffic light control strategies. A detailed description of the ***SmartTLC*** framework can be found in [6], and its implementation can be downloaded from [7].

Some future works include: (1) enriching the scenarios with pedestrians and new types of vehicles; (2) considering special adaptation policies for emergency vehicles (e.g., fire trucks or ambulances); (3) supporting additional traffic light control algorithms; (4) using more complex artificial intelligence models to predict traffic intensity; (5) running simulations on more complex topologies and allowing drivers to perform additional actions (e.g., turn both left and right, overtaking, etc.); and (6) model how the traffic information obtained in a particular junction can be propagated to neighbor ones considering their distance, possible escapes and new traffic contributions from non-monitored roads, etc.

## References

[1] EIT for Urban Mobility: Solving the mobility challenges facing our cities together, https://www.eiturbanmobility.eu/ (2021).

[2] U. S. EPA: Fast Facts on Transportation Greenhouse Gas Emissions, https://www.epa.gov/greenvehicles/fast-facts-transportation-greenhouse-gas-emissions (2021).

[3] Hedges Company: How many cars are in the world in 2022: Market Research, https://hedgescompany.com/blog/2021/06/how-many-cars-are-there-in-the-world/ (2021).

[4] U. S. EIA: International Energy Outlook 2021, https://www.eia.gov/outlooks/ieo/index.php (2021).

[5] Eclipse SUMO. https://www.eclipse.org/sumo/.

[6] Lozano Pinilla, José R. SmartTLC: A Smart Traffic Light Control System for Urban Environments. MSc. Thesis (2022).

[7] Lozano Pinilla, José R., SmartTLC Implementation (2022).

# Transforming Mobile Software Ecosystems with Semi-Automatic Feature Integration through Dialogue-Based Feedback

Quim Motger
PhD Student
Department of Service and Information System Engineering
Universitat Politècnica de Catalunya
Barcelona, Catalonia, Spain
jmotger@essi.upc.edu

Xavier Franch
Co-supervisor
Department of Service and Information System Engineering
Universitat Politècnica de Catalunya
Barcelona, Catalonia, Spain
franch@essi.upc.edu

Jordi Marco
Co-supervisor
Department of Computer Science
Universitat Politècnica de Catalunya
Barcelona, Catalonia, Spain
jmarco@cs.upc.edu

***Abstract*—Mobile applications have become a daily-use commodity worldwide. Users need to manage a wide variety of complex use cases using multiple, isolated applications in combination to achieve higher, more complex goals. As each user's goals are unique, defining these integrations becomes a challenging task. To address these challenges, users can play a proactive role by providing valuable feedback for runtime integration. Hence, the use of conversational agents to assist users by collecting natural language feedback can be considered as a key research trend towards this end. Following a design science methodology, we aim at exploring how the integration of mobile application features can be better supported by actively integrating users through dialogue-based feedback collection techniques. To this end, we define the following scientific objectives: (1) transforming mobile applications with automatic feature integration mechanisms; (2) generating natural language data-sets for training conversational agents in the context of mobile software ecosystems; and (3) designing explicit feedback collection techniques for feature integration. We envisage that our research will contribute to explore the potential of users' natural language feedback for personalized software experiences in the context of mobile software ecosystems.**

## I. Introduction and motivation

With the adoption of smartphone devices as ubiquitous tools for both personal and professional use cases [1], mobile software ecosystems [2] have become complex, heterogeneous and constantly-evolving environments, with particular challenges from the user experience perspective. As users' goals and needs are unique, managing and fully exploiting the potential of the applications' portfolio is a challenging task, which includes the ability of such systems to adapt hardware and software components to match the users' needs [3]. Despite the intrinsic interconnectivity between mobile apps and their features, there is a lack of focused research in the field of cross-app feature integration, and these strategies vaguely offer any customization to extend mobile software ecosystems in order to deliver personalized feature integration paradigms. To achieve a highly personalized, customizable user experience in terms of feature integration, actively involving the user is fundamental to learn about their unique goals and needs. To this end, effective analysis of explicit user feedback in adaptive software systems is a fundamental strategy in adaptive software systems [4], to which the use of natural language interfaces or conversational agents is emerging as a leading research trend in a wide variety of domains [5], [6].

Based on this context, in this paper, we summarize the objectives, research questions and initial results of the thesis titled "*Transforming Mobile Software Ecosystems with Semi-Automatic Feature Integration through Dialogue-Based Feedback*". This research is intended to explore how the integration of a mobile-based conversational agent can be used to provide personalized user experiences in terms of cross-app feature integration among the applications' portfolio of mobile users. We expect that our research will contribute to lay the groundwork for future research both in adaptive mobile software ecosystems and the adoption of personalized conversational agents.

## II. Research method

The research method is based on an adaptation of the Design Science methodology for Information Systems and Software Engineering as defined by Wieringa [7]. We designed a research plan based on three research process iterations, including the following activities: (i) definition of objectives; (ii) design and development; (iii) demonstration; (iv) evaluation and verification; and (v) dissemination.

Following the *Goal Question Metric* (GQM) template, we state the general objective of this project as follows:

***Analyze*** feature integration supported by dialogue-based feedback
***for the purpose of*** enhancing and personalizing the user experience
***with respect to*** cross-app feature integration
***from the point of view of*** users
***in the context of*** mobile software ecosystems.

To guide the achievement of the aforementioned objective, we have defined the following research questions (RQ):

| |
|---|
| **RQ1.** What is the current state of research in the field of feature integration in mobile software ecosystems? |
| **RQ2.** What is the current state of research in the field of software-based dialogue systems? |
| **RQ3.** How mobile app and feature integration can be better supported by actively integrating users through dialogue-based feedback collection techniques? |

## III. Initial results

### A. RQ1: Mobile app feature integration

We conducted a state-of-the-art review of gray and white literature in the field of mobile app feature integration. The results of this narrative literature review demonstrate that there are very few methodological and technical references to feature and app integration in the context of mobile software ecosystems which go beyond service and data integration (i.e., focusing on functionalities). Some proposals like mashup environments offer easy-to-use integration components, tools, and services which are generally focused on service and data integration. And while some approaches like MashReDroid [8] define feature integration strategies, users are required to be actively involved only by explicitly defining their own, specific integrations following a record and replay strategy.

### B. RQ2: Software-based dialogue systems

We conducted a systematic literature review of secondary studies in the field of dialogue-based software systems [9]. The results of this review reinforce software-based dialogue systems as an emerging trend in recent scientific literature. The latest innovations in the field are focused on contextualized, personalized dialogue experiences not only through more advanced natural language understanding strategies (e.g., transformer models) but also through the integration of conversational agents as embedded subcomponents into large, complex software systems. Consequently, contextualization and personalization are perceived as key features to achieve higher user adherence and user satisfaction.

### C. RQ3: Feature integration through dialogue-based feedback

Using the conclusions from RQ1 and RQ2 as a proxy, we refined three scientific objectives based on the general objective (as presented in Section II). We additionally identified three scientific objectives based on the research objective:

- Design and develop a mobile app repository for the management, transformation and delivery of mobile applications with semi-automatic feature integration capabilities.
- Integrate metadata and natural language data collection, data modelling and data storage techniques in order to build a natural language understanding data-set for a domain-specific sub-set of mobile applications.
- Design and develop a mobile-based conversational agent to facilitate personalized, contextualized mobile feature integrations using dialogue-based feedback collection techniques.

During the first iteration of the research process, we have focused on analyzing and adopting the required technologies to build our solution, as well as to develop proof-of-concept (PoC) data-sets, artifacts, tools, and processes for each of the software components composing the solution. Specifically:

- Data collection service based on the automatic exploration of APIs, app stores and web scrapping of mobile apps search engines and catalogs for the collection of natural language related data (e.g., app descriptions, reviews, official websites...).
- Repository of mobile app metadata and natural language data fields using a knowledge graph data model strategy.
- Keyword extraction process for the automated recognition of mobile app functionalities using natural language data sources, based on syntactic and semantic natural language processing techniques.
- Conversational agent based on an adaptive knowledge base for discussions based on a personalized mobile app catalog, including support for a proof-of-concept integration of mobile features.
- Proof-of-concept integration between two features of two open-source native Android applications.
- Draft model for structuring, documenting and maintaining personalized feature integrations for a specific user in a specific domain.

## Acknowledgment

With the support from the Secretariat for Universities and Research of the Ministry of Business and Knowledge of the Government of Catalonia and the European Social Fund. This paper has been funded by the Spanish Ministerio de Ciencia e Innovación under project / funding scheme PID2020-117191RB-I00 / AEI/10.13039/501100011033.

## References

[1] K. Bahia and A. Delaporte, "The state of mobile internet connectivity report 2020 - mobile for development," 2021. [Online]. Available: https://www.gsma.com/r/somic/

[2] E. M. Grua, I. Malavolta, and P. Lago, "Self-adaptation in mobile apps: a systematic literature study," in *2019 IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, 2019.

[3] Q. Motger, X. Franch, and J. Marco, "Integrating adaptive mechanisms into mobile applications exploiting user feedback," in *Research Challenges in Information Science*, Cham, 2021.

[4] K. Jasberg and S. Sizov, "Human uncertainty in explicit user feedback and its impact on the comparative evaluations of accurate prediction and personalisation," *Behaviour & Information Technology*, 2020.

[5] Nivethan and S. Sankar, "Sentiment analysis and deep learning based chatbot for user feedback," in *Intelligent Communication Technologies and Virtual Mobile Networks*. Springer International Publishing, 2019.

[6] C. Liu, B. Zhang, and G. Peng, "A systematic review of information quality of artificial intelligence based conversational agents in healthcare," in *Distributed, Ambient and Pervasive Interactions*, 2021.

[7] R. J. Wieringa, *Design Science Methodology for Information Systems and Software Engineering*. Springer Berlin Heidelberg, 2014.

[8] J. Zheng, L. Shen, X. Peng, H. Zeng, and W. Zhao, "MashReDroid: enabling end-user creation of Android mashups based on record and replay," *Science China Information Sciences*, vol. 63, no. 10, 2020.

[9] Q. Motger, X. Franch, and J. Marco, "Software-based dialogue systems: Survey, taxonomy and challenges," *ACM Comput. Surv.*, 2022.

# New ideas and work in progress

# BLEU it All Away! Refocusing SE ML on the Homo Sapiens

Leonhard Applis
TU Delft
L.H.Applis@tudelft.nl

***Abstract*—Many tasks in machine learning for software engineering rely on prominent NLP metrics, such as the BLEU score. The metrics are under heavy criticism themselves within the NLP community, but the SE community adapted them for lack of better alternatives. Within this paper, we summarize some of the problems with common metrics at the examples of code and look for alternatives. We argue that our only hope is the worst of all possible options: Humans.**

## I. Introduction

In ancient Greece, Hephaistos was accompanied by servant automatons to help around his forge, freeing him to spent his time on true masterpieces. This Hellenic ideal of automation lives up to this day and has its renaissance with software engineers: Tedious tasks such as writing tests [1] or documentation [2] are shifted towards automation to give room for the developers creativity. The narrative is great — the results are often humbling. Presentations for Githubs CoPilot [3] pick cherries, but thorough investigations usually lead to disturbing or amusing results. How did we end up here ?

One issue are the metrics. For this paper we focus on Documentation Generation [2], which is lately often interpreted as a translation task from source code to human language (i.e. English) and draws a lot from NLP research, such as sequence-to-sequence models [4] but also the most common metric BLEU [5]. In recent work, Gehrmann et al. [6] criticized the metric driven approaches and publications in NLP (specifically generation tasks). Among their primary findings are that ⓐ people *blindly* use existing datasets without manual inspection, sampling, etc. ⓑ people rarely inspect output manually or involve *end-users* ⓒ all publications use BLEU for lack of better options or for acceptance at a venue. Gehrmann et al. proposition is as compelling as it is easy: Instead of using *big data* and arguably weak metrics, center the evaluation around a group of expert users.

The remainder of this paper first highlights some flaws with BLEU in documentation generation in Section II and elaborates on the proposed solution in Section III. While we cover only one domain briefly, we consider this to be a general critique applicable for most domains. We close in Section IV by arguing that we need to change the course of SE-ML-Metrics, and while the proposal might not be perfect, it is one we haven't tried in a long time.

## II. The Flaws

BLEU [5] is a metric to evaluate quality of translation and text-generation techniques. It compares the overlap of n-grams in a produced text compared to one or more reference texts, where commonly a four-gram is used, as it correlates closest to human acceptance [7]. There is wide criticism on BLEU [8], [9], but we highlight issues specific to the domain of software engineering:

① While BLEU takes n-grams into account, many pieces of programming language and documentation will produce a solid score despite sometimes contrary meaning. With common tokenization, *return ( a + b) == ( b - a );* and *return ( a - b) == ( b + a );* scores near perfect in BLEU. Some publications opt for one-gram BLEU, for which the above example gives an optimal match.

② Eghbali et al. [10] investigated the BLEU-Scores of randomly chosen samples from within different corpora. In a corpus of English literature there was a BLEU of ≈20%, comparing two random elements from Javas scores ≈40%. This is stunning insofar as these numbers form the expected *baseline* if we could produce random elements that follow the same distribution. In their initial publication, CodeBERT produces a BLEU-Score of 17.65% [2], which is 2.4% worse than drawing random elements.

③ Unlike natural language, programming languages (and their documentation) invent new words frequently. This is known as the open vocabulary problem [11] and is addressed in SE mostly by encodings. Prominent are BytePairEncoding [11] and Subword-Splitting [12]. Both increase the number of tokens - hitherto they benefit the BLEU score. It poses two primary issues: (3a) it is harder to evaluate and compare the metric if evermore strings become attached. (3b) the research field itself becomes overburdened in experiment-complexity only for the sake of metrics.

## III. The Options

One approach to address the issues is to blame the metric; *The BLEU is dead, long live the BLEU*. One can easily stitch together *"MetaBLEU"* that combines normal BLEU for language-representation and stopword-cleaned BLEU for content-coverage. Similar *fixes* for BLEU have been proposed [13], [10], [14], mostly ductaping the underlying problems. These are not done in bad faith, on the contrary they fit perfectly in the current paradigm of ML publications — more data, more features and better tuned models can be used with the same benchmark and promise a safe academic voyage. But as a research field, we will hit dead ends by the need for ever more data, a cacophony of metrics and hungry computation.

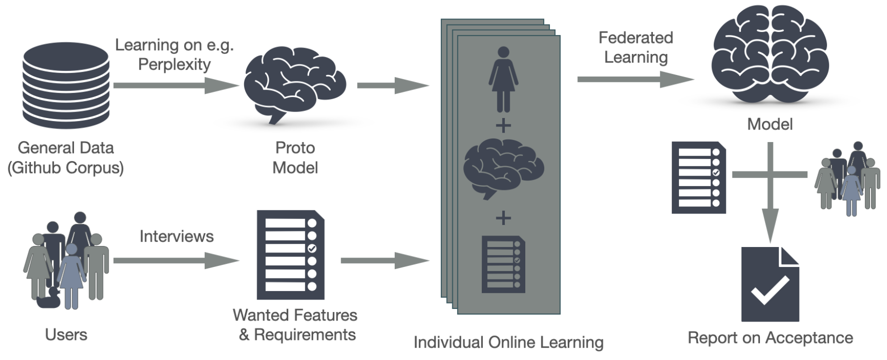

Learning on e.g. Perplexity
General Data (Github Corpus)
Proto Model
Users
Interviews
Wanted Features & Requirements
+
+
Individual Online Learning
Federated Learning
Model
Report on Acceptance

Fig. 1. Proposed Pipeline for SE Model Training

Apart from these issues, another question remains: Is a model with good BLEU score useful? The only way to answer this is to ask real humans, real users. Gehrmann et al. [6] come to a similar conclusion and argue for model-cards based on expert-based qualitative analysis. Theoretically there are few fields easier to change evaluations than Software Engineering; Software Engineers produce the data, ML-libraries, models, metrics and are the final users.

The concrete suggestion (shown in Figure 1) is to start models with metrics, and produce *proto models* that cover a basic understanding of vocabulary and distributions. The downstream-tasks should be tuned with humans in the loop, by rating various aspects of the specific task (content, quality of language, feedback time, inter-prediction quality, etc.). Rating-Criteria should be derived from and with the final users, in a fashion like requirements engineering. This pipeline is similar to e.g. CodeBERT [15], which learns general perplexity on Code and then is fine-tuned for the specific task and language. The BERT-Core and the Code-Addition would form the *proto model* and the downstream-task of documentation generation would be done in active learning with experts rating samples, instead of *blind* metrics. Pieces for this novel pipeline are available and tested [16], [17], and could themselves make great use-cases for reinforcement learning and federated learning.

## IV. Conclusion

Following metrics down the rabbit hole lead us into a ML wonderland of free publications — but for outsiders we are just kids in an asylum. If our goal is to make models that are useful to developers and help them in their business, the only metric we really have to maximize is their feedback. No developer tries to write documentation with a certain BLEU score, hence we should turn our back on these proxy-metrics. We should trust our users that they know what they want, and change our own research to accommodate for their needs.

## References

[1] G. Fraser and A. Arcuri, “Evosuite: automatic test suite generation for object-oriented software,” in *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, 2011, pp. 416–419.

[2] “Codexglue: A benchmark dataset and open challenge for code intelligence,” 2020.

[3] Github. [Online]. Available: https://copilot.github.com/

[4] B. Li, M. Yan, X. Xia, X. Hu, G. Li, and D. Lo, *DeepCommenter: A Deep Code Comment Generation Tool with Hybrid Lexical and Syntactical Information*. New York, NY, USA: Association for Computing Machinery, 2020, p. 1571–1575. [Online]. Available: https://doi.org/10.1145/3368089.3417926

[5] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, “Bleu: a method for automatic evaluation of machine translation,” in *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, 2002, pp. 311–318.

[6] S. Gehrmann, E. Clark, and T. Sellam, “Repairing the cracked foundation: A survey of obstacles in evaluation practices for generated text,” *arXiv preprint arXiv:2202.06935*, 2022.

[7] D. Coughlin, “Correlating automated and human assessments of machine translation quality,” in *Proceedings of Machine Translation Summit IX: Papers*, 2003.

[8] C. Callison-Burch, M. Osborne, and P. Koehn, “Re-evaluating the role of bleu in machine translation research,” in *11th conference of the european chapter of the association for computational linguistics*, 2006, pp. 249–256.

[9] G. Doddington, “Automatic evaluation of machine translation quality using n-gram co-occurrence statistics,” in *Proceedings of the second international conference on Human Language Technology Research*, 2002, pp. 138–145.

[10] A. Eghbali and M. Pradel, “Crystalbleu: Precisely and efficiently measuring the similarity of code,” 2022, 2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE). [Online]. Available: https://conf.researchr.org/details/icse-2022/icse-2022-posters/14/CrystalBLEU-Precisely-and-Efficiently-Measuring-the-Similarity-of-Code

[11] R.-M. Karampatsis, H. Babii, R. Robbes, C. Sutton, and A. Janes, “Big code!= big vocabulary: Open-vocabulary models for source code,” in *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*. IEEE, 2020, pp. 1073–1085.

[12] H. Babii, A. Janes, and R. Robbes, “Modeling vocabulary for big code machine learning,” *arXiv preprint arXiv:1904.01873*, 2019.

[13] S. Ren, D. Guo, S. Lu, L. Zhou, S. Liu, D. Tang, N. Sundaresan, M. Zhou, A. Blanco, and S. Ma, “Codebleu: a method for automatic evaluation of code synthesis,” *arXiv preprint arXiv:2009.10297*, 2020.

[14] T. Sellam, D. Das, and A. P. Parikh, “Bleurt: Learning robust metrics for text generation,” 2020. [Online]. Available: https://arxiv.org/abs/2004.04696

[15] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang *et al.*, “Codebert: A pre-trained model for programming and natural languages,” *arXiv preprint arXiv:2002.08155*, 2020.

[16] B. Settles, “Active learning literature survey,” 2009.

[17] M. Aledhari, R. Razzak, R. M. Parizi, and F. Saeed, “Federated learning: A survey on enabling technologies, protocols, and applications,” *IEEE Access*, vol. 8, pp. 140 699–140 725, 2020.

# Generating Complex Metamorphic Relations for Cyber-Physical Systems with Genetic Programming

Jon Ayerdi*, Valerio Terragni†, Aitor Arrieta*, Paolo Tonella‡ and Maite Arratibel §
Mondragon Unibertsitatea*, University of Auckland †, Università della Svizzera italiana (USI) ‡, Orona §
*{jayerdi,aarrieta}@mondragon.edu, †v.terragni@auckland.ac.nz, ‡paolo.tonella@usi.ch, §marratibel@orona-group.com

***Abstract*—One of the major challenges of testing complex Cyber-Physical Systems (CPS) is determining whether the behavior of the system on a particular test execution is correct or not, the so called oracle problem. Metamorphic testing is an alternative testing technique which can alleviate this problem by reasoning over relations between the inputs and outputs of multiple test executions, which are known as Metamorphic Relations (MRs). However, defining effective MRs is a complicated task which is error-prone and often requires domain-expertise and knowledge about the system under test. In this paper, we present an approach for the automatic generation of MRs based on genetic programming. Our current implementation is specifically tailored for generating MRs for performance testing CPSs, and is based on previous work on the evolutionary generation of program assertions. Furthermore, we propose an extension based on our experience with this approach. We also present the results obtained with an industrial case study in previous iterations of this work, which have been promising thus far.**

## I. Introduction

Cyber-Physical Systems (CPSs) are complex heterogeneous systems that integrate physical and software components [1]. The complexity of CPSs and their interactions with their environment makes the definition of test oracles especially challenging for them [2]. This inability to determine whether these system's behaviour is correct or not is known as the *oracle problem* [3].

We present an automated approach to mitigate the oracle problem in an industrial case study from Orona [4], one of the leading elevator companies in Europe. We employ oracles based on metamorphic testing, and we automate the definition of these oracles by using genetic programming and samples of correct and incorrect executions [5]. Based on our experience and previous results, we also describe the future research avenues to improve the approach and make it more applicable.

## II. Background

Metamorphic Testing aims to detect system failures by defining necessary relations between the inputs and outputs of two or more test executions, the so called **Metamorphic Relations (MRs)** [6]. Typically, a single *source* test case $\langle I_s, O_s \rangle$ is executed, and then a *follow-up* test case $\langle I_f, O_f \rangle$ is generated such that $I_s$ and $I_f$ satisfy the input relation.

An example MR derived for multi-elevator installations [7] is that increasing the number of available elevators ($E$) should reduce the average waiting time of the passengers ($AWT$), since the system will have more resources to meet the transportation demands. This MR can be expressed as:

$$E_f = E_s \cup E' \Rightarrow AWT_s \geq AWT_f$$

where $E_f = E_s \cup E'$ is the *input relation*, which defines how the follow-up test inputs relate with the source test inputs, and $AWT_s \geq AWT_f$ is the *output relation*, i.e., the assertion that defines whether the system's behaviour is correct.

## III. Approach

Manually defining MRs is, unfortunately, a difficult and error-prone task which requires expertise with the system under test. Because of this, we proposed GAssertMRs [5], a tool for automatically generating MRs by using samples of correct and incorrect system behaviours.

GAssertMRs is built based on GAssert [8], a technique to automatically generate or improve assertion oracles. GAssertMRs explores the space of candidate MRs with a co-evolutionary algorithm introduced by GAssert, which formulates the oracle improvement problem as a multi-objective optimization problem with three objectives of descending priorities: (1) minimizing the number of *false positives*, (2) minimizing the number of *false negatives*, and (3) minimizing the *complexity* of the generated MRs. GAssertMRs evolves two populations of MRs in parallel: One which favors fewer false positives first, and another which favors fewer false negatives first. The remaining objectives are used to break ties, with MR complexity being the lowest priority objective in all cases. Both populations share genetic material by periodically exchanging their best individuals.

GAssertMRs employs user-defined input transformations, and generates output relations of the following form:

$$O_f \; [operator] \; F(O_s, I_s, I_f)$$

where $O_s$ and $O_f$ are the values for the output variable, $[operator]$ is a relational operator, $I_s$ and $I_f$ are the input variable values, and $F$ is the expression generated by GAssertMRs. We generate MRs following this specific template in order to improve the readability of the generated MRs and greatly reduce the search-space for the algorithm.

The current implementation of GAssertMRs supports only boolean and numeric variables, constants and operators, which can be limiting. For complex types, the variable can often be decomposed or serialized into boolean and numeric variables. For instance, an elevator passenger call ($c$) can be destructured into the arrival time ($c.timestamp$), arrival floor ($c.source$), and destination floor ($c.destination$).

Unfortunately, this solution is not ideal because it cannot handle element collections, such as the lists of elevator positions ($E$) and passenger calls ($C$) from our elevation case study [7]. The current workaround is to allow user-defined functions to reduce the test inputs into domain-specific boolean or numeric features, which can then be used to generate the MRs [5]. For instance, $TD_{worst}(E_1, E_2)$ is a domain-specific function which computes the distance between the elevator positions $E_1$ and $E_2$ [7].

Our proposed solution, inspired by lambda calculus [9], is to add *higher-order functions* such as $fold$, $map$ and $filter$ to GASSERTMRS's expression language to enable it to reason on item sequences such as lists or sets. Implementing this feature implies introducing the new *sequence* and *function* types for expressions. The new *function* expressions could use the following literal syntax:

$$[param1, param2, ...]\{expression\}$$

With this extension, all of the user-defined functions we use for the elevation case study can be generated by GASSERTMRS, meaning that there would be no need for the users to define them. For example, $TD_{worst}(E_1, E_2)$ could be expressed by GASSERTMRS as follows:

$$sum(map([e_1, e_2]\{abs(e_1 - e_2)\}, zip(E_1, E_2)))$$

here, $zip(C_1, C2)$ combines pairs of elements from $C_1$ and $C_2$ into a single sequence, $map(F, C)$ applies the transformation $F$ to each of the elements from $C$, and $sum(C)$ computes the sum of all the elements from $C$. All of these are generic functions which could be part of GASSERTMRS's expression language. The benefit of this extension is two-fold: Firstly, GASSERTMRS becomes more expressive and generic, and secondly, the implicit bias introduced by user-defined functions is eliminated.

## IV. Preliminary results

We evaluated GASSERTMRS (with no support for higher-order functions) on ORONA's case study [5], comparing the automatically generated MRs with ones that were manually generated with the assistance from domain experts [7].

The same test suite and system variants with manually seeded faults were used to evaluate both approaches. The selected Metamorphic Relation Input Patterns (MRIPs), which define the input relations of the MRs, were the following:

- **MRIP1:** An additional passenger call is appended.
- **MRIP2:** One or more additional elevators are enabled.
- **MRIP3:** The initial position of the elevators is changed.

Table I shows the results obtained by GASSERTMRS compared with the manually generated MRs. This evaluation shows that GASSERTMRS is able to generate non-trivial MRs which dominate the results obtained by the manual MRs in terms of both detected failure count and mutation score. Unlike the manual MRs, the ones generated by GASSERTMRS did have some FPs, but the median was always 0.

TABLE I
EVALUATION RESULTS (MEDIAN) [5]

| MRIP | Metric | Operator | Detected Failures | | Mutation Score | |
|---|---|---|---|---|---|---|
| | | | **GAssertMRs** | **Manual** | **GAssertMRs** | **Manual** |
| MRIP1 | AWT | $\geq$ | 16.5 | 13.0 | 11.80% | 11.24% |
| | | $\leq$ | 28.0 | 14.0 | 14.61% | 14.61% |
| | TD | $\geq$ | 12.0 | 12.5 | 12.36% | 13.48% |
| | | $\leq$ | **30.5** | **4.5** | 13.48% | 5.06% |
| | TM | $\geq$ | **20.0** | **7.5** | 16.85% | 7.87% |
| | | $\leq$ | 9.0 | 0.0 | 4.49% | 0.00% |
| MRIP2 | AWT | $\geq$ | **11.0** | **0.0** | **7.87%** | **0.00%** |
| | | $\leq$ | 74.0 | 74.0 | 19.10% | 18.54% |
| | TD | $\geq$ | 9.0 | 0.5 | **8.43%** | **0.56%** |
| | | $\leq$ | **47.5** | **8.5** | 12.36% | 9.55% |
| | TM | $\geq$ | 2.0 | 1.5 | 1.12% | 1.12% |
| | | $\leq$ | **22.0** | **2.0** | 8.99% | 2.25% |
| MRIP3 | AWT | $\geq$ | **28.5** | **1.5** | **13.48%** | **1.12%** |
| | | $\leq$ | **50.0** | **0.0** | **19.10%** | **0.00%** |
| | TD | $\geq$ | **58.0** | **1.5** | **17.98%** | **1.69%** |
| | | $\leq$ | **30.0** | **0.5** | **12.36%** | **0.56%** |
| | TM | $\geq$ | **16.0** | **0.5** | 7.87% | 0.56% |
| | | $\leq$ | **19.0** | **0.0** | 2.25% | 0.00% |

In conclusion, GASSERTMRS appears to be a feasible approach for generating effective MRs instead of manually defining them.

## V. Conclusion

This paper summarizes our approach to automatically generate MRs for CPSs by using correct/incorrect execution samples and genetic programming. We also describe our experience of using our implementation of this approach, GASSERTMRS, on an industrial case study from the elevation domain provided be ORONA. Furthermore, we point out the current limitations that hinder the general adoption of this technique and present the solutions which we intend to implement.

## References

[1] R. Baheti and H. Gill, "Cyber-physical systems," *The impact of control technology*, vol. 12, no. 1, pp. 161–166, 2011.

[2] A. Kane, T. Fuhrman, and P. Koopman, "Monitor based oracles for cyber-physical system testing: Practical experience report," in *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE, 2014, pp. 148–155.

[3] E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, and S. Yoo, "The oracle problem in software testing: A survey," *IEEE TSE*, vol. 41, no. 5, pp. 507–525, 2014.

[4] Orona, "Orona group," https://www.orona-group.com/, 2021.

[5] J. Ayerdi, V. Terragni, A. Arrieta, P. Tonella, G. Sagardui, and M. Arratibel, "Generating metamorphic relations for cyber-physical systems with genetic programming: an industrial case study," in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2021, 2021, pp. 1264–1274. [Online]. Available: https://doi.org/10.1145/3468264.3473920

[6] T. Y. Chen, S. C. Cheung, and S. M. Yiu, "Metamorphic testing: A new approach for generating next test cases," Technical Report HKUST-CS98-01, Department of Computer Science, The Hong Kong University of Science and Technology, Tech. Rep., 1998.

[7] J. Ayerdi, S. Segura, A. Arrieta, G. Sagardui, and M. Arratibel, "Qos-aware metamorphic testing: An elevation case study," in *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2020.

[8] V. Terragni, G. Jahangirova, P. Tonella, and M. Pezzè, "Evolutionary improvement of assertion oracles," in *ESEC/FSE*, 2020, pp. 1178–1189.

[9] H. P. Barendregt *et al.*, *The lambda calculus*. North-Holland Amsterdam, 1984, vol. 3.

# Regression Testing for Self-driving Cars as Cyber-physical Systems in Virtual Environments

Christian Birchler
*Zurich University of Applied Sciences*
Winterthur, Switzerland
birc@zhaw.ch

***Abstract*—Detecting bugs early in the software testing process is crucial to lowering the costs. Testing cyber-physical systems (CPS) is more expensive than traditional software systems, so regression testing for CPS is a promising approach to deal with the higher testing costs. Regression testing consists of test selection, prioritization, and minimization. Our work focuses on using self-driving cars (SDC) as a CPS use case.**

**First, we tackled the problem of test selection for CPS in virtual environments. For this, we developed a tool to select test scenarios from the test suite with machine-learning models for SDCs. *SDC-Scissor* is a cost-effective test selector for SDC software that can predict failing tests with an F1-score up to 96% and speed up the test execution in simulation by 170%. This selection approach reduces the time spent for running tests that likely pass but increases those that likely fail.**

**Secondly, to deal with the test prioritizing problem, we developed a multi-objective genetic algorithm called *SDC-Prioritizer*. It finds the optimal test execution so that the diversity of tests maximizes and the execution time minimizes. The evaluation of *SDC-Prioritizer* shows that our approach outperforms a random and a greedy baseline prioritizer statistically significantly. I.e., *SDC-Prioritizer* detects more defects as the baseline with the same execution time. With this tool, the testing process is getting more efficient by revealing more defects in a shorter time.**

**Both tools use labeled datasets of road scenarios that consist of lists of consecutive road points for evaluation. The execution of the scenarios labels the tests as safe or unsafe, i.e., if the SDC is driving off the lane or not. Future work will minimize the test scenarios and complete the regression testing framework.**

## I. Introduction

The future of mobility will be driven by Artificial Intelligence (AI). There are several examples of the replacement of humans with AIs [20], [24], [26]. In 2021, Tesla began to deliver their SDCs that only rely on camera input [25]. The AI that drives the car relies only on the camera data and the according vision-based algorithms. These examples show the direction toward autonomous mobility that heavily rely on AI and algorithms.

However, some CPS, like SDCs are life-critical and require proper testing. Several incidents with humans happened in the past [18], [19], [21], [22] which clearly shows that some of these incidents could have been prevented with proper testing. Proper testing of CPS is also challenging [1], and several studies have addressed already some aspects [9], [12], [27].

Nevertheless, testing CPS is still costly, especially in-field tests and even simulation-based tests. In-field tests require testing real, physical, and most likely also expensive hardware in a real-world environment that might damage the test subject. To mitigate this issue, testing in simulation environments can prevent damaging the real test subject, but the computation of the simulation environment is more expensive than testing traditional software systems since complex computation regarding the virtual environment must be performed [13], [15].

We adapt the concept of regression testing to the CPS domain for system-level testing in virtual environments. Our contribution is a cost-effective test selector *SDC-Scissor* [7] that covers the selection process in regression testing, whereas *SDC-Prioritizer* [8] orders the selected test cases based on a multi-objective genetic algorithm.

We aim to minimize the test scenarios to execute only the critical part of a scenario. With this approach, we try to minimize further the execution costs of tests in virtual environments. The work on test case minimization will complement our central research vision:

> Development of a regression testing framework for cyber-physical systems.

This paper will give a brief overview of the related work, methodology, preliminary results, and conclusion in Sections II, III, IV and, V respectively.

## II. Related Work

Several studies about testing CPS efficiently based on model checking or simulations [10], [16], [23]. A more concrete approach proposes Humeniuk et al. [14] with a search-based test generation. They applied a multi-objective search algorithm to have diversity and track the deviation of the observed system's behavior from its expected behavior. A diverse test suite can be generated for testing the lane-keeping ability of SDCs in simulation.

First studies regarding regression testing for CPS have also been performed [2]–[6], [17]. They tackled the problem of test case selection and prioritization for different levels, e.g., MiL, SiL, HiL, etc. The use cases mainly were CPS with different configurations in product lines by applying different search algorithms.

In our vision, we want to have a complete regression testing framework for SDCs that includes test case selection, prioritization, and minimization. The techniques use ML models and evolutionary genetic algorithms to minimize the costs in the testing process. SDCs were not often considered use cases in previous work, although SDCs are already a reality on our streets.

## III. Methodology

We run test scenarios in simulation to have a labeled dataset. A single test case consists of road points defining the whole road the SDC has to follow. The execution of simulation tests allows to classify the tests as "safe" and "unsafe". Several test data are already publicly available [11].

The primary pipeline is provided by *SDC-Scissor*. The components and APIs of *SDC-Scissor* allow modifying, adjusting, or creating a pipeline for conducting experiments with SDCs in virtual environments.

## IV. Preliminary Results

*SDC-Scissor* is a cost-effective test selector for SDC software that can predict failing tests with an F1 score of up to 96% and speed up the test execution in simulation by 170%. This selection approach reduces the time spent running tests that likely pass but increases those that likely fail.

The evaluation of the *SDC-Prioritizer* shows that our approach outperforms a random and greedy baseline prioritizer statistically significantly. I.e., *SDC-Prioritizer* detects more defects as the baseline with the same execution time. With this tool, the testing process is getting more efficient by revealing more defects in a shorter time.

## V. Conclusions

Our ongoing work shows that we can improve test selection and prioritization significantly. The testing process for SDCs in virtual environments can be more time-efficient. The next step towards our vision is to enable test case minimization for SDCs in virtual environments sot that only relevant parts of a virtual scenario are executed.

## References

[1] S. Abbaspour Asadollah, R. Inam, and H. Hansson. A survey on testing for cyber physical system. In *International Conference on Testing Software and Systems*, pages 194–207. IFIP, Springer, 2015.

[2] A. Arrieta, J. A. Agirre, and G. Sagardui. Seeding strategies for multi-objective test case selection: an application on simulation-based testing. In *Genetic and Evolutionary Computation Conference*, pages 1222–1231. ACM, 2020.

[3] A. Arrieta, S. Wang, A. Arruabarrena, U. Markiegi, G. Sagardui, and L. Etxeberria. Multi-objective black-box test case selection for cost-effectively testing simulation models. In *Genetic and Evolutionary Computation Conference*, pages 1411–1418. ACM, 2018.

[4] A. Arrieta, S. Wang, U. Markiegi, A. Arruabarrena, L. Etxeberria, and G. Sagardui. Pareto efficient multi-objective black-box test case selection for simulation-based testing. *Information and Software Technology*, 114:137–154, 2019.

[5] A. Arrieta, S. Wang, G. Sagardui, and L. Etxeberria. Search-based test case selection of cyber-physical system product lines for simulation-based validation. In *International Systems and Software Product Line Conference*, pages 297–306, 2016.

[6] A. Arrieta, S. Wang, G. Sagardui, and L. Etxeberria. Search-based test case prioritization for simulation-based testing of cyber-physical system product lines. *Journal of Systems and Software*, 149:1–34, 2019.

[7] C. Birchler, N. Ganz, S. Khatiri, A. Gambi, and S. Panichella. Cost-effective simulation-based test selection in self-driving cars software with sdc-scissor. In *IEEE International Conference on Software Analysis, Evolution, and Reengineering*. IEEE, 2022.

[8] C. Birchler, S. Khatiri, P. Derakhshanfar, S. Panichella, and A. Panichella. Single and multi-objective test cases prioritization for self-driving cars in virtual environments. *arXiv preprint arXiv:2107.09614v2*, 2022.

[9] J. H. Castellanos and J. Zhou. A modular hybrid learning approach for black-box security testing of cps. In *International Conference on Applied Cryptography and Network Security*, pages 196–216. Springer, 2019.

[10] P. M. Chu, M. Wen, J. Park, H. Kaisi, and K. Cho. Three-dimensional simulation for training autonomous vehicles in smart city environments. In *International Conference on Internet of Things and IEEE Green Computing and Communications and IEEE Cyber, Physical and Social Computing and IEEE Smart Data*, pages 848–853. IEEE, 2019.

[11] P. Derakhshanfar, A. Panichella, A. Gambi, V. Riccio, C. Birchler, and S. Panichella. TRAVEL: A Dataset with Toolchains for Test Generation and Regression Testing of Self-driving Cars Software, Jan. 2022.

[12] J. Deshmukh, M. Horvat, X. Jin, R. Majumdar, and V. S. Prabhu. Testing cyber-physical systems through bayesian optimization. *ACM Transactions on Embedded Computing Systems (TECS)*, 16(5s):1–18, 2017.

[13] H.-M. Huang, T. Tidwell, C. Gill, C. Lu, X. Gao, and S. Dyke. Cyber-physical systems for real-time hybrid structural testing: a case study. In *International Conference on Cyber-physical Systems*, pages 69–78. ACM/IEEE, 2010.

[14] D. Humeniuk, F. Khomh, and G. Antoniol. A search-based framework for automatic generation of testing environments for cyber-physical systems. *Information and Software Technology*, page 106936, 2022.

[15] P. S. Kumar, W. Emfinger, and G. Karsai. A testbed to simulate and analyze resilient cyber-physical systems. In *International Symposium on Rapid System Prototyping (RSP)*, pages 97–103. IEEE, 2015.

[16] T. Kuroiwa, Y. Aoyama, and N. Kushiro. Testing environment for cps by cooperating model checking with execution testing. *Procedia Computer Science*, 96:1341–1350, 2016.

[17] U. Markiegi, A. Arrieta, L. Etxeberria, and G. Sagardui. Dynamic test prioritization of product lines: An application on configurable simulation models. *Software Quality Journal*, 29(4):943–988, 2021.

[18] 2 killed in driverless tesla car crash, officials say. https://www.nytimes.com/2021/04/18/business/tesla-fatal-crash-texas.html. Accessed: 2022-01-28.

[19] Drohne krachte in zürich auf den boden - post bezeichnet vorfall als inakzeptabel. https://www.nzz.ch/zuerich/absturz-von-post-drohne-bericht-stellt-gravierende-maengel-fest-ld.1492295?reduced=true. Accessed: 2022-02-08.

[20] J. O'Callaghan. Few aeroplanes land automatically but new systems could make this the norm. https://ec.europa.eu/research-and-innovation/en/horizon-magazine/few-aeroplanes-land-automatically-new-systems-could-make-norm, October 2019. Accessed: 2022-06-25.

[21] After deadly 737 max crashes, damning whistleblower report reveals sidelined engineers, scarcity of expertise, more. https://www.theregister.com/2021/12/15/boeing_737_max_senate_report/. Accessed: 2022-01-28.

[22] Post-drohne über zürichsee abgestürzt. https://www.srf.ch/news/regional/zuerich-schaffhausen/blutprobe-verloren-post-drohne-ueber-zuerichsee-abgestuerzt. Accessed: 2022-02-08.

[23] J. Sun and Z. Yang. Objsim: efficient testing of cyber-physical systems. In *International Workshop on Testing, Analysis, and Verification of Cyber-Physical Systems and Internet of Things*, pages 1–2. ACM SIGSOFT, 2020.

[24] Future of driving. https://www.tesla.com/autopilot. Accessed: 2022-06-25.

[25] Transitioning to tesla vision. https://www.tesla.com/support/transitioning-tesla-vision. Accessed: 2022-06-25.

[26] Waymo. https://waymo.com/. Accessed: 2022-06-25.

[27] X. Zhou, X. Gou, T. Huang, and S. Yang. Review on testing of cyber physical systems: Methods and testbeds. *IEEE Access*, 6:52179–52194, 2018.

# Incremental Just-In-Time Test Generation in Lock-Step with Code Development

Carolin Brandt
*Delft University of Technology*
c.e.brandt@tudelft.nl

***Abstract*—State-of-the-art test generation strategies employ advanced analyses of the code under test and powerful optimization algorithms to generate automatic test cases for software systems. As these techniques require a large amount of computational power, they are often limited to generating tests after the code under test is already written. However, today's broad education about the importance of software testing lets developers strive to create test cases directly with new code they are contributing.**

**To support these developers, we want to develop an incremental just-in-time test generation tool that works in close proximity to the development of the code under test. Whenever the developer creates a new class or functionality, the tool automatically proposes a matching test case. When the developer finishes implementing a new condition, the tool automatically recommends an additional test case that tests the code which was just added. The generated test cases are closely based on the existing test cases in the project with small, incremental changes to test the new lines of code.**

**To realize such a just-in-time test generation tool we have to tackle many challenges: Detecting the completion of a test-worthy condition, generating a fitting test case in a short time on the developer's machine, or effectively communicating the value of the new test case to the developer. With the participants of the SMILESENG Summer School we want discuss our new idea, brainstorm on the challenges that this research opens up and identify possible approaches to tackle them.**

## I. Introduction

To illustrate our idea of just-in-time test generation, let us introduce this anecdotal use case: Think of Jada, a software engineer in a large software development company. Her team is working on an app for public transport trips and tickets. The transport provider decided to introduce a new promotion: In the summer months of 2022, each monthly pass will cost only nine euros. Today, Jada's task is to adapt the ticket selection algorithm to propose the new ticket whenever the normal cost of a trip would be more than nine euros. Jada opens the code for the ticket selection component and adds a new condition comparing the trip price to the promotional ticket price. After finishing this new edge case, a notification pops up in the corner of her editor:

> "Do you want to add a test that a summer ticket is proposed when it is cheaper than the normal fare?"

As their project policy requests all new code to be fully tested, she is relieved to not have to write a test from scratch. She selects "Inspect Test" and the editor opens on the test class of the ticket selection component. The new test is already added to the source code and a green indicator shows her that the test is passing. Jada reviews the new test case and is pleased that here addition seems to work as she intended: An expensive route—initialized just as in the other tests—is passed to the selection, which then returns the summer ticket instead of the normal ticket. She accepts the new test case and commits it together with her changes. When creating the pull request, she can be confident that all changes are already covered by the test suite. And that mostly automated, thanks to the just-in-time test generator!

This research was funded by the Dutch science foundation NWO through the Vici "TestShift" grant (No. VI.C.182.032)

This is one use case we envision for our new technology. The just-in-time test generator closely follows the software developer's actions, identifies testable, test-worthy and finished scenarios, quickly generates matching test cases in the background, and immediately presents these test cases to the developer. The developer inspects the new test cases, modifies them where they see fit and takes them over into their maintained test suite.

Placing the test generation so close to the code development, provides advantages in several known challenges of automatic test generation:

- It narrows the search space by focusing the generation efforts on the just modified code.
- It makes it easier for developers to understand the behavior and coverage impact of the new test case, as they are still in the mental context of the code under test [1].
- By generating the oracle through executing the just modified code, the developer receives immediate feedback on the actual behavior of their code and whether it matches with the behavior they intended.
- This approach widens the application area of automatic test generation to the initial development of code and directly supports developers that aim to write test cases in conjunction with their production code (test-guided [2] or iterative test-last development [3])

## II. What We Can Build Upon: Related Work

The idea of just-in-time test generation is closely related *test suite augmentation* [4]: adding new test cases to an existing test suite in order to improve its code coverage. Code-to-test traceability approaches [5], [6] can identify test cases that execute code close to the just modified code and with the help of symbolic execution and similar techniques we can modify these base test cases to exercise the new scenario [7].

Powerful *search-based test generation* tools like EvoSuite [8] are already able to generate whole test suites from scratch. To apply them to just-in-time test generation, one would need to investigate how existing test cases could be used effectively as an initial population and how well the limited search space can improve the runtime/generation quality towards interactive performance. *Test amplification* [9] generates new test cases by mutating the input stage of the test case and generating assertions matching the new test behavior. In a previous study, we investigated the interaction of software developers with automated test amplification [1]. We saw that it is important to give the developer control over the interaction, provide them with the information necessary to judge the test cases and effectively communicate the impact that the generated test case will have on the quality of their test suite. Taking into account the results of further user studies of test generation tools [10]–[12], we conjecture that a strong focus on the design of the user interaction is crucial for our just-in-time test generation approach to be successful.

Machine learning approaches are gaining popularity also in the area of automatic test generation, e.g. to generate assertion statements [13], [14]. Recently, neural code completion tools such as GitHub Copilot[1] are able to propose fully fledged implementations when triggered with a natural language method name. Nonetheless, it remains to be investigated how effective they are in generating useful test cases and which information needs to be encoded in the trigger to steer the generation towards the intended code under test.

## III. Divide and Conquer: Steps to Tackle

On the way to fully-fledged just-in-time test generation we see several challenges to be addressed. These could be the basis for our discussion at the SMILESENG summer school, together with the following questions:

Which further challenges do you see as part of realizing just-in-time test generation?
What approaches should we explore to tackle them?
What chances or caveats do you see with these approaches?

- **Cutting out a *test-worthy* condition / scenario** as a target for the test generation. Approaches could be detecting coherent edits made by the developer, and learning from single-concern commits or the coverage of existing test cases. Together with the following test generation, this should be a deterministic and transparent process to let the developer build trust and understanding in the capabilities of our tool.
- **Detecting the right moment to contact the developer** as well as giving them control to start and feed the tool themselves.
- **Rapid on-device generation** to enable interactive cooperation between developer and tool. To speed up the test generation, we propose to leverage *incrementality*: Building upon existing test cases by modifying them only slightly. In addition, we look at incremental compilation and building to speed up a run of a test generation tool, as well as explore ways to avoid the expensive executions of test cases to measure adequacy metrics such as mutation score or structural coverage.

[1] https://copilot.github.com/

## IV. An Outlook into the Future

The possible applications of a just-in-time test generation tool and its components will be much wider than just adding test cases for new conditions in the code. We could propose immediate updates to no-longer-passing test cases before the developer reruns their tests after a change. We could determine no longer needed test cases after large code cleanups and propose suiting re-locations of test code after refactorings. Integrating test generation right within the developer's workflow lets them become familiar with the advantages test generation can offer and lets them gain trust in the capabilities of (partially) automated software engineering.

While just-in-time test generation is still many steps away, the development of each its subparts helps us strengthen and better understand the area of automatic test generation and its interaction with software developers. We are excited to present our idea to the SMILESENG Summer School participants and together discuss approaches to tackle it.

## References

[1] C. Brandt and A. Zaidman, "Developer-centric test amplification," *Empir. Softw. Eng.*, vol. 27, no. 4, 2022.

[2] M. Beller *et al.*, "Developer testing in the IDE: patterns, beliefs, and behavior," *IEEE Trans. Softw. Eng.*, vol. 45, no. 3, 2019.

[3] A. Santos *et al.*, "A family of experiments on test-driven development," *Empir. Softw. Eng.*, vol. 26, no. 3, 2021.

[4] R. Bloem *et al.*, "Automating test-suite augmentation," in *2014 14th Int. Conf. on Quality Softw.* IEEE, 2014.

[5] N. Aljawabrah *et al.*, "Understanding test-to-code traceability links: The need for a better visualizing model," in *Computational Science and Its Applications - 19th Int. Conf.* Springer, 2019.

[6] B. V. Rompaey and S. Demeyer, "Establishing traceability links between unit test cases and units under test," in *13th European Conf. on Softw. Maintenance and Reengineering*. IEEE Computer Society, 2009.

[7] Z. Xu, Y. Kim, M. Kim, G. Rothermel, and M. B. Cohen, "Directed test suite augmentation: techniques and tradeoffs," in *18th ACM SIGSOFT Int. Symp. on Foundations of Softw. Eng.* ACM, 2010.

[8] G. Fraser and A. Arcuri, "EvoSuite: Automatic test suite generation for object-oriented software," in *m19th ACM SIGSOFT Symp. on the Foundations of Softw. Eng.* ACM, 2011.

[9] B. Danglot, O. L. Vera-Pérez, B. Baudry, and M. Monperrus, "Automatic test improvement with DSpot: A study with ten mature open-source projects," *Empir. Softw. Eng.*, vol. 24, no. 4, 2019.

[10] M. M. Almasi *et al.*, "An industrial evaluation of unit test generation: Finding real faults in a financial application," in *39th Int. Conf. on Softw. Eng.: Softw. Eng. in Practice*. IEEE Computer Society, 2017.

[11] S. Panichella *et al.*, "The impact of test case summaries on bug fixing performance: An empirical investigation," in *38th Int. Conf. on Softw. Eng.* ACM, 2016.

[12] J. M. Rojas *et al.*, "Automated unit test generation during software development: A controlled experiment and think-aloud observations," in *2015 Int. Symp. on Softw. Testing and Analysis*. ACM, 2015.

[13] C. Watson, M. Tufano, K. Moran, G. Bavota, and D. Poshyvanyk, "On learning meaningful assert statements for unit test cases," in *42nd Int. Conf. on Softw. Eng.* ACM, 2020.

[14] H. Yu *et al.*, "Automated assertion generation via information retrieval and its integration with deep learning," in *42th Int. Conf. on Softw. Eng.*, 2022.

# Towards Bug Localization in Models in Game Software Engineering

Rodrigo Casamayor, Lorena Arcega, Francisca Pérez, and Carlos Cetina
Universidad San Jorge
Escuela de Arquitectura y Tecnología
Zaragoza, Spain
Email: rcasamayor@acm.org, {larcega, mfperez, ccetina}@usj.es

***Abstract*—Today, video game development is one of the fastest growing industries in the world. Video games present characteristics that differentiate their development and maintenance from the development and maintenance of classic software. Nowadays, most video games are developed by means of the so-called game engines. A key artifact of game engines are software models. Hence, the way in which testing is done must inevitably be different in video games than in traditional software since the artifacts used are also different. Current approaches for locating bugs in software are focused on source code, while the approaches needed to locate bugs in video games should consider other artifacts such as software models. The lack of specific approaches leads to a longer development time, which sometimes causes delays in the deadlines and postponement of the launch date. This results in the video game being released without having been properly tested. In our work, we want to evaluate whether the bug localization techniques applied to the models of classic software engineering work in game software engineering. Specifically, we want to leverage the use of evolutionary algorithms and explore different fitness functions, such as textual similarity with the bug reports and the defect localization principle. In addition, we want to study the application of one of the main inherent features underlying the video game domain to achieve bug location: the simulations using non-player characters (NPC). It is common for video games to include non-player characters that accompany the player in the adventure, are the enemies to beat, or simply populate the world recreated in the video game. These NPCs have pre-programmed behaviors and can be used to launch gameplay simulations.**

## I. Introduction

Today, video game development is one of the fastest growing industries in the world. Such is the relevance and depth that video games have in our society that, if we put it in terms of developer population, the video game industry is responsible for 8.8M active developers as of 2019 [1]. According to the same report, the total number of active software developers is 18.9M, so almost one out of every two developers is involved in the games sector. Furthermore, video game development is instrumental in achieving the vision of the Metaverse. This might suggest that the number of video game developers will continue to grow in the future as the Metaverse is developed.

Video games present characteristics that differentiate their development and maintenance from the development and maintenance of classic software [2]. Most video games are developed by means of so called game engines. A game engine refers to a development environment that integrates a graphics engine and a physics engine as well as a set of tools that wraps around them in order to accelerate development. Developers can create video game content directly using code (e.g., C++) or the software models of the engines.

Current approaches for locating bugs in software focus on source code, while the approaches needed to locate bugs in video games should consider other artifacts such as software models [2]. Nevertheless, a fault localization survey [3] reveals that none of the bug localization approaches consider models as the source of the bugs, which poses a considerable problem for developers since much of the video game content remains unexplored. Actually, Politowski et al. argue that the way in which developers deal with bugs must inevitably be different in video games than in traditional software since the artifacts used are also different [4].

The lack of specific bug localization approaches leads to a longer development time, which sometimes causes delays in the deadlines and postponement of the launch date. This results in the video game being released with an excessive number of bugs, such as the case of the blockbuster *Cyberpunk 2077* [5].

There are significant differences between Classic Software Engineering (CSE) and Game Software Engineering (GSE) [6], [4]. Our work argues that the differences can become opportunities for tackling the challenge of bug localization in video games. Specifically, we propose to leverage game simulations to locate bugs in the software models of video games. In video games, it is common to include non-player characters (NPCs). They accompany the player in the adventure, are the enemies to beat, or simply populate the world recreated in the video game. These NPCs have pre-programmed behaviors and can be used to launch gameplay simulations.

## II. Background

The case study that we are using to evaluate our ideas is performed using the bosses of the video game Kromaia [1]. Kromaia is a three-dimensional video game. Each of the levels involves a player's spaceship flying from a starting point to a target destination reaching the goal before being destroyed. If the player manages to reach the destination, the final boss corresponding to that level appears and must be defeated in order to complete the level.

[1]See the official Playstation trailer to learn more about Kromaia: https://youtu.be/EhsejJBp8Go

The bosses are specified with the Shooter Definition Model Language (SDML). SDML is a DSL model for the video game domain. This DSL follows the main ideas of MDE using models for Software Engineering. The models are created using SDML and interpreted at runtime. SDML defines aspects such us the anatomical structure, the amount and distribution of vulnerable parts, weapons, and defenses in the structure/body of the character, and the movement behaviors associated to the whole body or its parts. This modeling language has concepts such as hulls, links, weak points, weapons, and AI components.

The simulations using the Kromaia case study simulate a duel between a boss and a human player. During the simulation, the simulated player faces the boss in order to destroy the weak points that are available at that moment, whereas the boss acts according to the anatomy, behavior, and attack/defense balance that is included in its model, trying to defeat the simulated player. In the simulation, both the boss and the simulated player try to win the match and do not avoid confrontation, try to prevent draw/tie games, and try to ensure that there is a winner.

## III. Approach

Our starting point is BLiMEA [7], [8], which is specifically designed to locate bugs in software models. As many bug localization approaches do [3], BLiMEA uses bug reports and the defect localization principle [9]. BLiMEA iterates through the models of a system and assesses model fragments as possible sources of bug. To do so, BLiMEA uses a multi-objective evolutionary algorithm with two fitness functions: Information Retrieval (IR) and modification timespan. This approach receives a bug description and a set of software models as input. The output is a set where each model fragment has been assigned two fitness values: the similarity to the bug description, and the timespan to the most recent model fragment modifications.

However, we want to leverage game simulations to locate bugs in the software models of video games. The aim of the approach is to find the most relevant simulation to locate the target bug. Our preliminary approach takes as input a set of software models in which we want to locate the bug. The goal is to obtain a ranked list of simulation traces that are ordered by their relevance in locating the bug. To do so, the evolutionary algorithm performs a search that is guided by a fitness function. The first attempt for the fitness function is reward simulations that are farthest from what developers expected. The idea is that if they have strayed from what the developers expected, they might be relevant when locating a bug.

## IV. Preliminary results

BLiMEA was successful in the context of CSE, for example, by locating bugs in the firmware that controls induction hobs from the brands of the BSH group [7]. However, in our preliminary evaluations in the context GSE, the BLiMEA results obtained lower values in the main measures (precision, recall, and F-measure).

We have performed preliminary test of the ideas in which the approach was able to reveal blocking bugs where the boss was indestructible because one weak point overlapped another weak point. Although we do not have enough results to conduct a quantitative evaluation, our intuition indicates that our approach can be highly useful for bug localization in GSE.

Finally, our intuition is that thinking about the lack of fun heuristics can help design better guides to locate bugs in video games. All in all, video game developers seem to be more concerned about bugs that imperceptibly frustrate the player than a null pointer exception.

## V. Conclusion

For years, bug reporting and the defect localization principle have proven to be useful for locating bugs in software. Bug localization in Game Software Engineering has received little attention despite the rise of video games and the problems that their developers have in locating bugs.

Perhaps, bug reports and the defect localization principle are not enough to locate bugs in video games. We explore a novel route to locate bugs in video games by means of evolving video game simulations that produce traces that are relevant to locating bugs. To locate bugs, we want to leverage non-player characters, which is one of the key features that are inherent to the video game domain. Our work could open a novel research direction for bug localization in video games that could also potentially be used in Classic Software Engineering.

## References

[1] SlashData. Global developer population report 2019. https://sdata.me/GlobalDevPop19, 2019. [Online; accessed 21-November-2021].

[2] Luca Pascarella, Fabio Palomba, Massimiliano Di Penta, and Alberto Bacchelli. How is video game development different from software development in open source? In Andy Zaidman, Yasutaka Kamei, and Emily Hill, editors, *Proceedings of the 15th International Conference on Mining Software Repositories, MSR 2018, Gothenburg, Sweden, May 28-29, 2018*, pages 392–402. ACM, 2018.

[3] W. Eric Wong, Ruizhi Gao, Yihao Li, Rui Abreu, and Franz Wotawa. A survey on software fault localization. *IEEE Trans. Software Eng.*, 42(8):707–740, 2016.

[4] Cristiano Politowski, Fábio Petrillo, and Yann-Gaël Guéhéneuc. A survey of video game testing. In *2nd IEEE/ACM International Conference on Automation of Software Test, AST@ICSE 2021, Madrid, Spain, May 20-21, 2021*, pages 90–99. IEEE, 2021.

[5] PC Gamer. How buggy is cyberpunk 2077, really? https://www.pcgamer.com/how-buggy-is-cyberpunk-2077-really/, 2020. [Online; accessed 21-November-2021].

[6] Apostolos Ampatzoglou and Ioannis Stamelos. Software engineering research for computer games: A systematic review. *Information and Software Technology*, 52(9):888–901, 2010.

[7] Lorena Arcega, Jaime Font, Øystein Haugen, and Carlos Cetina. An approach for bug localization in models using two levels: model and metamodel. *Software and Systems Modeling*, 18(6):3551–3576, 2019.

[8] Lorena Arcega, Jaime Font Arcega, Øystein Haugen, and Carlos Cetina. Bug localization in model-based systems in the wild. *ACM Trans. Softw. Eng. Methodol.*, 31(1), oct 2021.

[9] Ahmed E. Hassan and Richard C. Holt. The top ten list: Dynamic fault prediction. In *Proceedings of the 21st IEEE International Conference on Software Maintenance*, ICSM '05, page 263–272, USA, 2005. IEEE Computer Society.

# Improving Search-based Test Case Generation by means of Interactive Evolutionary Computation

Pedro Delgado-Pérez
Dpto. de Ingeniería Informática
University of Cádiz, Spain
pedro.delgado@uca.es

Aurora Ramírez
Dpto. Informática y Análisis Numérico
University of Córdoba, Spain
aramirez@uco.es

Kevin J. Valle-Gómez
Dpto. de Ingeniería Informática
University of Cádiz, Spain
kevin.valle@uca.es

Inmaculada Medina-Bulo
Dpto. de Ingeniería Informática
University of Cádiz, Spain
inmaculada.medina@uca.es

José Raúl Romero
Dpto. Informática y Análisis Numérico
University of Córdoba, Spain
jrromero@uco.es

***Abstract*—This talk summarizes our research efforts towards getting more human-like and competitive results in search-based software testing (SBST). Our ongoing work seeks to enhance automated SBST tools like EvoSuite by incorporating the tester's knowledge and preferences through interactive evolutionary computation. In our envisioned tester-centered interactive search approach, the tester and the evolutionary algorithm cooperate to reach more satisfactory test suites in terms of their detection capability and readability. Here, we present our current ideas and initial steps in this direction, and discuss current challenges.**

## I. Context and motivation

Automated and efficient testing tools have been developed in the last years to guide the generation of effective test suites, thus reducing the costs of manual testing [1]. In this context, search-based software testing (SBST) has been revealed as a fruitful approach, clearly illustrated by the success of EvoSuite [2]. EvoSuite provides several genetic algorithms that can be applied to evolve a population of whole test suites or individual test cases for Java classes. Individuals represent test cases encoded as sequences of method invocations, and are iteratively modified by crossover and mutation operators. The quality of the test cases is assessed by a configurable fitness function that maximizes different coverage targets.

As EvoSuite has gained popularity and maturity, its ability to detect real faults on industrial systems has been analyzed too [3]. This study shows that SBST tools still have some limitations to reveal complex faults due to the difficulty to reach some parts of the code or to generate convenient combinations of input values to detect the fault. Furthermore, other works have evaluated the comprehension of automatically generated test suites and the willingness of testers to accept them [4]. These studies reveal that testers perceive automatic test suites as less helpful than those manually designed, which clearly hamper their possible adoption in industrial contexts. To overcome this, some authors have proposed fully automated techniques to improve the appearance of the final test suite [5], with the ultimate goal of gaining testers' acceptance.

Our vision is that one alternative option to address both issues —detection capability and readability— is to *put the tester in the loop*. Interactive search-based software engineering (iSBSE) [6] proposes showing intermediate results to the engineer and incorporate their feedback into the search process. Designing an effective interactive experience is a hard task, requiring a careful design of the algorithm to prevent fatigue and information overload. Interactive approaches for SBST have only been explored in the context of test data generation for embedded software [7]. Similarly, combining the best of automated tools and the knowledge of testers about the system under test (SUT) seems to be a better approach to take one step further in the test case generation process. We have high expectations on the possibilities that an interactive approach could bring to such process, improving the tester's experience when using SBST tools and fostering a wider acceptance from the software industry.

## II. Interactive opportunities in SBST

The design of an interactive search algorithm involves many decisions that should be adapted to the particularities of the problem domain and the purpose of the interaction. In a recent work [8], we analyzed the general interactive options compiled by the iSBSE review [6], mapping them to the test case generation problem. Figure 1 summarizes the proposed framework to design algorithm variants depending on whether the interaction is focused on improving test case readability, fault detection capability, or both.

Involving the tester in the evaluation could be a proper way to assess the readability of the test cases, while his/her knowledge about the SUT could be exploited to guide the search towards the detection of hard faults. During an interaction, the tester could be asked to evaluate characteristics of the candidate test cases, but also modify their statements and parameters, or choose between similar test cases (e.g., those that detect the same mutant). As for the interaction scheduling, internal information about the evolution could be considered. For instance, the level of achieved coverage can be useful

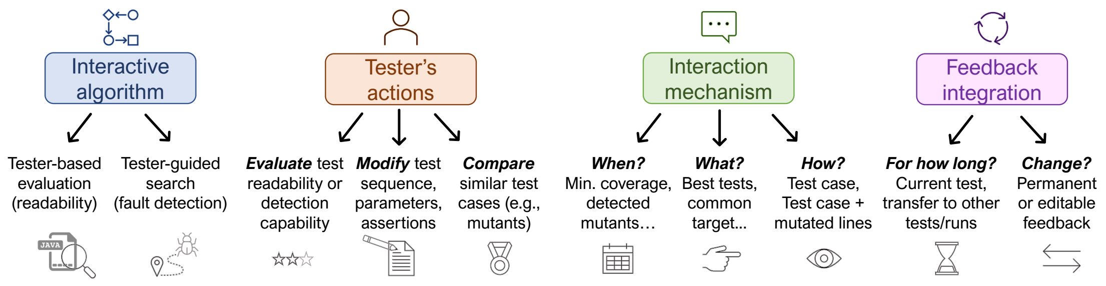

Interactive algorithm
Tester-based evaluation (readability)
Tester-guided search (fault detection)
Tester's actions
Evaluate test readability or detection capability
Modify test sequence, parameters, assertions
Compare similar test cases (e.g., mutants)
Interaction mechanism
When? Min. coverage, detected mutants…
What? Best tests, common target…
How? Test case, Test case + mutated lines
Feedback integration
For how long? Current test, transfer to other tests/runs
Change? Permanent or editable feedback

Fig. 1. Design alternatives for a search-based interactive algorithm to address the test case generation problem.

to avoid premature interactions, and the relation between test cases and targets can be used to guide the selection of test cases to be shown. The tester's feedback —which might be revisited or not— could be applied to the selected test cases, or transferred to other tests in the same or a different run.

## III. Towards interactive readability assessment

Our current efforts are focused on the challenge of improving the readability of the generated test suites based on the tester's preferences. With that purpose in mind, we are designing a new version of EvoSuite that includes an interactive module to pause the search at different points throughout the execution and to incorporate readability assessments of several test cases. At each of those moments, the system is intended to prepare an interaction with the tester, in which:

- Different test cases from the population are selected, all of them covering one of the targets pursued by the search.
- Both the test cases and the target are shown to the tester for his/her revision.
- The system receives the readability assessment made by the tester in the form of a readability score for each test.

The collected scores are intended to be used in the formation of the final test suite, prioritizing the most readable test cases from the tester's perspective. This interactive version will be configurable to set the desired number of interactions and how many tests are selected for revision, among other parameters.

## IV. Challenges and open issues

We are currently facing the following challenges:

*a) Technical challenges:* The interactive module should monitor the state of the evolution to choose the right moments to interact and decide which test cases are worthy revised by the tester. This is clearly conditioned by the inner procedures of EvoSuite, which includes different evolutionary algorithms and performs additional steps to build the final test suite (e.g., removal of redundant tests). If the tester is asked to evaluate readability, a practical ranking scale should be defined. Even more, the incorporation of this information should not deviate the search from its primary objective, i.e., the test suite should not increase code readability at the expense of reducing coverage. Also, the algorithm should be prepared to deal with possible inconsistencies in the tester's feedback.

*b) Experimental challenges:* Choosing the class under test becomes an important decision: while simple classes might be easily covered by EvoSuite, complex ones might result difficult for the human to understand. Another aspect is the difficulty in recruiting participants with the necessary expertise on software testing to empirically validate the interactive approach in a realistic scenario. Also, participants with different testing knowledge will have different perceptions of readability because it is considered a highly subjective concept.

*c) Practicability issues:* Transferring our approach to an industrial setting represents a long-term challenge. Whether this interactive approach could be practical when applied on industrial codebases and could actually serve to bridge the gap between the state of research and practice is still an open issue.

## Acknowledgment

Work supported by the European Commission (FEDER), the Spanish Ministry of Science and Innovation (RTI2018-093608-BC33, RED2018-102472-T, PID2020-115832GB-I00) and the Andalusian Regional Government (DOC_00944).

## References

[1] S. Ali, L. C. Briand, H. Hemmati, and R. K. Panesar-Walawege, "A systematic review of the application and empirical investigation of search-based test case generation," *IEEE Trans. Softw. Eng.*, vol. 36, no. 6, pp. 742–762, 2010.

[2] G. Fraser and A. Zeller, "Mutation-driven generation of unit tests and oracles," *IEEE Trans. Softw. Eng.*, vol. 38, no. 2, pp. 278–292, 2012.

[3] M. M. Almasi, H. Hemmati, G. Fraser, A. Arcuri, and J. Benefelds, "An industrial evaluation of unit test generation: Finding real faults in a financial application," in *Proc. 39th Int. Conf. Software Engineering (ICSE): Software Engineering in Practice Track*, 2017, pp. 263–272.

[4] S. Shamshiri, J. M. Rojas, J. P. Galeotti, N. Walkinshaw, and G. Fraser, "How do automatically generated unit tests influence software maintenance?" in *Proc. 11th Int. Conf. Software Testing, Verification and Validation (ICST)*, 2018, pp. 250–261.

[5] D. Roy, Z. Zhang, M. Ma, V. Arnaoudova, A. Panichella, S. Panichella, D. Gonzalez, and M. Mirakhorli, "DeepTC-Enhancer: Improving the Readability of Automatically Generated Tests," in *Proc. 35th IEEE/ACM Int. Conf. Automated Software Engineering (ASE)*, 2020, pp. 287–298.

[6] A. Ramírez, J. R. Romero, and C. L. Simons, "A systematic review of interaction in search-based software engineering," *IEEE Trans. Softw. Eng.*, vol. 45, no. 8, pp. 760–781, 2019.

[7] B. Marculescu, R. Feldt, R. Torkar, and S. M. Poulding, "Transferring interactive search-based software testing to industry," *J. Syst. Softw.*, vol. 142, pp. 156–170, 2018.

[8] A. Ramírez, P. Delgado-Pérez, K. J. Valle-Gómez, I. Medina-Bulo, and J. R. Romero, "Interactivity in the generation of test cases with evolutionary computation," in *Proc. IEEE Congress on Evolutionary Computation (CEC)*, 2021, pp. 2395–2402.

# Road to Human as the Fitness Function

Jaime Font*, Lorena Arcega*, Francisca Pérez* and Carlos Cetina*
*SVIT Research Group
Universidad San Jorge, Zaragoza, Spain
Email: jfont@usj.es, larcega@usj.es, mfperez@usj.es and ccetina@usj.es

***Abstract*—In Search-Based Software Engineering, more than 100 works have involved the human in the search process to obtain better results. However, the case where the human completely replaces the fitness function remains neglected. There is a good reason for that; no matter how fast or efficient the human is, humans cannot assess millions of candidate solutions in a reasonable time. By contrast, computational intelligence techniques (such as evolutionary computation or machine learning) are capable of dealing with large search spaces but are not as good as humans in understanding the context when making a decision. We propose to combine both, using the Human as the Fitness Function of an evolutionary approach designed to minimize the amount of work that the human has to perform.**

**We are using three ingredients to minimize the effort of the human. Firstly, we focus on Search-Based Model-Driven Engineering (SBMDE) because inspecting models should require less human effort than inspecting code thanks to the higher abstraction level of models compared to the code. Secondly, we apply intelligent operators that do not rely only on randomness to explore the search space but also embed relevant information for the search being performed. Thirdly, we apply techniques to handle unfeasible individuals (models that do not conform to their metamodel) and thus reducing the search space.**

## I. Introduction

Search-Based Software Engineering (SBSE) proposes to reformulate problems from the field of software engineering as search problems that can be addressed by existing meta-heuristic algorithms such as evolutionary algorithms. It has attracted many research efforts from different fields, as only three items are needed to apply SBSE: (1) an encoding to represent candidate solutions to the problem as individuals that can be manipulated; (2) a set of operators used to generate new individuals; (3) a fitness function that can assess how good is each candidate as a solution to the problem. Then, candidate solutions (which are encoded following the representation chosen) are evolved (by applying the operators) and are evaluated (by the fitness function) in an iterative process until optimal solutions to the problem are found.

SBSE has been applied to many different problems [1], and we have successfully applied it to maintenance tasks such as bug localization [2], traceability links recovery [3], or feature location [4]. Feature Location is defined as the task of identifying the set of software elements that realize a particular feature. This can be applied to different software artifacts (e.g.: source code or software models) and the elements identified can vary in granularity (a line of code, a method, or a whole class). In one of our previous works, we tackled the problem of feature location from the software of CAF, an industrial partner that has been producing railway solutions since 1917. We estimated that the time needed by a single engineer to manually locate all the features present in their software products would be over 30 years [3].

Encodings and operators can be reused across domains or applications and adapted with minimum changes to be applied to different problems. The typical encoding used includes binary strings, value encoding, or tree encoding. Similarly, default operations for those encodings can be directly applied, either selection operators, crossover operators, or mutation operators. However, the fitness function is more bounded to the domain and not so easily translated from one domain to another, as it needs to take into account much information about the domain.

We propose to replace the fitness function with a human, taking advantage of the better understanding of the domain and the context of humans compared to computers. However, there is a good reason for not doing it, no matter how efficient the human is, humans cannot evaluate thousands of candidate solutions in reasonable time as they will suffer from fatigue. By contrast, computers are never tired and will surpass humans in tasks like the manipulation of individuals or the exploration of large search spaces. The following sections describe the steps that enabled us to use the Human as the Fitness Function (HaFF).

## II. Related work

Traditionally, SBSE that has been applied to reformulate software engineering tasks into search problems relegated the human to the background and put most of the burden on the search strategy. Some works have attempted to address this by enabling the interaction of the human in the search process, mainly by providing scores to complement the fitness function. As part of our work, we have analyzed existing surveys about search-based software engineering applied to model-driven engineering [5] (covering works from 1998 to 2016) and about interactive search-based software engineering [6] (covering works from 1999 to 2017) and updated them (up to August 2020).

Most of the works (54.55%) involve humans in the selection of solutions. The next most common type of interaction is the modification of the obtained solution (31.82%). Our work is focused on the evaluation (fitness function), only 27.27% of the works involve humans in the evaluation. None of the works have been able to successfully replace the fitness function with a human in an industrial context as our work does.

## III. Approach

To effectively apply HaFF we need to reduce the number of fitness evaluations done by the search strategy. We have explored three ideas to reduce the number of generations needed to reach the optimal solution: (1) the use of software models; (2) the use of intelligent operators; (3) the use of strategies to handle unfeasible individuals.

Applying the concepts of model-driven engineering [7], where models are the cornerstone of the process and source code can be generated directly from the software model, the search space can be reduced. The rise in the abstraction level introduced by the models helps in reducing the magnitude of the problem; however, the search space is still too big to be manually traversed. Using a binary encoding to represent a model that is used to generate the source code of an induction hob yields a search space of more than $10^150$ individuals [8].

By using intelligent operators, we can reduce the number of generations needed by the evolutionary algorithm to reach the optimal solution; thus reducing the number of fitness evaluations performed. In one of our previous works [9], we proposed a set of intelligent operators that leverage the latent semantics of software models instead of applying random mutation and crossover operators. Specifically, the operators proposed are based on query reformulation techniques that we have successfully applied in the past [10]. In our evaluations, the use of these operators significantly reduced the number of generations needed to reach the optimal solution (from 347.685 to 384 on average for the best scenario).

When working with software models, the conformance between models and metamodels must be guaranteed, ensuring that the software model is built using the concepts and following the constraints formalized by its metamodel. However, when traditional genetic operators are applied to individuals encoding software models, the conformance can be easily broken. We have recently proposed nine strategies to handle the nonconforming individuals when applying SBSE to individuals that encode software models [8]. The nine strategies are based on methods proposed in the literature: penalty functions, which reduce the fitness value if constraints are violated; strong encoding, which guarantees by construction that unfeasible individuals cannot be created; closed operators, which always produce new individuals that fulfill the constraints; repair operators, which can turn an unfeasible individual into a feasible one, fixing the problematic parts. Our evaluation shows that the application of these strategies reduced the number of generations needed to reach the optimal solution (from 6405 to 456 on average for the best-performing strategy).

## IV. Results

In a recent work [11], we have applied intelligent operations and individuals encoding software models to reduce the number of fitness evaluations to a level manageable by a human. That is, we replaced the fitness function with a human in a real-world case study of feature location in software models. In the evaluation, 29 software engineers from the industrial partner were involved in acting as the fitness function, assessing 400 individuals each of them.

The results showed that HaFF leads to significantly better results than the best baseline. The results are excellent for all the features obtaining 82.34% in precision, 96.01% in recall, and 88.34% in F-measure. The work includes guidelines on how to apply HaFF to other problems from SBSE.

## V. Conclusion

The fitness function can be replaced by a human, providing better values than the best fitness available in the literature for the problem of feature location [11]. However, this is just the first step, there is still room for improvement: (1) reducing the number of fitness evaluations even more with proper handling of unfeasible individuals should lead to better results (as the human will be able to refine more the individual with the same number of fitness evaluations); (2) there are other benefits from the use of HaFF, the human is less affected to problems with language (as vocabulary mismatch or tacit knowledge [12]) and we would like to empirically evaluate it; (3) there are more opportunities of synergy emerging from the involvement of the human as fitness, we want to explore the collaboration between more than one human to be the fitness and the possibility of hybrid fitness that only ask the human from time to time.

## References

[1] M. Harman, S. A. Mansouri, and Y. Zhang, "Search-based software engineering: Trends, techniques and applications," *ACM Comput. Surv.*, vol. 45, no. 1, pp. 11:1–11:61, Dec. 2012.

[2] L. Arcega, J. Font, Ø. Haugen, and C. Cetina, "An approach for bug localization in models using two levels: model and metamodel," *Softw. Syst. Model.*, vol. 18, no. 6, pp. 3551–3576, 2019.

[3] F. Pérez, R. Lapeña, J. Font, and C. Cetina, "Fragment retrieval on models for model maintenance: Applying a multi-objective perspective to an industrial case study," *IST*, vol. 103, pp. 188–201, 2018.

[4] J. Font, L. Arcega, Ø. Haugen, and C. Cetina, "Achieving feature location in families of models through the use of search-based software engineering," *IEEE Transactions on Evolutionary Computation*, vol. PP, no. 99, pp. 1–1, 2017.

[5] I. Boussaïd, P. Siarry, and M. Ahmed-Nacer, "A survey on search-based model-driven engineering," *Automated Software Engineering*, vol. 24, no. 2, pp. 233–294, Jun 2017.

[6] A. Ramírez, J. R. Romero, and C. L. Simons, "A systematic review of interaction in search-based software engineering," *IEEE Transactions on Software Engineering*, vol. 45, no. 8, pp. 760–781, 2019.

[7] S. Kent, "Model driven engineering," in *Integrated Formal Methods*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 286–298.

[8] J. Font, L. Arcega, Ø. Haugen, and C. Cetina, "Handling nonconforming individuals in search-based model-driven engineering: nine generic strategies for feature location in the modeling space of the meta-object facility," *Softw. Syst. Model.*, vol. 20, no. 5, pp. 1653–1688, 2021.

[9] F. Pérez, T. Ziadi, and C. Cetina, "Utilizing automatic query reformulations as genetic operations to improve feature location in software models," *IEEE Transactions on Software Engineering*, no. 01, pp. 1–1, jun 2020.

[10] F. Pérez, J. Font, L. Arcega, and C. Cetina, "Collaborative feature location in models through automatic query expansion," *Autom. Softw. Eng.*, vol. 26, no. 1, pp. 161–202, 2019.

[11] F. Pérez, J. Font, L. Arcega, and C. Cetina, "Empowering the human as the fitness function in search-based model-driven engineering," *IEEE Transactions on Software Engineering*, no. 01, pp. 1–1, oct 2021.

[12] C. Cetina, J. Font, L. Arcega, and F. Pérez, "Improving feature location in long-living model-based product families designed with sustainability goals," *J. Syst. Softw.*, vol. 134, pp. 261–278, 2017.

# GitHub Actions Adoption Among Projects, What Are The Best Practices?

Ali Khatami
*Software Engineering Research Group*
*Delft University of Technology*
*Delft, The Netherlands*
s.khatami@tudelft.nl

***Abstract*—GitHub Actions (GHA) is a fast-growing automation service for software workflows on GitHub, and is becoming the dominant CI service in open source software development. However, developers lack assistive tools, and suggested best practices when configuring GHA's workflows. To address this challenge, this paper will focus on studying: the GHA adoption across different programming languages, kind of tasks that are mostly automated, and common configuration patterns. The results will enable us to come up with a set of best practices for GHA adoption.**

**Next step following this study would be building recommendation tools to assist developing GHA workflows for software projects on GitHub. The common patterns discovered in this study can help to build such recommenders.**

**I am eager to discuss and brainstorm the new ideas and challenges related to this work with the participants of the SMILESENG Summer School.**

## I. Introduction

To ensure the quality of software systems, software engineers use a range of quality assurance approaches, e.g., software testing, modern code review, automated static analysis, and continuous integration. We first studied these practices to have an overview of the state-of-the-practice in quality assurance in open-source software development.[1] This study led to insights into the prevalence of quality assurance approaches in Open Source Software (OSS) projects, in isolation, and in conjunction. Moreover, we discovered challenges that projects face for each of these practices, which then inspired novel ideas for future research. One of the highlighted challenges is lack of best practices for adopting GitHub Actions[2] (GHA). GHA is an automation service for software workflows on GitHub, and is becoming the dominant CI service on GitHub [1]. Despite the rapid growth, there is not enough studies to investigate best practices of adopting GHA, and evaluate different approaches of using it.

Projects have different strategies when configuring GHA workflows. During this configuration, developers need to set several components, which contain various options [2]. Thus, it is difficult for a developer with less experience to configure GHA workflows [2]. To address this challenge an idea is to create tools to assist developers when configuring the workflows [3], [2]. Yet, these assistive tools can depend on the Programming Language (PL) specific properties, that a project use. As an example, it is possible to define testing phase after build phase in a Maven *'pom.xml'* configuration file of a Java project, but doing the exact same thing it not as similarly possible for a Python project. In other words, the development ecosystem differs from one PL to another, which also makes configuration of workflows different.

Accordingly, studying the state of the practice in GHA adoption, with a focus on the PL specific properties is of importance. Besides, finding common patterns used in workflows configuration is another idea of addressing the challenge. Particularly, best practices can be the same accepted common patterns of configuration that occur repeatedly.

## II. Related Works

Kinsman *et al.* have found developers have a positive reception of GHA by manually analyzing GHA related discussions on GitHub. They also have reported a 0.7% adoption of GHA among around 400K repositories mainly for continuous integration, utilities, and deployment purposes. Moreover, they have investigated the impact of GHA adoption on projects and found that the number of commits of merged pull requests decreases, and the number of monthly rejected pull requests increases after the adoption of GHA [4].

Also, Golzadeh *et al.* have conducted a longitudinal study focused on the evolution of CI on GitHub projects. By studying CI co-usage and migration of around 90K GitHub repositories, and considering 7 different CI platform including GHA, they have found Travis-CI and GHA are the predominant CIs. Moreover, they mentioned GHA is becoming the most dominant CI platform and is used in more repositories than Travis-CI. Particularly, they have observed a fall in Travis-CI usage due to migration to GHA [1].

In another work, Chen *et al.* have focused on the state of the practice in using GHA, and reported a 22.5% adoption of GHA among their set of 27K popular repositories. Also, they have studied the correlation between GHA usage and properties of a project to see its impacts. Moreover, they have looked at the common specifications that projects use when configuring their GHA workflows. After all, they have highlighted the challenge of configuring workflows, difficulty

[1]A. Khatami, and A. Zaidman, "State-Of-The-Practice in Quality Assurance in Open Source Software Development", Submitted to Journal of Systems and Software, May 23rd, 2022.

[2]https://github.com/features/actions, last visited 3rd June 2022. GitHub Actions was made public on 13th November 2019 on GitHub.

of building commonly used action sequences, need of assistive tools to configure GHA, and the latency of communication among jobs through the artifacts [2].

Valenzuela-Toledo *et al.* have studied the modifications on workflows' configuration. By manual inspection of 222 commits on GHA workflows of 10 software projects, they have highlighted the need of tools for workflow creation/edition, identification of syntax errors, and recommendation of specific common task [3].

## III. Proposed Research Questions

To reach the goal of this research, I will focus on answering the following research questions:

**RQ1 How GHA is adopted across different programming languages?**
There has been studies that looked at the porpotion of GHA usage among different projects with different PLs [4], [2]. But none looked deeper to see if there is any PL specific characteristic in GHA adoption. Consequently, to answer this research question I want to see how projects make use of GHA by looking at the type of actions they use, number/kinds of jobs/workflows, number of steps in each job, and the relation between jobs in a workflow. More specifically, I would like to see how GHA adoption differs in Java projects that already benefit from Maven or Gradle, and other projects that do not have such build automation tools.

**RQ2 What kind of tasks are mostly automated by GHA?**
Other related works have only looked at the actions and did a categorization of actions used in projects' workflows [4], [2]. However, I aim to see what tasks are automated, how they are automated, and whether they are using the same actions for the same purposes? If they are not, what are the reasons?

**RQ3 What are the common patterns in GHA workflows configuration?**
Common patterns of workflow configurations are of high importance, since they can be recommended to other projects with similar properties. I want to look deeper into these configurations to find the common patterns. As an example, obviously *"actions/checkout"* is the most commonly used action among projects [4], [2] because they all need to "checkout" the changes before running any kind of actions on the repository. Hence, I want to be more focused at task level configurations that contain many actions, and also the relation among those actions.

## IV. Building the Dataset

As an empirical study, the first step is to build the initial dataset of selected projects. Similar to Chen *et al.* [2], I will start with a set of popular projects on GitHub (using GitHub Search (GHS) dataset [5]) and then filter the ones using GHA workflows by checking if they have any files with YAML format in the *"./github/workflows"* directory.

Based on the proposed RQ1, the next step would be choosing the top PLs using GHA. Again according to Chen *et al.* [2]'s study, JavaScript, Python, Go, TypeScript, and Java are among the top PLs using with more than 500 repositories for each of them.

Other steps include: cloning projects, extracting their YAML configuration files, and parsing them to obtain detailed well formatted configuration data. This data will be used in the analysis part to answer the research questions.

## V. Preliminary Results

This work is still in the stage of brainstorming and exploring new ideas, so there are no preliminary results available yet. But, if obtained before the time of the live talk, will be presented at the SMILESENG Summer School.

## VI. Conclusion

In this short paper, we saw the importance of studying GitHub Actions (GHA) workflows to find the so-called best practices in configuring software workflows on GitHub. Also, we saw there is a need of assistive tools for software developers designing these workflows.

As a first step towards building the assistive tools, and proposing the best practices, I mentioned three research questions, and briefly explained what are the benefits of answering them. Moreover, I mentioned a few steps of the data collection step of the study.

I am excited to see how participants of the SMILESENG Summer School find this research idea, receive feedbacks, and discuss each of the steps of conducting it.

## References

[1] M. Golzadeh, A. Decan, and T. Mens, "On the rise and fall of CI services in GitHub," in *29th IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2022.

[2] T. Chen, Y. Zhang, S. Chen, T. Wang, and Y. Wu, "Let's supercharge the workflows: An empirical study of github actions," in *21st IEEE International Conference on Software Quality, Reliability and Security, QRS 2021 - Companion, Hainan, China, December 6-10, 2021*. IEEE, 2021, pp. 1–10. [Online]. Available: https://doi.org/10.1109/QRS-C55045.2021.00163

[3] P. Valenzuela-Toledo and A. Bergel, "Evolution of github action workflows," in *29th IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2022.

[4] T. Kinsman, M. S. Wessel, M. A. Gerosa, and C. Treude, "How do software developers use github actions to automate their workflows?" in *18th IEEE/ACM International Conference on Mining Software Repositories, MSR 2021, Madrid, Spain, May 17-19, 2021*. IEEE, 2021, pp. 420–431. [Online]. Available: https://doi.org/10.1109/MSR52588.2021.00054

[5] O. Dabic, E. Aghajani, and G. Bavota, "Sampling projects in GitHub for MSR studies," in *International Conference on Mining Software Repositories (MSR)*. IEEE, 2021, pp. 560–564.

# Traceability Links Recovery in BPMN Models through Evolutionary Learning to Rank

Raúl Lapeña, Ana Marcén, Jaime Font, and Carlos Cetina
SVIT Research Group
Universidad San Jorge
Villanueva de Gállego, Spain
Email: [rlapena,acmarcen,jfont,ccetina]@usj.es

***Abstract*—Traceability Links Recovery (TLR), defined as the software engineering task that deals with the automated identification of dependencies between software artifacts, is a key to success in the scene of industrial software, and has been a subject of fundamental and applied investigation for many years within the software engineering community. Most TLR techniques perform traceability based on the linguistic clues of the software artifacts under study, causing BPMN models to pose an additional challenge for TLR, since they tend to contain less textual information than other artifacts. Over the past few years, we have studied TLR between natural language requirements and Model Driven Development (MDD) models through an Evolutionary Learning to Rank approach (ELtoR), retrieving traceability links through the combination of evolutionary computation and machine learning techniques, outperforming five other TLR approaches. One of the reasons behind the improvements is that ELtoR is not as dependent on the linguistic clues of the artifacts as the other TLR approaches. Our hypothesis is that ELtoR can be used to improve the state of the art in TLR between requirements and BPMN models. Through this communication, we report new ideas on how to adapt ELtoR and the necessary encodings to work over BPMN models, plus our expectations regarding the outcomes of evaluating the approach in an industrial scenario.**

## I. Introduction

Traceability Links Recovery (TLR) is an important support activity for development, management, and maintenance of software, and is considered as a good practice by numerous major software standards [1]. Affordable TLR can be critical to the success of a project [2], and leads to increased maintainability and reliability of software systems [3], also decreasing the expected defect rate in developed software [4]. However, establishing and maintaining traceability links has proven to be a time consuming, error prone, and person-power intensive task [1]. Therefore, automated TLR has been a subject of investigation for many years within the software engineering community. In recent years, it has been attracting more attention, becoming a subject of both fundamental and applied research [5].

Software engineers from our industrial partner, an international manufacturer in the railway domain, express system requirements in natural language, and use them to design BPMN models [6]. The BPMN models are used to describe the interactions that occur between the humans and the trains, and to design and derive other software artifacts. State-of-the-art automated TLR techniques rely greatly on the language and the syntactical, lexical, and semantical particularities of the software artifacts under study. For instance, Latent Semantic Indexing (LSI), which is the most popular TLR technique and the one that has yielded the best TLR results so far [7], is based on exploiting term similarities among the requirements and the software artifacts. BPMN models tend to present less terms and an overall lack of textual information in comparison to other artifacts. Since TLR techniques rely on the textual components of the artifacts under study, TLR becomes an ever harder task when performing TLR directly among requirements and BPMN models.

TLR-ELtoR is a Traceability Links Recovery (TLR) approach that is based on an Evolutionary Algorithm and a Learning to Rank technique (ELtoR). The results obtained through this approach indicate that TLR-ELtoR may be a better alternative than LSI or other approaches when the software artifacts are incomplete or do not have much textual content [8]. This work is our first step in adapting TLR-ELtoR for its application on BPMN models taking into consideration the particularities of this kind of model [9]. We also report our expectations regarding the outcomes of evaluating the approach in an industrial real-world scenario.

## II. Related Work

Most of the existing works focus on Traceability Link Recovery between requirements and source code. CERBERUS [10] provides a hybrid technique that combines information retrieval, execution tracing, and prune dependency analysis allowing the tracing of requirements to source code. Eaddy et al. [11] present a systematic methodology for identifying which code is related to which requirement, and a suite of metrics for quantifying the amount of crosscutting code. Some other works target the TLR tasks on models. De Lucia et al. [12] present a Traceability Link Recovery method and tool based on LSI in the context of an artifact management system, which includes models. In contrast, this work does not focus on source code or MDD models. Rather, we propose new ideas on how to adapt the TLR-ELtoR approach to work over BPMN models taking their particularities into account.

## III. Approach

The ELtoR approach is based on an Evolutionary Algorithm that relies on genetic operations and a fitness function to provide the model fragment from a given model that realizes a specific requirement. The approach receives as input the model that implements a specific requirement. An evolutionary algorithm then iterates over a population of model fragments, evolving them using genetic operations. Finally, the score of each model fragment and its position in the ranking are calculated through a fitness function that uses LtoR as its objective. As output, the approach provides a model fragment ranking where each model fragment is ranked taking into account how well the model fragment implements the input requirement. ELtoR has three steps:

1) Initialization: generation of a population of model fragments from the model, which serves as input for the evolutionary algorithm.
2) Genetic operations: genetic operations generate candidate model fragments for the target requirement.
3) Fitness function: the new model fragment population is evaluated through the fitness function.

The last two steps of the approach are repeated until the solution converges to a certain stop condition. When the stop condition is met, the evolutionary algorithm provides a model fragment list, ranked according to the objectives for the requirement.

## IV. Preliminary Results

The fitness function of the TLR-ELtoR approach is based on LtoR algorithms, which are machine learning algorithms that automatically address ranking tasks. Specifically, the LtoR algorithms make it possible to build a classifier that contains a set of rules to rank objects. In our approach, the classifier is used to determine how well each model fragment realizes a specific requirement. If the classifier is not properly trained, the approach cannot determine which model fragment is the best solution for a specific requirement. Therefore, the classifier is a critical element in our approach. However, the training of the classifier is not a simple task.

First, a classifier is trained using a knowledge base. This knowledge base contains a set of examples used for learning, or in other words, to train the classifier. In our case, it must contain a set of links between requirements and model fragments. The specific particularities of the model fragments can have an impact on the training, so using a representative knowledge base for the training is very important. In BPMN models, there are many model elements with little or no text, but several language patterns that can be used to link the requirements with the models [9]. Therefore, the examples (model fragments) must contain different amounts of text and also sample all the possible identified patterns.

Second, the examples in the knowledge base must be encoded as feature vectors to apply LtoR algorithms. In [13], we described three different encodings for MDD models, but these encodings are based on texts, that is, on counting the occurrences of a specific term. We can adapt the encodings to count the occurrences of an element type or pattern for the BPMN models.

Third, machine learning algorithms have different capabilities. For example, in [8], we used Rankboost because this LtoR algorithm can benefit from a small knowledge base together with a small number of features in the encoding to reduce the overfitting problem [14]. However, due to the lack of text in BPMN models, we expect to need a larger knowledge base and other machine learning techniques such as neural networks.

## V. Conclusions

Traceability Links Recovery (TLR) is a key to success in the scene of industrial software. BPMN models pose an additional challenge for TLR due to their lack of text. Our hypothesis is that our Evolutionary Learning to Rank approach (ELtoR), which is not as dependent on the linguistic clues of the artifacts as the other TLR approaches, can be used to improve the state of the art in TLR between requirements and BPMN models. So far, we have managed to transport the encoding to BPMN models. Our current challenge lies on adapting the training process and knowledge base.

## References

[1] R. Oliveto, M. Gethers, D. Poshyvanyk, and A. De Lucia, “On the Equivalence of Information Retrieval Methods for Automated Traceability Link Recovery,” in *2010 IEEE 18th International Conference on Program Comprehension*. IEEE, 2010, pp. 68–71.

[2] R. Watkins and M. Neal, “Why and How of Requirements Tracing,” *IEEE Software*, vol. 11, no. 4, pp. 104–106, 1994.

[3] A. Ghazarian, “A Research Agenda for Software Reliability,” *IEEE Reliability Society 2009 Annual Technology Report*, 2010.

[4] P. Rempel and P. Mäder, “Preventing Defects: the Impact of Requirements Traceability Completeness on Software Quality,” *IEEE Transactions on Software Engineering*, vol. 43, no. 8, pp. 777–797, 2017.

[5] R. M. Parizi, S. P. Lee, and M. Dabbagh, “Achievements and Challenges in State-of-the-Art Software Traceability between Test and Code Artifacts,” *IEEE Transactions on Reliability*, vol. 63, no. 4, pp. 913–926, 2014.

[6] M. Chinosi and A. Trombetta, “BPMN: An Introduction to the Standard,” *Computer Standards & Interfaces*, vol. 34, no. 1, pp. 124–134, 2012.

[7] J. Rubin and M. Chechik, “A Survey of Feature Location Techniques,” in *Domain Engineering*. Springer, 2013, pp. 29–58.

[8] A. C. Marcén, R. Lapeña, Ó. Pastor, and C. Cetina, “Traceability link recovery between requirements and models using an evolutionary algorithm guided by a learning to rank algorithm: Train control and management case,” *Journal of Systems and Software*, vol. 163, p. 110519, 2020.

[9] R. Lapeña, F. Pérez, C. Cetina, and O. Pastor, “Leveraging BPMN particularities to improve traceability links recovery among requirements and BPMN models,” *Requirements Engineering*, pp. 1–26, 2021.

[10] M. Eaddy, A. V. Aho, G. Antoniol, and Y.-G. Guéhéneuc, “Cerberus: Tracing Requirements to Source Code Using Information Retrieval, Dynamic Analysis, and Program Analysis,” in *ICPC 2008 conference*. IEEE, 2008, pp. 53–62.

[11] M. Eaddy, A. Aho, and G. C. Murphy, “Identifying, Assigning, and Quantifying Crosscutting Concerns,” in *Proceedings of the First International Workshop on Assessment of Contemporary Modularization Techniques*, 2007, p. 2.

[12] A. de Lucia *et al.*, “Enhancing an Artefact Management System with Traceability Recovery Features,” in *Proceedings of the 20th IEEE International Conference on Software Maintenance*. IEEE, 2004, pp. 306–315.

[13] A. C. Marcén, F. Perez, O. Pastor, and C. Cetina, “Enhancing software model encoding for feature location approaches based on machine learning techniques,” *Software and Systems Modeling*, vol. 21, no. 1, pp. 399–433, 2022.

[14] Z.-H. Zhou and J. Feng, “Deep Forest: Towards an Alternative to Deep Neural Networks,” *arXiv preprint arXiv:1702.08835*, 2017.

# Active Learning-driven Testing of Web APIs

A. Giuliano Mirabella
*SCORE Lab, I3US Institute*
*Universidad de Sevilla*
Seville, Spain
amirabella@us.es

***Abstract*—Automated test case generation for web APIs is a thriving research topic. Most approaches in this domain follow a black-box approach, where test cases are randomly derived from the API specification. These techniques show promising results, but they neglect constraints among input parameters (so-called *inter-parameter dependencies*), as these cannot be formally described in current API specification languages. As a result, when testing real-world services, most randomly generated API requests (i.e., test cases) are invalid since they violate some of the inter-parameter dependencies of the service, making human intervention indispensable. In this paper, an active learning-based method is proposed to efficiently train a classifier to predict the validity of requests before invoking the API, so that invalid requests can be discarded in advance. This strategy minimises the data required to learn during testing, making the approach affordable in practice. Our technique learns as it generates test cases, so that the percentage of valid calls progressively increases up to 90% in commercial APIs such as GitHub and Stripe. More importantly, the number of detected failures is three times grater than a fuzzing baseline. These results show the potential of artificial intelligence to improve current test case generation techniques achieving an unprecedented level of automation.**

## I. Introduction

RESTful Web APIs (also called REST APIs) [1] are the de facto standard for Web integration. These APIs expose a uniform interface through which data and services can be accessed via HTTP interactions. A common phenomenon in REST APIs is that they exhibit inter-parameter dependencies (or simply "dependencies"), i.e, constraints between two or more input parameters that must be met to form valid service calls. For example, in the Google Maps API, when searching for locations, if the `location` parameter is used, the `radius` parameter must also be used, otherwise an error is returned (status code 400, “Bad Request”). Likewise, when querying the GitHub API [2] to retrieve the authenticated user's repositories, the optional parameters `type` and `visibility` must not be used together in the same API request, otherwise an error will be returned. A recent study [3] revealed that these dependencies are very common: they appear in 4 out of 5 APIs, across all application domains and types of operations. Unfortunately, current API specification languages, such as OpenAPI Specification (OAS) [4], do not support the formal description of such dependencies.

In a previous study [5] we showed that, using machine learning techniques, a classification algorithm can be trained to predict the validity of an API call, i.e. whether it satisfies all API dependencies or not, thus avoiding unnecessary API requests. This approach is efficient (it is fully automatic) and effective (it generates a high number of valid requests), but requires a sufficiently varied and balanced training set (dataset). Such a dataset is costly to achieve with current testing techniques, as it involves the generation of many requests and the consequent expenditure of API resources (with the number of allowed API calls often being a limiting factor). The aim of this work is to propose a technique for the efficient collection of a dataset of requests and responses, maximizing the learning of the request classifier, and therefore incurring the minimum possible number of API calls.

## II. Active Learning

A supervised learning system must be trained with hundreds or thousands of labeled observations. There are cases where these observations have a minimal cost (e.g., the rating that users give to movies on a website), but in other cases obtaining the labels may involve a non-negligible cost in terms of time, money or resources. This is the case here, where tagging training calls as valid or invalid requires invoking APIs hundreds or thousands of times.

Active learning (AL) is a branch of machine learning whose key idea is that the learning algorithm can actively choose the observations it learns from [6]. Higher accuracy can be achieved with fewer training labels if the algorithm is allowed to interactively query a source of information, called an oracle, to label new observations with the correct outputs [6]. For example, transcribing an audio to text can take up to ten times longer than the original audio, and requires trained linguists. In the AL paradigm, it is the algorithm that actively asks the linguist to transcribe certain audios, and then train on them [7].

## III. Approach

The goal of this work is to maximise the percentage of valid calls sent to the API fully automatically. For this purpose, we propose a technique based on AL that allows to collect a training set in an optimal way. This technique consists of two phases: start and learning (Figure 1).

### A. *Start*

The process starts with the generation of $n$ random requests, which are executed by invoking the API and from which the responses are collected. Those requests that received a 2XX status code are labeled as valid, and those that received a 4XX are labeled as invalid (this happens only when a dependency

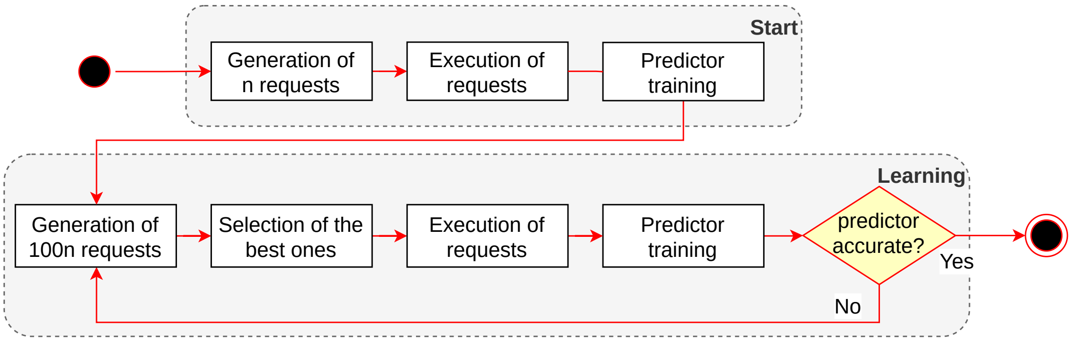


Fig. 1. Approach diagram.

is violated: we rule out the possibility that errors are due to individual values using predefined data dictionaries). Finally, the predictor is trained on these observations.

### B. Learning

As long as the accuracy of the predictor is less than a configurable threshold, active learning is performed by iterations. In each iteration, $100n$ random requests are generated. From these, the $n$ requests with the highest *uncertainty* are chosen, where *uncertainty* is greater the closer the probabilities of the request being valid or invalid are. For example, of the requests shown in the Table, the one with the highest uncertainty is number 3, since the probabilities of being valid ($P_v$) and invalid ($P_i$) are very similar (0.49 and 0.51, respectively). Finally, the API is invoked with the chosen requests, the requests are labeled and the predictor is retrained, obtaining another accuracy value.

TABLE I
EXAMPLE OF THREE UNLABELED REQUESTS.

| ID | type | visibility | affiliation | sort | direction | $P_v$ | $P_i$ |
|---|---|---|---|---|---|---|---|
| 1 | - | 'all' | 'collaborator' | - | 'asc' | 0.9 | 0.1 |
| 2 | 'private' | - | 'owner' | 'created' | - | 0.3 | 0.7 |
| 3 | 'public' | - | 'collaborator' | - | - | 0.49 | 0.51 |

## IV. EVALUATION

We evaluated the proposed technique on two commercial API operations: reading repositories on GitHub and creating a product on Stripe. For each operation, we generated 2000 requests and tried to maximise the number of valid ones (those getting a status code 2XX). The table shows the percentage of valid requests obtained with our technique (AL) versus those obtained with random techniques (Random). The percentage of valid requests increases a lot with our proposal, reaching a $\sim 98\%$. Thanks to this, we are able to detect up to three times more errors (*failures*) of non-conformity with the specification. The proposal has been implemented in Python with `pandas` and `scikit-learn`, and the classification technique used is the Random Forest [8].

TABLE II
EXPERIMENTAL RESULTS.

| API | Valid requests (%) | | Failures | |
|---|---|---|---|---|
| | Random | AL | Random | AL |
| GitHub | 62.10 | 98.65 | 313 | 1159 |
| Stripe | 55.75 | 99.30 | 104 | 284 |
| Media | 58.93 | 98.98 | 209 | 722 |

## V. CONCLUSIONS

In this article we present a new technique for the automatic generation of more efficient test cases in REST APIs. Preliminary results show that the technique, based on active learning, achieves up to a $\sim 98\%$ of valid requests, compared to the $\sim 59\%$ achieved by random techniques, in commercial APIs from GitHub and Stripe, tripling the number of errors detected and demonstrating the potential of artificial intelligence for the automatic generation of test cases in REST APIs.

## REFERENCES

[1] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," Ph.D. dissertation, 2000.

[2] "GitHub API," accessed January 2020. [Online]. Available: https://developer.github.com/v3/

[3] A. Martin-Lopez, S. Segura, and A. Ruiz-Cortés, "A Catalogue of Inter-Parameter Dependencies in RESTful Web APIs," in *International Conference on Service-Oriented Computing*, 2019, pp. 399–414.

[4] "OpenAPI Specification," accessed April 2020. [Online]. Available: https://www.openapis.org

[5] A. G. Mirabella, A. Martin-Lopez, S. Segura, L. Valencia-Cabrera, and A. Ruiz-Cortés, "Deep Learning-Based Prediction of Test Input Validity for RESTful APIs," in *International Workshop on Testing for Deep Learning and Deep Learning for Testing*, 2021.

[6] B. Settles, "Active learning literature survey," University of Wisconsin–Madison, Computer Sciences Technical Report 1648, 2009.

[7] X. Zhu, J. Lafferty, and R. Rosenfeld, "Semi-supervised learning with graphs," Ph.D. dissertation, USA, 2005, aAI3179046.

[8] T. K. Ho, "Random decision forests," in *Proceedings of 3rd international conference on document analysis and recognition*, vol. 1. IEEE, 1995, pp. 278–282.

# Automatizing Software Cognitive Complexity Reduction: What is Next?

Rubén Saborido, Javier Ferrer, Francisco Chicano
ITIS Software, University of Málaga, Spain
Email: rsain@uma.es, jferrer@uma.es, chicano@uma.es

***Abstract*—As software increases in complexity, developers spend more time fixing bugs or making code work rather than designing or writing new code. Thus, improving software understandability and maintainability would translate into an economic relief over the total cost of a project. However, reducing the complexity of a piece of code is not straightforward. Recently, we modeled software cognitive complexity reduction as an optimization problem and we proposed an approach to assist developers on this task. This approach enumerates sequences of Extract Method refactoring operations until a stopping criterion is met. As result, it returns the minimal sequence of Extract Method refactoring operations found that is able to reduce the cognitive complexity of a method to the given threshold. We evaluated our approach over 10 open-source software projects and was able to fix 78% of the 1,050 existing cognitive complexity issues reported by SonarQube. However, we uncover some limitations and interesting open questions that need further discussion and future work.**

## I. Introduction

In 2017, a novel cognitive complexity metric has been proposed and integrated in the well-known static code tools SonarCloud[1] and SonarQube[2], an open-source service and platform, respectively, for continuous inspection of code quality. This cognitive complexity metric, which we refer to as **SonarSource Cognitive Complexity (SSCC)**, has been defined as a measure of how hard the control flow of a method is to understand and maintain [1]. It breaks from the practice of using mathematical models to assess software maintainability. The SSCC is given by a positive number which is increased every time a control flow sentence appear. Their nested levels also contribute to the SSCC of a method. Although SonarQube suggests to keep methods' cognitive complexity no greater than 15, software developers lack support to reduce the SSCC of their code to this threshold. Recently, we modelled the reduction of the SSCC to a given threshold as an optimization problem where the search space contains all feasible sequences of extract method refactoring opportunities [2]. Here, we summarize this research line and conclude with open questions for future work.

The remainder of this paper is organized as follows. Sec. II introduces the challenge of reducing SSCC of a method. Sec. III summarizes a recently proposed approach to reduce methods SSCC. Sec. IV shows the results of the validation of this approach over 10 open-source Java projects. Sec. V concludes with open research gaps and future work.

[1]https://www.sonarcloud.org/
[2]https://www.sonarqube.org/

## II. Background

The SSCC at method level can be reduced to a threshold applying Extract Method refactoring operations: extracting as a new method in the same class sequences of sentences (i.e., lines of code). However, this task is not straightforward because (i) more than one Extract Method refactoring could be needed to reduce the SSCC of a method, (ii) there are Extract Method refactorings which are not applicable in practice, and (iii) Extract Method refactoring opportunities are bounded by $r = \binom{n}{2} = \frac{n\cdot(n-1)}{2}$, where $n$ is the number of sentences of the method. This is the combination of $n$ sentences taken two at a time without repetition. These two sentences determine the beginning and ending of a code extraction. Thus, the main problem developers face is to find sequences of code extractions that reduce the SSCC of the method. Nevertheless, one would need to evaluate all possible sequences of Extract Method operations, totaling up to $2^r$ alternatives.

## III. Approach

We recently proposed a SSCC reducer approach, consisting in a solver method that takes as input the path to the software project to process and the cognitive complexity threshold ($\tau$). Then, for each method with SSCC greater than $\tau$, it searches for sequences of applicable Extract Method refactoring operations. In order to perform this task, our approach generates the corresponding Abstract Syntax Tree (AST) of the method. Second, it parses the AST and annotates different properties[3] in each node. Third, it processes the annotated AST to obtain Extract Method refactoring opportunities. Once the approach identifies Extract Method refactoring opportunities, it checks if the extractions are applicable. This is done with the help of refactoring tools which are able to check pre-conditions, post-conditions, and apply the corresponding operation over the source code.

We have developed a Java SSCC reducer tool as an Eclipse application. This provides the necessary means for generating an Eclipse product that can be run from the operating system

[3]These are used to compute the SSCC of extracted methods.

command-line as a standalone executable, without the need for opening Eclipse for running. This is particularly useful if, for instance, one needs to integrate it in their current development workflow (e.g., using continuous integration). The tool uses the Extract Method refactoring operation provided by the Java Development Toolkit (JDT) of Eclipse to test the feasibility of code extractions programmatically. Finally, the tool chooses the best sequence of method extractions found during the search: the one that reduces the SSCC to (or below) the threshold and minimizes the number of method extractions.

## IV. Preliminary Results

We conducted a study to evaluate the proposed approach when reducing the SSCC of methods in 10 open source projects from GitHub: two popular frameworks for multi-objective optimization, five platform components to accelerate the development of smart solutions, and three popular open-source projects with more than 10,000 stars and forked more than 900 times. In total, these projects have 1,050 cognitive complexity issues. The proposed approach was able to fix, on average, 78% of the cognitive complexity issues on these projects, taking almost 20 hours to process all methods on the 10 software projects under study. Most studied open-source projects required more than one code extraction to reduce the SSCC of their methods to 15. However, five projects required five or more code extractions to reduce the SSCC of some methods. In general, code extractions reduced, on average, SSCC by 12 units. Nevertheless, some code extractions reduced methods SSCC up to 72 units.

## V. Conclusion

We formulated the reduction of software cognitive complexity provided by SonarCloud and SonarQube, to a given threshold, as an optimization problem. We then proposed an approach to reduce the cognitive complexity of methods in software projects to the chosen threshold though the application of sequences of Extract Method refactoring operations. We also conducted some experiments in 10 open-source software projects, analyzing more than 1,000 methods with a cognitive complexity greater than the default threshold suggested by SonarQube (15). The proposed approach was able to reduce the cognitive complexity to or below the threshold in 78% of those methods.

Despite the obtained results, there are several open questions to further discuss:

- It could exist multiple optimal Extract Method refactoring operations when reducing the cognitive complexity of a method, each of them impacting the code differently: number of resulting and/or extracted lines of code (LOC), number of arguments in the signature of extracted methods, SSCC reduction, SSCC of new extracted methods, etc.). *Which one should we apply?* This opens the door to multiple criteria decision-making.

- An aspect that is out of the scope of this article is the choice of the name for the new extracted methods. Given that the name of new methods can influence the understanding of the resulting source code, *how can we name new extracted methods?* Creating a dictionary with keywords in the original method and using natural language processing techniques with Transformers could be a good starting point to handle this fact.

- Having too many `return`, `break`, and `continue` statements in a method decreases the method's essential understandability. This happens because the flow of execution is broken each time one of these statements is encountered. This fact could prevent the extraction of the code, making an instance of the cognitive complexity reduction problem unsolvable. *Can we pre-process a method to refactor `return`, `break`, and `continue` statements in order to favor the cognitive complexity reduction task?*

- Enumeration algorithms used so far in the proposed approach could fail to scale with the code size because the number of refactoring plans can grow exponentially with the number of lines of code. We believe that modeling SSCC reduction as an Integer Linear Programming optimization problem makes sense. This would make it feasible to apply efficient solvers, like CPLEX, to get optimal solutions very quickly. Nevertheless, there is still another open question: *is the software cognitive complexity reduction of a method an NP-hard problem?*

## Acknowledgment

This resarch has been supported by Universidad de Málaga (grants B1-2020_01 and B4-2019-05) and project PID2020-116727RB-I00 funded by MCIN/AEI /10.13039/501100011033. Rubén Saborido and Javier Ferrer are supported by postdoc grants POSTDOC_21_00567 and DOC/00488, respectively, funded by the Andalusian Ministry of Economic Transformation, Industry, Knowledge and Universities.

## References

[1] G. A. Campbell, "Cognitive Complexity: An Overview and Evaluation," in *Proceedings of the 2018 International Conference on Technical Debt*, ser. TechDebt '18. New York, NY, USA: Association for Computing Machinery, 2018, pp. 57–58. [Online]. Available: https://doi.org/10.1145/3194164.3194186

[2] R. Saborido, J. Ferrer, F. Chicano, and E. Alba, "Automatizing software cognitive complexity reduction," *IEEE Access*, vol. 10, pp. 11 642–11 656, 2022.

# Tool showcases

# Online Testing of RESTful APIs with RESTest

Alberto Martin-Lopez
*SCORE Lab, I3US Institute*
*Universidad de Sevilla*
Seville, Spain
alberto.martin@us.es

***Abstract*—Online testing of web APIs—testing APIs in production—is gaining traction in industry. Platforms such as RapidAPI and Sauce Labs provide online testing and monitoring services of web APIs 24/7, however, they require designing test cases manually, which are continuously executed at regular intervals. In this talk, we present the RESTest testing ecosystem as an alternative for automated and thorough online testing of RESTful APIs. First, we describe its architecture and functionality. Then, we explain how it can be used to test your own APIs, including a demonstration. Lastly, we delve into our latest results when deploying this testing ecosystem in practice. On the one hand, we uncovered over 200 bugs in industrial APIs over the course of 15 days of testing. On the other hand, we identified challenges posed by online testing at scale, which open exciting research opportunities in the areas of search-based software engineering and machine learning.**

## I. Introduction

Web APIs provide access to data and functionality over the Internet, via HTTP interactions. They are the cornerstone of software integration, especially *RESTful* web APIs [1], currently considered the de facto standard for Web integration. As RESTful APIs become more pervasive and widespread in industry, their validation becomes more critical than ever before. A single bug in an API may affect tens or hundreds of other services using it. In this scenario, test thoroughness and automation are of utmost importance. Recently, academia and industry have made great efforts to address this problem, which has led to an explosion in the number of approaches and tools for testing RESTful APIs. Research approaches are focused on the automated generation of test cases, especially from a black-box perspective (i.e., without requiring access to the source code of the API). On the other hand, industrial solutions are mostly concerned with automating test case execution and providing online testing services, where APIs are continuously tested while in production. Customers of online testing platforms such as RapidAPI [2] or Sauce Labs [3] may choose among different pricing plans determining features such as the test execution frequency and the integration with CI/CD platforms, among others.

In this talk, we present the RESTest testing ecosystem as a powerful alternative for automated and thorough online testing of RESTful APIs at scale. First, we describe its architecture and its main features. Then, we make a demonstration on how it can be used to test your own APIs. Lastly, we report on our latest results on deploying the ecosystem to test industrial APIs. In particular, we uncovered over 200 bugs over the course of 15 days, but we also identified challenges of online testing when performed on a large scale, some of which could be tackled with search- and machine learning-based approaches.

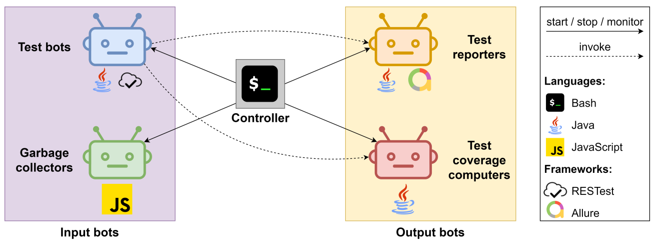


Fig. 1. Testing ecosystem architecture.

## II. RESTest Testing Ecosystem

The RESTest testing ecosystem is specifically designed for online testing of RESTful web APIs, that is, it allows to continuously test and monitor APIs in production for any given period of time (e.g., days or months). It follows a black-box strategy, where test cases are automatically derived from the API specification (e.g., an OAS document [4]). Test cases are continuously generated, executed and reported, and the test results can be monitored in a user-friendly dashboard.

### A. Architecture

Figure 1 depicts the architecture of the testing ecosystem. As illustrated, the architecture is decoupled into multiple types of *bots*, i.e., highly cohesive and autonomous programs that perform specific tasks within the testing process (e.g., test reporting). Bots can be independently developed and deployed using different technologies. We distinguish between *input bots* (support the generation and execution of test cases) and *output bots* (responsible for analyzing and leveraging test outputs). Bots are started, stopped and monitored automatically by a *controller* component, and they can optionally interact with each other, e.g., by triggering the update of test reports.

There are two types of input bots: *test bots*, which generate and execute test cases, and *garbage collectors*, which delete resources created by test bots (e.g., playlists in the Spotify web API). Regarding output bots, we conceive two types: *test reporters*, which generate graphical test reports, and *test coverage computers*, which compute the API coverage achieved by test bots [5].

### B. RESTest Framework

Test bots are based on RESTest [6], a black-box testing framework for RESTful web APIs. This has several benefits. On the one hand, existing test case and test data generation strategies from RESTest are already available in the online testing ecosystem, e.g., constraint-based testing [7] and realistic input test data extracted from knowledge bases [8]. On the other hand, implementing new testing strategies is straightforward, as it simply requires integrating them into RESTest. For details on how to develop new components in RESTest (including test case and test data generators), we refer the reader to its documentation [9] and its reference paper [6].

### C. How-to Guide

The configuration of our online testing ecosystem depends on the APIs under test and the bots used to test the selected APIs. Each API can be tested by multiple bots simultaneously. For each test bot, the following resources are required:

- *OAS specification of the API.* This file contains a machine-readable definition of the API that can be used to drive the generation of test cases. Note that it can be reused by multiple bots.
- *Test configuration file.* This file specifies the test data generation strategy used by the bot, for instance, fuzzing dictionaries [10], semantically-related data [8], or manually set data generators [7].
- *Properties file.* This file specifies several details related to the testing process such as the type of testing to perform (positive or negative), the test execution frequency, and the number of test cases to generate (or total test time).

Besides test bots, the remaining components of the ecosystem require little configuration: test reporters and test coverage computers are enabled/disabled based on the configuration of each test bot (as specified in its properties file); garbage collectors only need to be configured for those APIs where resources are created; the controller component is a ready-to-use set of Bash scripts. In order to ease the configuration and deployment of the whole ecosystem, we provide a supplementary package explaining the required steps to this end [11].

## III. Recent Results

We deployed our testing ecosystem for 15 days continuously testing 13 industrial APIs, including highly popular APIs with millions of users worldwide such as YouTube, Spotify and Yelp. Overall, we generated 1,101,846 test cases, we uncovered 389,216 test failures, and we conservatively narrowed down these failures to 254 unique bugs. These bugs were varied, including inconsistencies between API implementation and API documentation, internal server errors with valid and invalid input data, inconsistencies between the status codes and response bodies, unparseable JSON responses and unexpected client errors, to name a few. Based on our evaluation results, we also extracted insightful conclusions such as the most effective testing techniques (constraint-based testing and data perturbation) and the most recurrent types of bugs (disconformities with the API specification), among others.

## IV. Challenges Ahead

Despite our promising results, we identified several challenges hindering the adoption of automated test case generation methods for online testing of APIs at scale, including automated fault identification (i.e., classifying thousands or millions of failures into tens or hundreds of unique faults), effective human interaction (i.e., leveraging human input to make bots more effective), and optimal selection of testing strategies (i.e., automatically determining the most effective techniques based on several factors), among others. We envision the application of AI techniques to tackle these challenges. For instance, search algorithms could be used to automatically debug failures and isolate failure-inducing inputs. Similarly, active learning algorithms could leverage human input to improve the classification of faults by bots (e.g., confirming/discarding failures labeled as "warning").

## V. Conclusion

Online testing of web APIs is becoming an increasingly common practice in industry. However, existing platforms mostly automate test execution, while test cases still need to be manually implemented. In this talk, we overviewed the RESTest testing ecosystem and its capabilities to find real-world bugs in industrial APIs. We also highlighted challenges for the application of such a framework in practice, and how AI techniques could help address these challenges.

## Acknowledgments

This work has been partially supported by FEDER/Ministerio de Ciencia e Innovación - Agencia Estatal de Investigación under project HORATIO (RTI2018101204-B-C21), by FEDER/Junta de Andalucía under projects APOLO (US-1264651) and EKIPMENT-PLUS (P18-FR-2895), by the FPU scholarship program, granted by the Spanish Ministry of Education and Vocational Training (FPU17/04077).

## References

[1] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," Ph.D. dissertation, University of California, Irvine, 2000.
[2] "RapidAPI," https://rapidapi.com, accessed March 2022.
[3] "Sauce Labs," https://saucelabs.com, accessed March 2022.
[4] "OpenAPI Specification," https://spec.openapis.org/oas/latest.html, accessed April 2022.
[5] A. Martin-Lopez, S. Segura, and A. Ruiz-Cortés, "Test Coverage Criteria for RESTful Web APIs," in *10th International Workshop on Automating TEST Case Design, Selection, and Evaluation*, 2019, pp. 15–21.
[6] A. Martin-Lopez, S. Segura, and A. Ruiz-Cortés, "RESTest: Automated Black-Box Testing of RESTful Web APIs," in *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2021, pp. 682–685.
[7] A. Martin-Lopez, S. Segura, and A. Ruiz-Cortés, "RESTest: Black-Box Constraint-Based Testing of RESTful Web APIs," in *International Conference on Service-Oriented Computing*, 2020, pp. 459–475.
[8] J. C. Alonso, A. Martin-Lopez, S. Segura, J. M. Garcia, and A. Ruiz-Cortes, "ARTE: Automated Generation of Realistic Test Inputs for Web APIs," *IEEE Transactions on Software Engineering*, 2022.
[9] "RESTest," https://github.com/isa-group/RESTest, accessed April 2022.
[10] V. Atlidakis, P. Godefroid, and M. Polishchuk, "RESTler: Stateful REST API Fuzzing," in *2019 IEEE/ACM 41st International Conference on Software Engineering*, 2019, pp. 748–758.
[11] "[Supplementary material] Online Testing of RESTful APIs: Promises and Challenges," https://doi.org/10.5281/zenodo.6365937, 2022.

# Type4Py: Machine Learning-based Type Auto-completion for Python

Amir M. Mir
Department of Software Technology
Delft University of Technology
Delft, The Netherlands
Email: s.a.m.mir@tudelft.nl

Sebastian Proksch
Department of Software Technology
Delft University of Technology
Delft, The Netherlands
Email: s.proksch@tudelft.nl

Georgios Gousios
Department of Software Technology
Delft University of Technology
Delft, The Netherlands
Email: g.gousios@tudelft.nl

***Abstract*—In this short report, we present Type4Py, a tool that assists Python developers to retrofit type annotations to their codebases. It is powered by a deep learning model that is trained on 5.2K open-source Python projects. Type4Py is available as a Visual Studio Code extension.**

## I. Introduction

The Python programming language is extremely popular nowadays among software developers as it is easy to use and allows rapid prototyping. The IEEE Spectrum ranks Python as the most popular programming language in 2021[1]. Despite its popularity, Python lacks static types, which causes type errors and unexpected run-time exceptions. To mitigate these issues, Python 3.5 added support for optional type annotations. This means that Python developers can gradually add type annotations to their existing codebases. However, this is a daunting and error-prone task.

Researchers have recently proposed machine learning (ML)-based type prediction models for dynamic programming languages [1], [2]. These ML-based type prediction models perform sophisticated feature extraction and computationally expensive analysis such as control/data flow analysis or search-based validation, which makes them impractical to be used in IDEs by developers. Motivated by this, very recently, we proposed Type4Py [3], a deep similarity learning (DSL)-based type inference model for Python. Specifically, the Type4Py model is based on hierarchical neural networks, which learns to discriminate between similar and dissimilar types in a high-dimensional space, namely, type clusters. Given this, K-nearest neighbor search is performed to suggest type annotations for a test query.

Type4Py has mainly two advantages compared to the recent state-of-the-art approaches, TypeWriter [1] and Typilus [2]:

1) Its mean reciprocal rank (MRR) score is 77.1%, which is 8.1% and 16.7% higher than Typilus and TypeWriter, respectively. For example, considering a list of 10 predictions, a higher MRR score means that the model predicts a correct type annotation among the first few predictions in the list.
2) It is practical and can be used as a developer tool, i.e., it can be used in an IDE, Visual Studio Code, to assist Python developers to gradually adding type annotations to their existing codebases.

In this short report, we present Type4Py as a developer tool. It was initially released in July 2021 and its Visual Studio extension[2] has over 1,100 installations at the time of this writing.

## II. Type4Py

### A. Core Features

Type4Py has the following main features:

- It is powered by a DSL-based model that is trained on the ManyTypes4Py dataset [4] with 5.2K open-source Python projects.
- It provides ML-based type auto-completion functionality in Visual Studio Code. Figure 1 shows one usage example from VS Code.
- It can predict type annotations for variables, functions' arguments, and return values.
- It has a local model that runs end-to-end locally on users' machines. This solves common privacy concerns by developers such as sharing or sending their source codes to external servers.

### B. Design

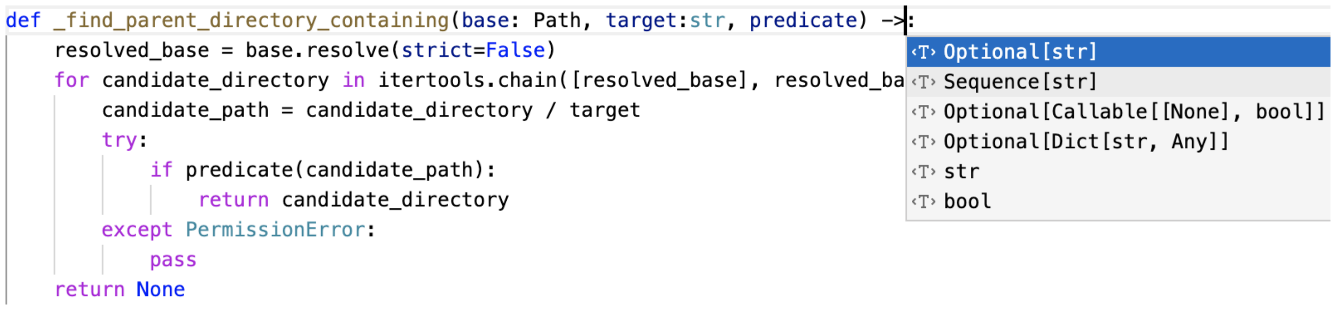
The design of Type4Py is shown in Figure 2. At the client-side, to provide type auto-completion, the VS Code extension sends Python source code file(s) to either the local model or the production server. At the server-side, a tiny REST API with a prediction endpoint queries the pre-trained Type4Py model. For the model to predict types, Python files are processed to extract features/type hints (see [3] for details). Finally, the predicted type information is returned as a JSON response to the extension.

### C. Implementation

The VS code extension is written in TypeScript[3] and the Type4Py model is implemented in Python 3 using the PyTorch framework[4]. Specifically, The REST API is implemented

[1]https://spectrum.ieee.org/top-programming-languages/

[2]https://marketplace.visualstudio.com/items?itemName=saltud.type4py
[3]https://github.com/saltudelft/type4py-vscode-ext
[4]https://github.com/saltudelft/type4py



```
def _find_parent_directory_containing(base: Path, target:str, predicate) ->:
    resolved_base = base.resolve(strict=False)
    for candidate_directory in itertools.chain([resolved_base], resolved_ba
        candidate_path = candidate_directory / target
        try:
            if predicate(candidate_path):
                return candidate_directory
        except PermissionError:
            pass
    return None
```

Fig. 1. A type auto-completion example from VS Code. The expected return type is `Optional[str]`.

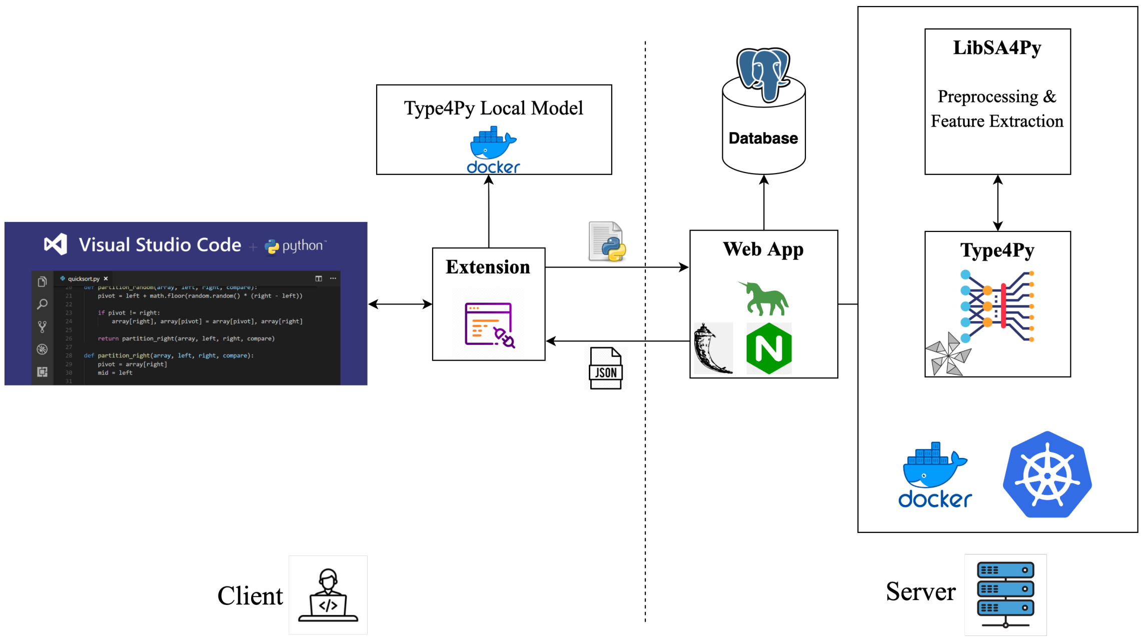


Fig. 2. Design and deployment of Type4Py in practice

using the light-weight Flask web framework. The pre-trained Type4Py model uses the ONNX runtime[5] for faster inference on both CPUs and GPUs. For KNN search, Annoy[6] is employed, which is fast and memory-efficient. To analyze Python source files for feature extraction, we use our LibSA4Py library[7].

### D. Deployment

To deploy Type4Py, a Docker image is created which contains the pre-trained model, pre-processing/feature extraction component, and the REST API. To scale the deployment, a number of containerized stateless Type4Py applications are currently deployed on our Kubernetes cluster.

## Acknowledgment

This research work was funded by H2020 grant 825328 (FASTEN).

[5]https://onnxruntime.ai/index.html
[6]https://github.com/spotify/annoy
[7]https://github.com/saltudelft/libsa4py

## References

[1] M. Pradel, G. Gousios, J. Liu, and S. Chandra, “Typewriter: Neural type prediction with search-based validation,” in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 209–220.

[2] M. Allamanis, E. T. Barr, S. Ducousso, and Z. Gao, “Typilus: neural type hints,” in *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2020, pp. 91–105.

[3] A. M. Mir, E. Latoskinas, S. Proksch, and G. Gousios, “Type4py: Practical deep similarity learning-based type inference for python,” in *2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE)*. IEEE, 2022.

[4] A. M. Mir, E. Latoškinas, and G. Gousios, “Manytypes4py: A benchmark python dataset for machine learning-based type inference,” in *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*. IEEE, 2021, pp. 585–589.

# SAd-CloudSim: A Toolkit for Modeling and Simulation of Self-Adaptive Cloud Software Architectures

Maria Salama
School of Computer Science
University of Birmingham
Birmingham, UK
m.salama@bham.ac.uk

***Abstract*—Cloud-based software systems are increasingly becoming complex and operating in highly dynamic environments. Self-adaptivity and self-awareness have recently emerged to cope with such level of dynamicity and scalability. Self-adaptivity has been motivated as a solution to achieve the level of dynamicity and scalability necessary for these systems, as well as to comply with the changes in components, fluctuations in workloads and environmental conditions during runtime. This talk presents a modeling and simulation tool for self-adaptive cloud architectures. The tool provides significant benefits for designing self-adaptive cloud architectures, as well as testing adaptation mechanisms. The tool is also beneficial as a symbiotic simulator during runtime to support runtime adaptation decisions.**

## I. Introduction

Self-adaptive software architectures are expected to manage themselves following the principles of autonomic computing, to respond to changes in end-user requirements and the environment and to cope with uncertainty in runtime operation for continued satisfaction of quality requirements under changing context conditions. In self-adaptive systems, adaptation decisions are taken during runtime with the aid of feedback loops (individual, collective or decentralised), analytical models, or by learning from historical data [1]. In this case, symbiotic simulations are powerful tools to support adaptation decisions during runtime. Such tools can be used symbiotically with the adaptation controller of the system, due to their ability to dynamically incorporate real-time data sensed from the system in running what-if scenarios and feedback to the adaptation controller with the effects of adaptation decisions.

On another side, simulation tools are needed to fill the gap between the conceptual research and the proof-of-concept implementation [2]. Such tools help to systematically model and study the behavior and performance of these systems that tend to operate in dynamically changing environments hard to define during system design [2]. In the context of cloud computing, simulators were known as tools to support and accelerate research and development of cloud computing systems, applications and services, as quantifying the performance of service provision in real cloud environment is challenging [3] [4].

Given the highly dynamic operating environment of cloud computing and its on-demand nature, cloud architectures tend to heavily leverage on adaptation to dynamically fulfill the uncertain and changing runtime demand [3]. The case of self-adaptive cloud architectures combines challenges of both clouds and self-adaptive architectures. In such cases, testing architecture design or resources provisioning mechanisms, quantifying the architecture performance, and measuring the quality of service provisioned in real environments are challenging tasks.

In this talk, we present *SAd-CloudSim*, a tool for modeling and simulation self-adaptive cloud architectures. The toolkit is build on the widely adopted cloud simulation environment *CloudSim* [3] [4]. The new extensions turn CloudSim to work with real systems at runtime as a symbiotic simulator, where self-adaptation helps in taking well-informed adaptation decisions.

The *SAd-CloudSim* toolkit offers the following novel extensions: (i) modeling and simulation of adaptation mechanisms for large-sale cloud-based systems, (ii) a self-contained platform for modeling and testing self-adaptation mechanisms, (iii) support for testing the performance of cloud systems under varying dynamic workloads and with different quality goals, and (iv) support extensions for modeling and testing self-adaptation frameworks and techniques.

The tool is published as open-source software, available at https://github.com/m-salama/SAdSAwCloudSim.

## II. SAd-CloudSim Architecture and Design

The *SAd-CloudSim* tool is built on top of the CloudSim core simulation engine and CloudSim core. The Self-Adaptation layer is added on top of the cloud core architecture, to model the adaptation controller of a self-adaptive software system. Researchers and practitioners, willing to design an adaptation technique or study the efficiency of an existing one, would need to implement their techniques in this layer. The top-most layer is the Simulation Application, inherited from CloudSim, which models the specification of the simulation to be conducted using the tool. Such specifications allow configuring the simulation of dynamic workloads, different service types and user requirements.

A foundational self-adaptation controller consists of: (i) monitor for correlating quality data, (ii) detector for analyzing the data provided by the monitor and detecting violations to trigger adaptation when necessary, (iii) adaptation engine to determine what needs to be changed and select the optimal adaptation strategy, and (iv) adaptation executor responsible for applying the adaptation action on the underlying infrastructure. Our initial implementation of *SAd-CloudSim* includes this foundational version of adaptation controller. Such components could be further extended to study more complex adaptation mechanisms, such as pro-active adaptations or MAPE-K adaptation process [1].

The *Monitor* component is responsible for monitoring the achievement of quality requirements. The *Detector* checks any violations occurring during runtime against quality goals. Whenever a violation is detected, adaptation is triggered. The *Adaptation Engine* is responsible for analyzing the current situation and selecting the optimal adaptation strategy that would achieve the quality targets, e.g. increasing VMs capacity. The selected adaptation tactic is executed dynamically during runtime on the cloud infrastructure by the *Adaptation Executor*.

Quality goals are the main objective or trigger for self-adaptation. *QoS Goals* represent the quality of service targets required to be fulfilled. Whenever violated, an adaptation should take place to achieve the quality goals. For each QoS Goal, a set of possible adaptation tactics is implemented in the tactics catalog. Also, adaptation rules are defined as *if-condition-then-action* rules, where the conditions are quality requirements and the actions are response tactics.

## III. Simulation of Self-adaptive Architectures using SAd-CloudSim

We have extended some core classes of CloudSim by adding necessary quality and power (energy) metrics. A *RuntimeWorkload* is added to allow conducting experiments for consecutive time intervals, and user requirements are added to configure QoS requirements.

The *Self-Adaptation* package encapsulates the components necessary for modeling and simulating a self-adaptive architecture. Our initial implementation includes the basic functionalities of these components. Figure 1 depicts the flow of the simulation process of self-adaptation. These components could be further extended with more sophisticated implementations, such as MAPE-K. This package is composed of the following classes:

- *Goals Model* class is the list of goals objects loaded from a configuration file.
- *Monitor* class contains methods sensing, measuring and collecting actual data of the QoS parameters of the executed requests.
- *Detector* class contains a method triggered to run after receiving data from the monitor. It checks the runtime values of the quality metrics against the Goals Model. If a violation is detected, adaptation is triggered.
- *Adaptation Engine* class is responsible for selecting the optimal adaptation action after receiving the adaptation trigger. The adaptation action is selected from the Adaptation Tactics Catalog according to the adaptation rules.
- *Adaptation Tactics Catalog* class contains a list of adaptation tactics.
- *Adaptation Rule* class links quality attributes with their adaptation tactics.
- *Adaptation Executor* class performs the actual execution of the selected adaptation action.

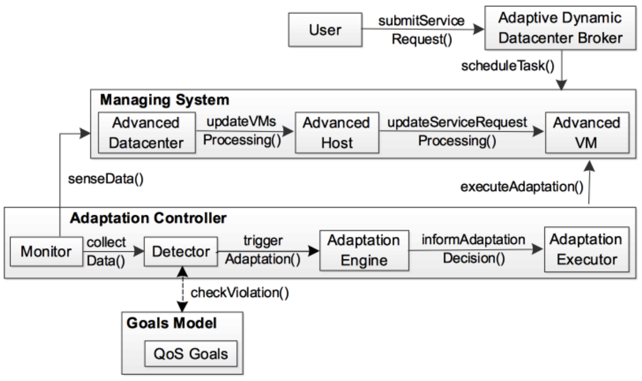


Fig. 1. Self-Adaptation simulation process

## Acknowledgment

Maria's PhD studentship was funded by the School of Computer Science, The University of Birmingham, UK. This research was supported in part by the Universitas 21 and the CLOUDS Lab, University of Melbourne, Australia. The author would like to especially thank Rodrigo N. Calheiros and Maria A. Rodriguez for the support in developing and testing the toolkit.

## References

[1] M. Salehie and L. Tahvildari, "Self-adaptive software: Landscape and research challenges," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 4, no. 2, pp. 1–42, 2009.

[2] T. D. Nya, S. C. Stilkerich, and P. R. Lewis, "A modelling and simulation environment for self-aware and self-expressive systems," in *IEEE 7th International Conference on Self-Adaptation and Self-Organizing Systems Workshops (SASOW)*, 2013, pp. 65–70.

[3] R. Buyya, R. Ranjan, and R. N. Calheiros, "Modeling and simulation of scalable cloud computing environments and the cloudsim toolkit: Challenges and opportunities," in *International Conference on High Performance Computing & Simulation (HPCS)*. IEEE, 2009, pp. 1–11.

[4] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. De Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, 2011.

# Author Index