GOTTEN: A Model-driven Solution to Engineer Domain-specific Metamorphic Testing Environments

Pablo Gómez-Abajo
Universidad Autónoma de Madrid
Madrid, Spain
Pablo.GomezA@uam.es

Pablo C. Cañizares
Universidad Autónoma de Madrid
Madrid, Spain
Pablo.Cerro@uam.es

Alberto Núñez
Universidad Complutense de Madrid
Madrid, Spain
Alberto.Nunez@ucm.es

Esther Guerra
Universidad Autónoma de Madrid
Madrid, Spain
Esther.Guerra@uam.es

Juan de Lara
Universidad Autónoma de Madrid
Madrid, Spain
Juan.deLara@uam.es

***Abstract*—Testing is essential for assessing the correctness of software systems. Metamorphic testing (MT) is an approach especially suited when the system under test lacks oracles, or they are expensive to compute. However, creating an MT environment for a specific domain (e.g., cloud simulation, model transformation, machine learning) requires substantial effort.**

To alleviate these difficulties, we present a model-driven tool that automates the construction of MT environments. Starting from a meta-model with the domain concepts, and a description of the domain execution environment, our tool produces an MT environment featuring comprehensive support for the MT process. This includes the definition of domain-specific metamorphic relations, their evaluation, detailed reporting of the testing results, and the automated search-based generation of follow-up test cases. This paper illustrates the tool on a case-study in the domain of video streaming APIs. A video showcasing the tool is available at https://youtu.be/DeuIW6V4LaQ.

***Index Terms*—Metamorphic testing, Model-driven engineering, Domain-specific languages, Video streaming APIs**

I. INTRODUCTION

Metamorphic testing (MT) is a testing technique suitable in scenarios where the system under test (SuT) has no oracle or is difficult to compute [1]. MT is based on establishing so-called metamorphic relations (MRs), which state that when two or more input test cases are related in a certain way, their outputs should be related as specified. For example, when testing the trigonometric sine function, one can define MRs that profit from identities like $-sin(x) = sin(-x)$. Thus, given two input test cases x_1 and x_2, s.t. $x_1 = -x_2$, the outputs of applying the *sin* function to them must satisfy the relation $-sin(x_1) = sin(x_2)$.

Due to its generality, MT has been applied in a wide range of domains [1], including compilers [2], machine learning [3], or autonomous driving [4]. MT has also been employed in model-driven engineering (MDE), e.g., to test model transformations [5] or the semantics of domain-specific languages (DSLs) [6]. Part of the success of MT is because MRs can embed domain knowledge, as well as be used both as oracles and to automatically generate *follow-up test cases*. The latter are input test cases related as required by a given MR (e.g., $x_1 = -x_2$ in our example).

Typically, MT engines are created manually, which incurs a high effort and often leads to ad-hoc solutions that are hardly extensible. This is so as a full-fledged MT solution needs to provide support to encode the MRs, generate follow-up test cases, execute the SuT with the test cases that satisfy the input part of the MRs, and check whether the execution results fulfil the output part of the MRs. However, MRs are frequently hard-coded within the MT solution, which is not extensible and does not offer full support for the MT cycle [7].

To alleviate this problem, we propose GOTTEN, a model-driven solution to engineer MT environments for specific domains [7]. The approach is based on a DSL to specify MRs, and on representing the input test cases as models conformant to a domain meta-model. GOTTEN provides full support for the MT cycle, including the automatic generation of follow-up test cases using model search, and detailed reporting of the MT results. Moreover, it offers facilities to compare the behaviour of several SuTs against the same MRs and test cases, which is useful, e.g., to compare several simulators, machine learning alternatives, or different versions of the same SuT for regression testing.

Paper organisation. Section II overviews our approach and introduces a running example in the domain of video streaming APIs. Section III explains the architecture of our solution, and the tooling, which is based on Eclipse. Section IV compares with related works, and Section V concludes.

II. OVERVIEW AND RUNNING EXAMPLE

As a running example, assume we would like to test a video streaming API, like that of Youtube[1] or Vimeo[2]. MT is useful here, since the expected system behaviour can be modelled as MRs, like this one: "*Perform a search for videos published before a year y. Then perform another search for videos after*

[1]https://developers.google.com/youtube/
[2]https://developer.vimeo.com/api/

y with same query as before. The result set should be disjoint" (taken from [8]).

Testing should be automated, and so, we would need to create an MT environment to define and execute MRs such as the exemplified one. However, manually building an MT tool for video streaming is challenging and requires considerable effort. Instead, in the remainder of this paper, we showcase how to facilitate its construction with the help of GOTTEN. Fig. 1 overviews the construction process, which involves the roles of *application expert*, *domain expert* and *tester*.

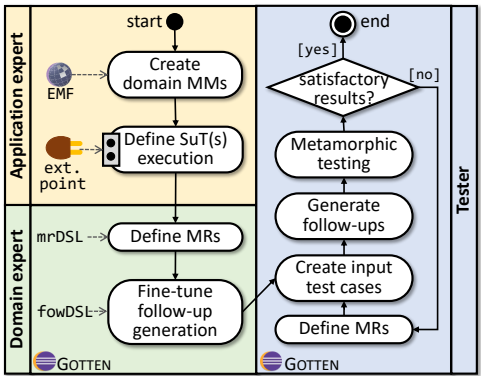

Fig. 1. Building and using an MT environment with GOTTEN.

First, the *application expert* defines a domain meta-model capturing the structure of the inputs expected by the SuT. Fig. 2 shows the meta-model for video streaming APIs that we have designed. It captures the possible actions of the API (e.g., SearchVideo, UploadVideo, UpdateVideo) and the input data. For example, when searching for a video, it is possible to input parameters like the maximum number of results (attribute maxResults), the search query (attribute query), the date range for the search (class VideoDate), the geo-location (class VideoPosition), and the ordering criterion (enum Order).

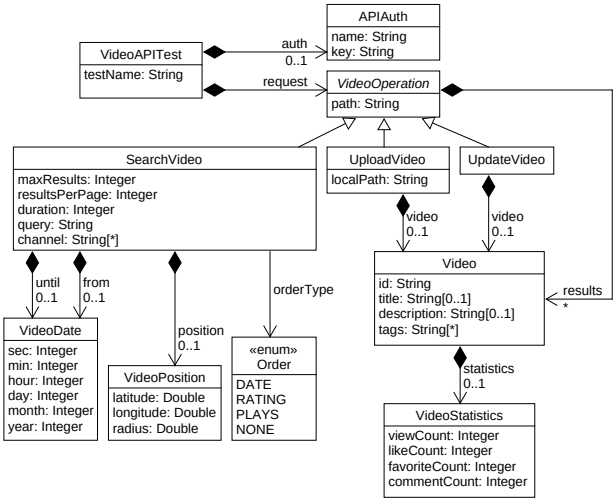

Fig. 2. Meta-model for video streaming APIs.

The application expert must also specify how to execute the SuT programmatically, using an extension point of GOTTEN. This extension must pass the inputs (instances of the defined meta-model) to the SuT, and execute the SuT. GOTTEN can handle several SuTs. In our example, this is useful to compare different APIs (e.g., Youtube against Vimeo), or to perform regression tests to ensure that the behaviour of a new API implementation conforms to the previous one (e.g., Youtube's API has 14 revisions since 2021[3]). From the provided information (domain meta-model and SuT execution process), GOTTEN generates a dedicated MT environment.

In this MT environment, the *domain expert* can use the *mrDSL* language to define MRs that capture knowledge of the domain. Starting from the defined MRs, GOTTEN is able to automatically generate follow-up test cases. For this purpose, it relies on the MOMoT search-based framework [9]. The domain expert can fine-tune this generation process by means of the *fowDSL* language, which allows configuring the mutation operations to be used during the test case search.

Finally, the *tester* can use the MT environment to perform MT. Specifically, the tester creates some initial test cases manually, which are used to automatically generate follow-up test cases. Then, the MT environment evaluates the MRs on the initial test cases and the follow-ups, producing a testing report. If the results are not satisfactory, new MRs, input test cases and follow-ups can be defined, and the process is repeated.

III. GOTTEN

Next, we describe GOTTEN's architecture (Section III-A) and the tooling (Section III-B).

A. Architecture of GOTTEN

Fig. 3 shows the architecture of GOTTEN. It is an Eclipse plugin with an extension point (*Processor*) that the application experts can instantiate to gain programmatic access to the SuT. For illustration, the figure shows two extensions for Youtube and Vimeo (label 1).

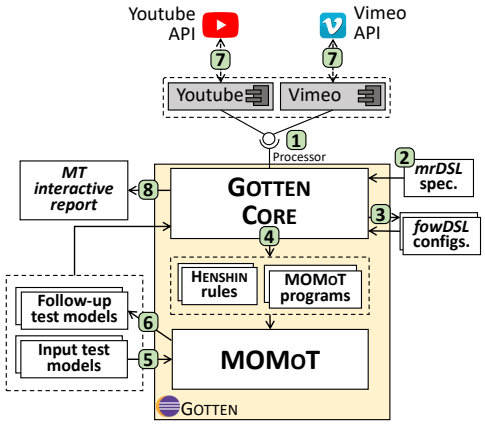

Fig. 3. Architecture of GOTTEN.

GOTTEN includes Xtext-based editors for the languages *mrDSL* and *fowDSL*. The former permits defining MRs (label 2), which will be used for the MT process, and also to

[3]https://developers.google.com/youtube/v3/revision_history

generate follow-up test cases. The test case generation relies on MOMoT [9], a model-based search engine. Specifically, starting from the MRs, GOTTEN generates a file defining the mutation operators to be used for the search by MOMoT. These operators are specified using the DSL *fowDSL*, and can be fine-tuned manually by the domain expert. Then, GOTTEN automatically transforms the *fowDSL* configurations into the input format of MOMoT, and into Henshin rules (labels 3 and 4). Finally, MOMoT uses the defined input test models as seeds (label 5), and synthesizes follow-up test cases according to the given search configuration (label 6).

At this point, the MT process can start. GOTTEN passes the test cases to the implemented processors (label 7), and the output of their execution is checked against the MRs. The result is displayed in interactive reports (label 8).

B. Tool support

The GOTTEN framework is available at http://g0tten.github.io/gotten/ along with an installation guide and usage examples. The framework supports two DSLs – *mrDSL* and *fowDSL* – explained next.

1) mrDSL: This DSL enables the definition of MRs. Listing 1 describes three MRs for the running example. Line 1 specifies the path of the videostream meta-model, and line 2 the location of the conforming input models.

```
1  metamodel videostream "/video/model/VideoStream.ecore" with m1, m2
2  models "/video/model/videotc"
3
4  videostream input Features {
5    context VideoAPITest def: IsFullSearch: Boolean =
6            request.oclIsTypeOf(SearchVideo)
7            and request.oclAsType(SearchVideo).maxResults = -1
8    context SearchVideo def: SearchOrder: Int = orderType
9    context SearchVideo def: UntilYear: Int = until.year
10   context SearchVideo def: FromYear: Int = from.year
11   context SearchVideo def: Radius: Double = position.radius
12 }
13
14 output Features {
15   NVideos: Long
16   Results: Set
17 }
18
19 Processor {
20   Name: String
21   Version: String
22 }
23
24 MetamorphicRelations {
25   MR1 = [ (IsFullSearch(m1) and SearchOrder(m1) <> SearchOrder(m2))
26           implies (NVideos(m1) == NVideos(m2)) ]
27   MR2 = [ (IsFullSearch(m1) and UntilYear(m1) < FromYear(m2))
28           implies (Results(m1)→excludes(Results(m2))) ]
29   MR3 = [ (IsFullSearch(m1) and Radius(m1) > Radius(m2))
30           implies (Results(m1)→includes(Results(m2))) ]
31 }
```

Listing 1. *mrDSL* specification of three MRs for the running example.

Lines 4–12 define input features (IsFullSearch, SearchOrder, UntilYear, FromYear, Radius), which are OCL expressions that extract information from the input models and may appear in the left part of the MRs. Lines 14–17 declare output features (NVideos, Results). These represent features obtained from the tests executed in the SuT, and can appear at the right part of the MRs.

Lines 19–22 include meta-data of the processor (Name and Version) to identify the implemented Processors, Youtube and Vimeo in our case study.

Finally, three MRs are coded in lines 24–31 by using the previously defined input and output features. MR1 states that if two searches are performed by only changing the sorting criteria (SearchOrder(m1) <> SearchOrder(m2)), then both should return the same number of videos. MR2 encodes the relation introduced in Section II ("*Perform a search for videos published before a year y. Then perform another search for videos after y with same query as before. The result set should be disjoint*"). MR3 expresses that, given two video searches, where the search radius of the first is greater than that of the second, then the results obtained from the first query must include the results obtained in the second. Please note that *mrDSL* assumes that the model elements not mentioned by the MR remain equal in both test models.

2) fowDSL: This DSL allows fine-tuning the generation of follow-up test cases. Listing 2 shows an example *fowDSL* configuration for generating follow-up test cases for MR1. Line 1 selects the meta-model of the input models and the MR used to guide the search of new follow-up test cases. Lines 2–3 specify the folders where the input models and follow-up test cases are located, respectively. Line 5 shows the rule that GOTTEN generates automatically for seeding variations in the SearchOrder feature. This includes a constraint preventing the value *NONE* for the feature.

Line 7 defines the fitness function used to give values to the search order. In this case, OrderType.NONE has value 0, while the rest of the enumerates have value 1.

Finally, lines 9–11 set the maximum number of solutions (i.e., follow-up test cases) to generate, the number of iterations of the search algorithm, and the evolutionary algorithms to be used in the search process, respectively.

```
1  followups for videostream using MR1
2  with source path = "/video/model/videotc"
3  and output folder = "/video/model/videotc"
4
5  SearchOrder →require SearchVideo.orderType <> NONE
6
7  maximize (SearchOrder(m2) – SearchOrder(m1))
8
9  maxSolutions 3
10 iterations 1
11 algorithms [Random, NSGAII, NSGAIII, eMOEA]
```

Listing 2. *fowDSL* configuration for MR1 in Listing 1.

3) The MT process: Fig. 4 presents a screenshot of the GOTTEN environment. It provides editors for *mrDSL* and *fowDSL* featuring code completion, syntax highlighting and validation (labels 1 and 2). The tool supports the creation of GOTTEN projects (label 3), and provides a wizard to synthesize *fowDSL* configurations from the given *mrDSL* specifications, and another to add Processor extensions to the workspace (two are shown in label 4). The environment allows launching the MT process for the selected Processors (see Fig. 5) and shows the results of the execution for each MR in two additional views (simplified in label 5, detailed in label 6 of Fig. 4).

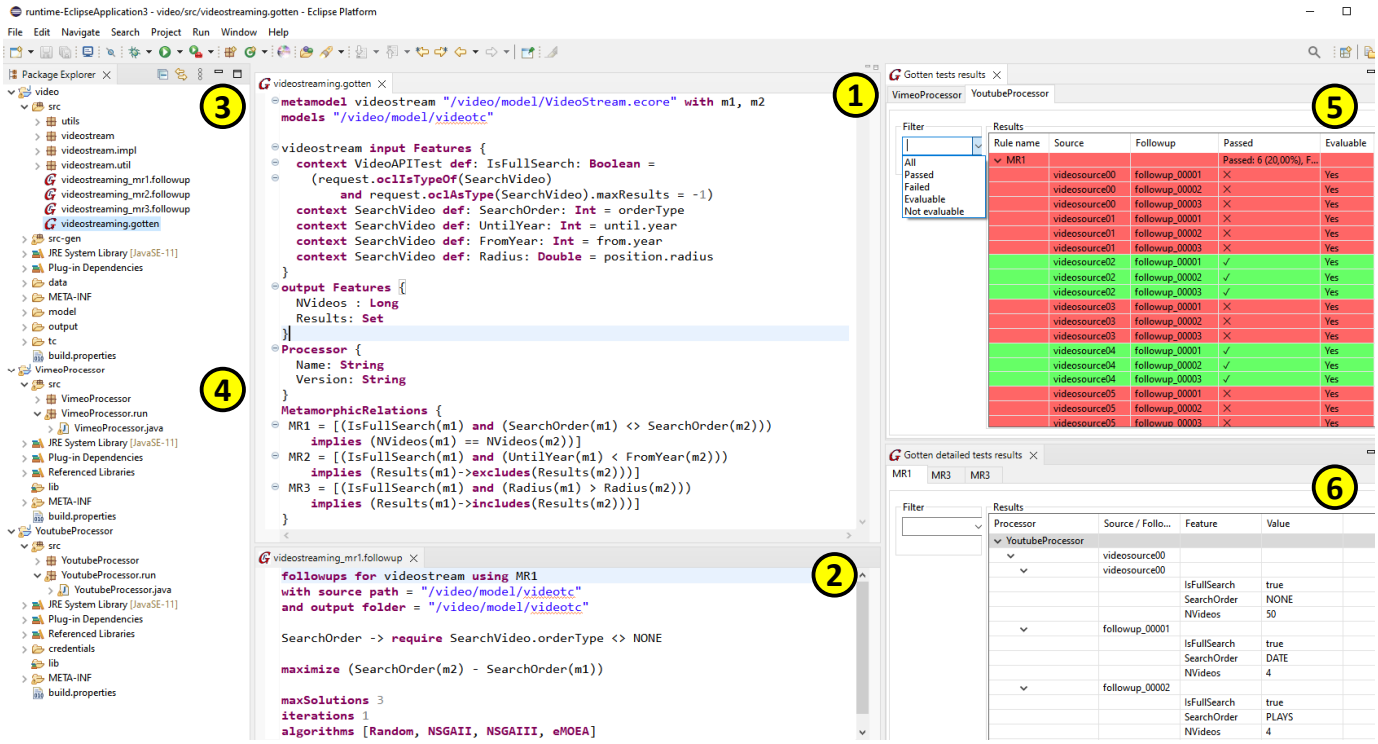

Fig. 4. GOTTEN development environment: (1) Defining MRs in the *mrDSL* editor; (2) Fine-tuning the follow-up test case generation with *fowDSL*; (3) GOTTEN project containing some *mrDSL* and *fowDSL* programs; (4) Project implementing the Processor Java interface for video streaming APIs; (5, 6) Different views with the results of the MT process.

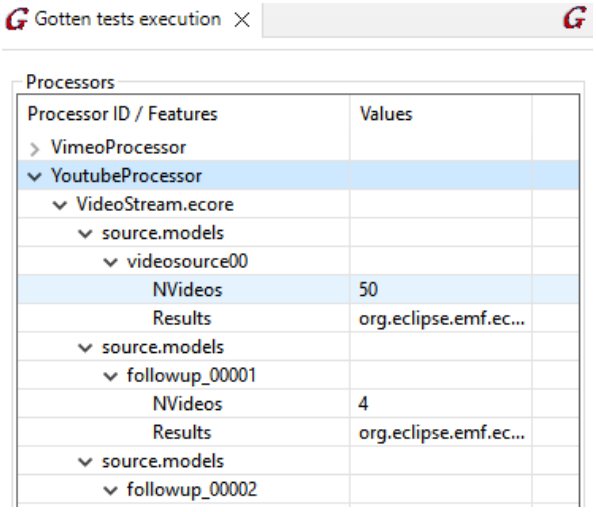

Fig. 5. GOTTEN execution view.

To show the suitability of GOTTEN, we have integrated two processors of video streaming services by implementing the Youtube and Vimeo APIs. After executing the tool using the three MRs (see Listing 1) and the two processors, we reach the following conclusions: i) MR1 detects an issue in both processors, because the number of videos returned by a query that specifies a search order differs from the number of videos returned by the same query without this parameter. This issue happens in two queries with the Youtube platform ('winter pentathlon 1949' and 'mistrustfully') and two more with the Vimeo platform ('Warzone' and 'music trend'). Regarding this MR1, the issues found by GOTTEN with the Youtube platform were also identified by Segura et al. [10], while to the best of our knowledge, the issues detected with the Vimeo platform are new. ii) MR2 is satisfied by the follow-up test cases for the Youtube processor. However, this MR detects a bug in Vimeo. In particular, the *filter date* field is not taken into account when the query is executed; iii) MR3 is also satisfied by the tests executed on the Youtube processor, but it is not supported by Vimeo.

Overall, we were able to create an MT environment for video streaming with GOTTEN; we reused all meta-models, input models and follow-up test cases for both processors; and we discovered issues in both APIs.

IV. RELATED WORK

In this section, we review related works on MT frameworks, and on the use of MT in MDE.

MT frameworks. In recent years, the number of works on MT has increased considerably. Moreover, MT has proven to be valuable in many research areas such as MDE, compilers, artificial intelligence, and autonomous driving, among others.

Li et al. [3] use MT as a robustness testing method to check the quality of DeepL, a neural machine translation service for which preliminary results reveal a lack of robustness. MT-DLComp [2] is an MT framework to reveal

erroneous compilations in deep learning (DL) compilers. The framework identified more than 435 inputs that can lead to faulty compilations, in four widely-used DL compilers by Amazon, Facebook, Microsoft, and Google. RMT [4] is a rule-based MT framework that converts human-written rules into MRs by employing an NLP-based parser that references an ontology list. The test cases are generated using various image transformation engines to ensure diversity. The authenticity of the test cases, and the validity of the anomalies detected, were confirmed through a human study conducted on Amazon Mechanical Turk. MIA (Metamorphic Interaction Automaton) [11] is the first reported large-scale implementation of MT in an industrial setting. The authors address challenges related to test flakiness and the oracle problem while testing Facebook's Web Enabled Simulation. This simulation system is built upon a Web infrastructure of hundreds of millions of lines of code. MT-EA4Cloud [12] combines MT and evolutionary algorithms to test cloud computing simulators using eight MRs about energy consumption.

Overall, these MT frameworks have been developed manually, which requires substantial effort. A tool like GOTTEN could have saved implementation effort.

MT in MDE. Several works combine MT and MDE. Most of them apply MT to model transformations, usually written in ATL [13]. For example, Du et al. [14] integrate MT with spectrum-based fault localisation to locate faulty rules in ATL transformations. To do so, the authors exploit code coverage information collected during the execution of individual test cases, and the test results. Somewhat similarly, Troya et al. [5] also analyse the execution traces of ATL transformations, in this case, to infer likely MRs for the transformations. The inferred MRs aim at detecting faults in three scenarios: regression testing, incremental transformation, and language migration. He et al. [15] evaluate the correctness of bidirectional transformations (BXs) via MT. They identify three generic MRs for BXs (GETPUT, PUTGET, and PUTTWICE), which can be complemented with domain-specific MRs to account for transformation-specific semantics. The approach is supported by an MT framework that allows defining test models via a tree editor, and execute them on ATL-based BXs.

Just a few MT proposals do not deal with model transformations. Boussaa et al. [16] use non-functional MRs modelling resource usage and performance to identify inconsistencies in code generators. More differently, Chaleshtari et al. [17] present a generative MT approach for security testing. It relies on an Xtext-based DSL to define MRs that capture security properties of Web systems, and from which Java code is generated to perform security testing. One of our first usages of GOTTEN [6] was to apply MT to test the semantics of DSLs, i.e., simulators that take as input a model in the DSL.

Overall, all these approaches were developed for a specific domain or language. However, to the best of our knowledge, none of the current MT frameworks use MDE to create MT environments. Our tool aims to fill this gap, as it can save effort and time when creating MT environments.

V. CONCLUSIONS AND FUTURE WORK

This paper has showcased GOTTEN, a tool to automate the engineering of MT environments for specific domains. The tool supports the full cycle of MT, including the specification of MRs, the generation of test cases, and detailed reporting. MT environments built with GOTTEN are extensible, since new MRs can be added externally, and the tool facilitates the comparison of several versions of the SuT.

We will improve the follow-ups generation process to support more powerful synthesis of strings (e.g., using Wordnet[4]). We are currently extending the *mrDSL* language with more primitives. We also plan to use GOTTEN for MT in typical MDE tasks, like model transformation and code generation.

ACKNOWLEDGMENTS

Work supported by the Spanish MICINN (PID2021-122270OB-I00, TED2021-129381B-C21).

REFERENCES

[1] S. Segura, D. Towey, Z. Q. Zhou, and T. Y. Chen, "Metamorphic testing: Testing the untestable," *IEEE Softw.*, vol. 37, no. 3, pp. 46–53, 2020.

[2] D. Xiao, Z. Liu, Y. Yuan, Q. Pang, and S. Wang, "Metamorphic testing of deep learning compilers," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 6, no. 1, 2022.

[3] Z. Li *et al.*, "Metamorphic robustness testing for DeepL translation," *J. Phys.: Conf. Ser.*, vol. 2456, no. 1, p. 012018, 2023.

[4] Y. Deng, X. Zheng, T. Zhang, H. Liu, G. Lou, M. Kim, and T. Y. Chen, "A declarative metamorphic testing framework for autonomous driving," *IEEE Trans. Software Eng.*, vol. 49, no. 4, pp. 1964–1982, 2023.

[5] J. Troya, S. Segura, and A. Ruiz Cortés, "Automated inference of likely metamorphic relations for model transformations," *J. Syst. Softw.*, vol. 136, pp. 188–208, 2018.

[6] P. C. Cañizares, P. Gómez-Abajo, A. Núñez, E. Guerra, and J. de Lara, "New ideas: Automated engineering of metamorphic testing environments for domain-specific languages," in *SLE*. ACM, 2021, pp. 49–54.

[7] P. Gómez-Abajo, P. C. Cañizares, A. Núñez, E. Guerra, and J. de Lara, "Automated engineering of domain-specific metamorphic testing environments," *Inf. Softw. Technol.*, vol. 157, p. 107164, 2023.

[8] S. Segura, G. Fraser, A. B. Sánchez, and A. Ruiz Cortés, "A survey on metamorphic testing," *IEEE TSE*, vol. 42, no. 9, pp. 805–824, 2016.

[9] R. Bill, M. Fleck, J. Troya, T. Mayerhofer, and M. Wimmer, "A local and global tour on MOMoT," *Softw. Syst. Model.*, vol. 18, no. 2, pp. 1017–1046, 2019.

[10] S. Segura, J. A. Parejo, J. Troya, and A. Ruiz Cortés, "Metamorphic testing of RESTful web APIs," *IEEE Trans. Software Eng.*, vol. 44, no. 11, pp. 1083–1099, 2018.

[11] J. Ahlgren *et al.*, "Testing web enabled simulation at scale using metamorphic testing," in *ICSE (SEIP)*, 2021, pp. 140–149.

[12] P. C. Cañizares, A. Núñez, J. de Lara, and L. Llana, "MT-EA4Cloud: A methodology for testing and optimising energy-aware cloud systems," *J. Syst. Softw.*, vol. 163, p. 110522, 2020.

[13] M. Jiang, T. Y. Chen, F. Kuo, Z. Zhou, and Z. Ding, "Testing model transformation programs using metamorphic testing," in *SEKE*. Knowledge Systems Institute Graduate School, 2014, pp. 94–99.

[14] K. Du, M. Jiang, Z. Ding, H. Huang, and T. Shu, "Metamorphic testing in fault localization of model transformations," in *SOFL+MSVL*. Springer, 2019, pp. 299–314.

[15] X. He, X. Chen, S. Cai, Y. Zhang, and G. Huang, "Testing bidirectional model transformation using metamorphic testing," *Inf. Softw. Technol.*, vol. 104, pp. 109–129, 2018.

[16] M. Boussaa, O. Barais, G. Sunyé, and B. Baudry, "Leveraging metamorphic testing to automatically detect inconsistencies in code generator families," *Softw. Test. Verification Reliab.*, vol. 30, no. 1, 2020.

[17] N. B. Chaleshtari, F. Pastore, A. Goknil, and L. C. Briand, "Metamorphic testing for web system security," *IEEE Trans. Software Eng.*, vol. 49, no. 6, pp. 3430–3471, 2023.

[4] https://wordnet.princeton.edu/