Generating Test Data for Blackbox Testing from UML-Based Web Engineering Content and Presentation Models

Quyet-Thang Huynh[1], Dinh-Dien Tran[1], Duc-Man Nguyen[2], Nhu-Hang Ha[2], Thi-Mai-Anh Bui[1], and Phi-Le Nguyen[1(✉)]

[1] School of Information and Communication Technology, Hanoi University of Science and Technology, Hanoi, Vietnam
{thanghq, anhbtm}@hust.soict.edu.vn, trandinhdien@gmail.com, lenp@soict.hust.edu.vn
[2] Duy Tan University, Da Nang, Vietnam
{mannd, hatnhuhang}@duytan.edu.vn

Abstract. Software testing is a process that produces and consumes huge amounts of data. Thus, the test data is usually either gathered manually by the testers or randomly generated by tools. The manual method consumes lot of time and highly depends on the testers' experience while the random approach faces the problem of redundant test data caused by identical use cases. By leveraging the concept of Model-based testing, this paper provides a novel method of testing to save the cost of manual testing and to increase the reliability of the testing processes. In Model-based testing, test cases and test data can be derived from different models. In this paper, we present a technique to generate test data from UML-based Web Engineering (UWE) presentation model for web application testing by using formal specification and Z3 SMT solver. We also build a model-based testing Eclipse Plug-in tool called TESTGER-UWE that generates test data based on the model of UWE for the web application. We evaluate the proposed methods by applying them to generate test data for an Address Book project of UWE. Experimental results show that our proposed methods can reduce the time significantly when generating test data for automation test tools such as Selenium, Katalon, Unit test, etc.

Keywords: Web application testing · Model-based testing · Test case generation · UML-based Web Engineering

1 Introduction

The UWE is an object-oriented approach, which was presented by the end of the 90s [1, 2]. This concept aims to find a standard for building models of analyzing and designing web systems based on object-oriented hypermedia design method (OOHDM) [3], relationship management methodology (RMM) [4], and web search and data mining (WSDM) [5]. These models are built at the different phases of the software development process and represent different views of the Web application corresponding to the different concerns. UWE follows a strict separation of concerns in the

T. Q. Duong et al. (Eds.): INISCOM 2019, LNICST 293, pp. 207–219, 2019.
https://doi.org/10.1007/978-3-030-30149-1_17

early stages of the development and implements a model-driven development process. In UWE, UML diagrams are exploited to visualize the models. By such ways, UWE can represent the structural aspects of the different views and provide support for model-driven web applications development [5]. Model-driven software development stresses the use of models at all levels of the software development process. This result changes the way software is designed, maintained, and tested [6]. According to [7]: "Testing often accounts for more than 50% of the required effort during system development".

A testing cycle encompasses three main parts: (i) Test case generation, (ii) Test execution, and (iii) Test evaluation. Test case and test data generation is perhaps the most complex and challenging part. Testing is a cumbersome task which needs to assure customer satisfaction and product safety. Automated test data generation is one of the main factors that contribute to the quality of automated testing. It is used for automatically generating test data for software under test (SUT) during software testing. Automation in the test data generation process could reduce testing expenses and increase the reliability of the entire testing process. For these reasons, automated test data generation has remained a topic of interest for the past four decades [9]. In many cases, model-based testing is more efficient than other testing techniques due to its possibility in generating large test suites and test data to provide evidence for accurate system implementation [8, 10].

In this paper, we aim at proposing a technique to generate test cases and test data from the UWE presentation model for Web application testing. We used the results of the previous study of Nguyen et al. [29] on the generation of test data using the formal specification and Z3 SMT Solver. From UWE presentation model and content models, they are transformed to XMI file, from which the formal specification file (called myDSL-Domain Specific Language defined by [29]) is generated for each Presentation class. Then the generation test engine invokes Z3 SMT Solvers to generate test data. We also build a plug-in tool called TESTGER-UWE that supports to transform XMI to myDSL and generate test data. The proposed method applies to the Address Book project of UWE.

The rest of this paper is organizing as follows: Sect. 2 provides the related studies; the proposed method is presenting in Sect. 3; In Sect. 4, we apply the proposed approach to generate test data for an Address book case study; Sect. 5 presents the results and discussion; Sect. 6 concludes the paper and describes our future works.

2 Related Works

UML is most generally used to provide a standard way to visualize the design of a system and also widely used for test case generations. There are many types of research in recent years about various techniques for the generation of test cases from UML diagrams.

Wang, et al. [11] proposed use case modeling for system tests generation. The proposed technique uses use case specifications, a domain model, a class diagram and constraints to generate executable system test cases. Oluwagbemi and Asmuni [12] presented an enhanced method for generating test cases from various UML diagrams.

A robust scheme for collecting artifacts from the underlying diagrams of the software under test was proposed. The intermediate representation of the artifacts is in the form of a tree over which the traversal of contents generates test cases. Anbunathan and Anirban [13] developed a method using basis path testing approach for test cases generation from UML class diagrams. From the class diagrams, a corresponding state chart diagram has been drawn and then is converted into a control flow graph. Test cases are generated manually as well as automatically, and the effectiveness of test cases has been performed using mutation analysis. Results show a significant decrease in cost as compared to other existing techniques.

Vinaya and Ketan [14] have presented a method for the generation of test cases from UML use case diagrams, class diagrams, and sequence diagrams, and then transform it into a Sequence Diagram Graph. A data dictionary is presented in the form of Object Constrained Language (OCL). The UML diagrams are drawn with the help of magic draw tool and then exported to XML format. The XML file has been parsed in java for extracting different nodes of the graph and generates all set of the scenarios from start node to end nodes.

Papadopoulos and Walkinshaw [15] presented a model-inference driven testing framework that is designed to support the inference-driven test generation for programs that are not sequential. The framework is designed to be modular; it is not necessarily tied to a specific model inference or test generation framework and can be in principle applied to any executable program, without the need for access to the source code.

The framework is deliberately flexible and uses C4.5 algorithm to infer decision trees from program executions and uses the Z3 solver to generate and execute tests from it [15]. The authors provided an openly-available Java implementation that can be extended to handle different types of programs, models, and test generators. The authors developed an evaluation of three openly-available programs and indicated that inference-driven testing could produce better test sets more efficiently than random testing.

According to Jain and Porwal survey, most of the researches proposed an approach to test data generation based on actually executing the program, analyzing the dynamic data flow, and using a function minimization method. These researches use popular heuristic approaches like Genetic Algorithm, Simulating Annealing, Particle Swarm Optimization, Ant Colony Optimization has been widely applied to search for effective test data. These approaches are verified to be more optimized than the random technique [16–22].

Currently, on the market, there are some tools to support test data generation online and offline for Data Driven Testing such as Generate data, Mockaroo, Yan Data Ellan [26–28] but these tools only let testers create data generation fields, select predefined data types and generate data. Their limitation is not to modify data size constraints and other user-defined constraints.

From the literature review, we could observe that there are different methods available to generate test cases and test data using different techniques. Studies focused on test case generation from UML diagrams by converting UML models to intermediate graphs and generating test cases. Other studies on test data generation are mostly based on the program code using meta-heuristics algorithms and optimization techniques.

3 A Novel Approach for Test Data Generation from UWE Content and Presentation Models

A metamodel represents these model elements and their relationships. UWE is compatible with the MOF exchange metamodel and therefore with XMI-based XML exchange format tools. The advantage of UML CASE tools which support for UML profiles or UML extension mechanisms can be used to create UWE models of web applications such as automatic model generation.

The presentation model provides an abstract view of the user interface (UI) of a web application. It is based on the navigation model. It describes the basic structure of the user interface, such as UI components (e.g., text, images, anchors, forms) used to represent navigation nodes. The UI components do not represent specific components of any presentation technology, but only describe what functionality is required at that particular point in the user interface. The basic elements of the presentation model are presentation classes, which are directly based on the nodes from the navigation model. Presentation classes may contain other presentation elements. In the case of UI components, such as text or images, the presentation properties associated with the navigation attribute containing the content will be displayed.

Based on our previous research on model transformation with OCL integration [23, 24] and development of rules and algorithms for code in UWE [30], we continue to expand and develop test case/test data generation technique for generating code, as well as test data based on UWE metamodels. In this context, we propose a new approach to generate test data using XMI file, domain specific language and Z3 SMT Solvers from the study of Nguyen and et al. [29]. The proposed framework is shown on Fig. 1.

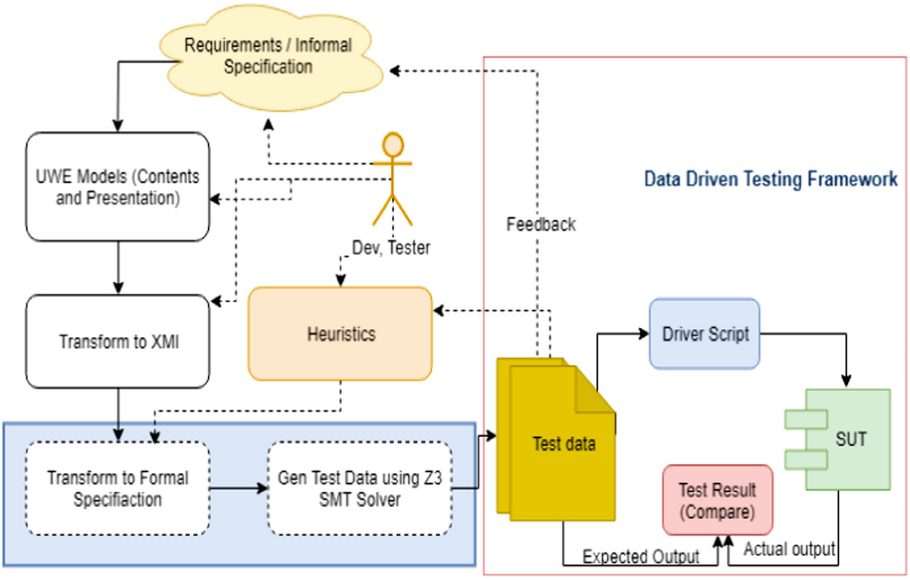


Fig. 1. The proposed framework for Test Data Generation.

The process of developing Web Application with model transformation techniques, using MagicUWE will include 5 steps as follows. We propose the technique for test data generation from UWE Content and Presentation Models to be implemented in step 3, step 4 and step 5, according to the framework shown in Fig. 1.

Step 1. Build up the charts in term of Content, Navigation, Presentation, Process structure and/or Process flow.
Step 2. Use MagicDraw to transform Content, Presentation Models to XMI file (e.g. Fig. 2)

```
    </packagedElement>
  - <packagedElement xmi:id="AAAAAAFpreBuJ4/4Ip4=" name="Contact" xmi:type="uml:Class" visibility="public"
    isActive="false" isLeaf="false" isFinalSpecialization="false" isAbstract="false">
      + <ownedMember xmi:id="AAAAAAFpreZmqZC/lAw=" xmi:type="uml:Association" visibility="public"
        isDerived="false">
        <ownedAttribute xmi:id="AAAAAAFpreCo95Akclg=" name="name" xmi:type="uml:Property" type="String_id"
            visibility="public" isLeaf="false" isID="false" isDerived="false" aggregation="none" isUnique="false"
            isOrdered="false" isReadOnly="false" isStatic="false"/>
        <ownedAttribute xmi:id="AAAAAAFpreDKqJArW+E=" name="email" xmi:type="uml:Property" type="String_id"
            visibility="public" isLeaf="false" isID="false" isDerived="false" aggregation="none" isUnique="false"
            isOrdered="false" isReadOnly="false" isStatic="false"/>
    </packagedElement>
  - <packagedElement xmi:id="AAAAAAFpreEukJAym04=" name="Address" xmi:type="uml:Class" visibility="public"
```

Fig. 2. An example of XMI file of Content model

Step 3. Leverage suggested functions to transfer XMI to formal specifications DSL. We utilize the results of previous research, proposed by Nguyen et al. [29] regarding the use of formal language to specify software requirements, the myDSL edit tool is built upon xText and DSL (Java environment), Fig. 3 shows the formal specification (called myDSL) of the Contact Class in Presentation model.

```
1   enum Contact {SUCCEESS FAIL INVALID}
2   function Contact (String name, String email)
3   define VALIDname {name.length > 8 && name.length <30}
4   define VALIDemail {email.contain(@) && email.contain(.)}
5   define conFail {!VALIDname || !VALIDemail}
6   precondition {name.length > 0 && name.length <255}
7   precondition {email.length > 0 && email.length <255}
8   testcase {
9         "SUCCESS" VALIDname && VALIDemail
10        "FAIL" conFail
11        "INVALID"
12  }
13  run
14
```

Fig. 3. myDSL of Contact class.

Step 4. Then, the engine generates data by transferring specification from the 3rd step to Z3 SMT language and call Z3 SMT solvers to find a set of solutions (test cases and test inputs). Z3 offers a compelling match for software analysis and verification tools since several common software constructs map directly into

supported theories. It is best used as a component in the context of other tools that require solving logical formulas. Figure 4 show the screenshot of generating test data.

Step 5. The results provide the test data in the format of XLS or CSV that shown in Table 1 (an example data), Tester and Developer can use these results for different testing purposes, especially for Data Driven Testing method of web applications. They also can use the results of test data to do some heuristics to optimize the data results by adding other constraints to the myDSL file and performing test generation again.

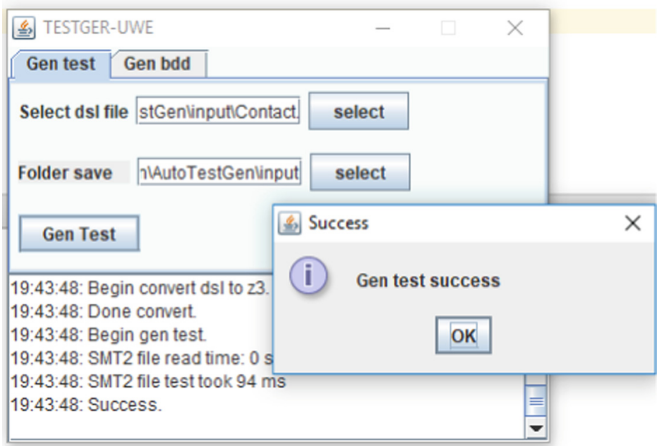


Fig. 4. Screenshot of generating test data tool

In Black Box Testing the code is not visible to the tester, functional test cases can have test data meeting following criteria:

- No data: Check system response when no data is submitted
- Valid data: Check system response when Valid test data is submitted
- Invalid data: Check system response when InValid test data is submitted
- Illegal data format: Check system response when test data is in an invalid format
- Boundary Condition Dataset (BVA): Test data meeting boundary value conditions
- Equivalence Partition Data Set (EPC): Test data qualifying your equivalence partitions.

4 Case Study: Address Book Web Application

Address Book with Searches is a typical example, used as a case study in UWE engineering research [25]. In this study, we also use this example to illustrate the proposed technique given in Sect. 3. This is an address book of contacts. Each contact

will contain a name, two phone numbers (main and alternative), two postal addresses (main and alternative), an e-mail address and a picture. The page will publish the details of the contact(s) matching a filtering condition. Users can create new, edit, update and search the contacts.

Figure 5 shows the content model of the Address Book with Searches, with the classes defined for Address-Book, Contact, Address, and Phone [25]. Figure 6 indicates the presentation model. The address book page contains the Introduction section and the Contacts list. For each contact, the corresponding email, phone and address fields are displayed [25].

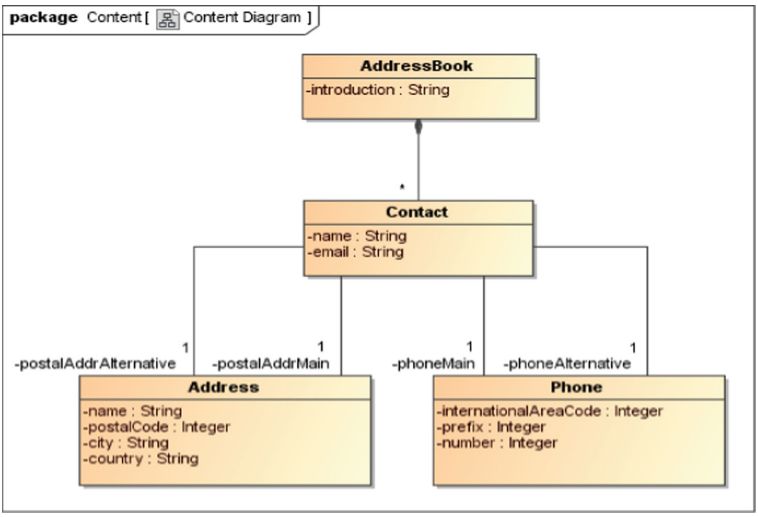


Fig. 5. UWE content model of the Address Book with Searches [25]

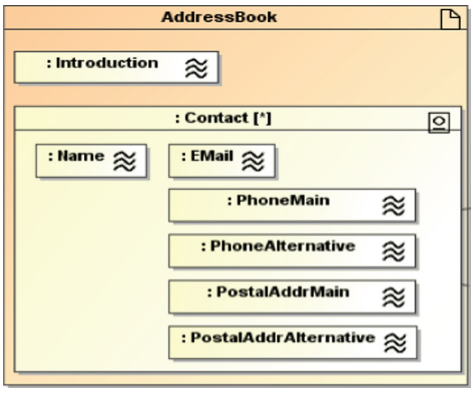
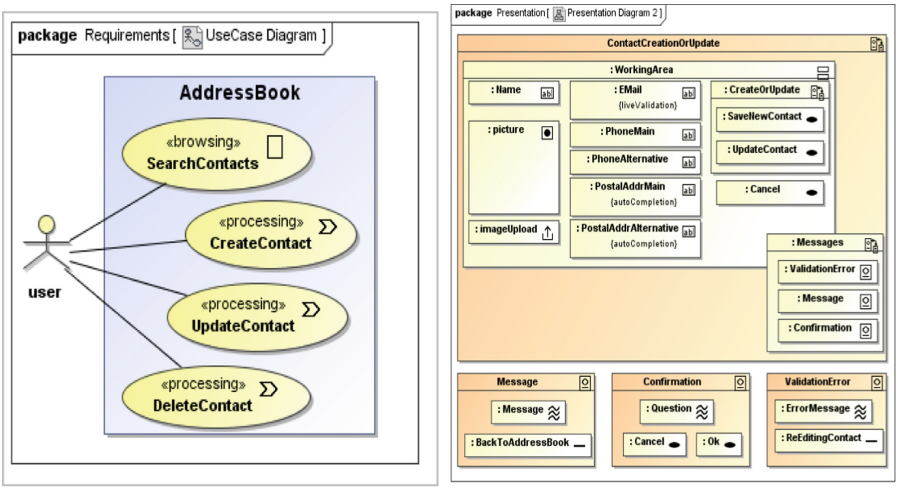


Fig. 6. UWE presentation model of a simple Address Book [25]

Table 1. Example test data for Contact form

Name	Email	Result
oJRNjn	dmC3Ih	Fail
hhC4fCkA	0ZQ6kf@qe9Y.ca	Success
m	x	Fail
6de	FeX	Fail
yrw9bobAMpLyOrAfw	6elcYlQV	Fail
5u	j7	Fail
jfyK8Yi	SsQtPWd	Fail
5GhPr0LXw	oJ3WWV@NShJV.bMN	Success
ntZt	y13 M	Fail
Hj%rD	G6amh@CO.Va	Fail



(a) (b)

Fig. 7. (a) Use case of Address Book, (b) Presentation models [25]

UWE specifies Web applications following the separation of concerns, i.e. modeling content, navigation structure, and presentation separately. Increasing functionality of the web application suggests making a detailed elicitation of requirements. Figure 7 (a) is the use case of the project. Figure 7(b) shows the presentation model of the running case study. The container form is selected to provide a more intuitive representation of pages. The ContactCreationOnUpdate page contains Contact's fields to be updated, buttons, image, message forms.

Apply the proposed method in Sect. 3 by following 6 steps:

Step 1 – Open AddressBookContent.mdzip in MagicDraw.
Step 2 – Export UWE content and presentation models to XMI format.

Step 3 – Transform XMI to DSL.
Step 4 – Call gentest engine to generate test data from DSL input file.
Step 5 – Validate the output and add some Heuristics and/or modify DSL file and re-generate test data.

Steps 3 to 5 are features of TESTGER-UWE plug-in tool.

Use the output to different purposes such as Selenium, Katalon, Testcomplete or Data Driven Testing framework.

5 Results and Discussion

Experimental results for Address Book application (including Simple Address Book, Address Book with Search feature and Address Book with Update Content), depending on the number of input fields on each form, or attribute fields in classes of Content models, the data type of each field as well as their size that are adapted from XMI to each respective DSL specification. For data fields at Presentation model, it must be based on the class diagram of a Content model to determine the type and size (or data constraints). The data generated for the fields is described in Table 2.

Table 2. The result of generating test data for content and presentation models

Class of content/presentation	Data type	Coverage of test data	No. Rows generated
AddressBook	String	Random values; minlength, maxlength, BVA, EPC (len), Valid data, Invalid data	100
Contact	String	Random values; minlength, maxlength, BVA, EPC (len), Valid data, Invalid data	200
Address	String	Random values; minlength, maxlength, BVA, EPC (len), Valid data, Invalid data	200
	Integer	Random values; min, maxint, BVA, EPC, valid data, invalid data	200
Phone	Integer	Random values; min, maxint, BVA, EPC, valid data, invalid data	200–1000
Picture	Integer	Random values; min, maxint, BVA, EPC, valid data, invalid data	200
SearchForm	String	Random values; minlength, maxlength, BVA, EPC (len), Valid data, Invalid data.	100
WorkingAreaForm	String	Random values; minlength, maxlength, BVA, EPC (len), Valid data, Invalid data.	200
	Number	Random values; min, maxint, BVA, EPC, valid data, invalid data	200

In this study, we only tested about 100 records data for each class/form. Data may be generated more by adjusting the data generation parameters. The data types supported in this study are Numeric, String (string length), Boolean type. Other data types will be studied in the future. The generated data corresponding to each data type is

declared/bound to valid data, invalid data, invalid format, using BVA and EPC to optimize output and detect the corner error.

Table 3. The comparison of 3 online tools with TESTGER-UWE

	TESTGER-UWE	Generatedata	Mockaroo	Yan Data Ellan
Format output data	CSV, XLS	CSV, XLS, JSON, SQL, XML	CSV, XLS, JSON, SQL	CSV, XLS, JSON, SQL, XML
Data coverage	**Valid, invalid data, invalid format, BVA, EPC**	Valid, invalid data	Valid, invalid data	Valid, invalid data
Generate type	Random	Random	Random	Random
Change/add constraints	**Yes**	No	No	No
Number row of data at a time	**5000+**	5000	5000	10000
Meaningful of data	No	Yes	Yes	Yes
Ease of use	**Plug-in/standalone**	Online	Online	Online
Expected result	**Yes**	No	No	No

Bold values in this table are advanced features of TESTGER-UWE compared to other tools.

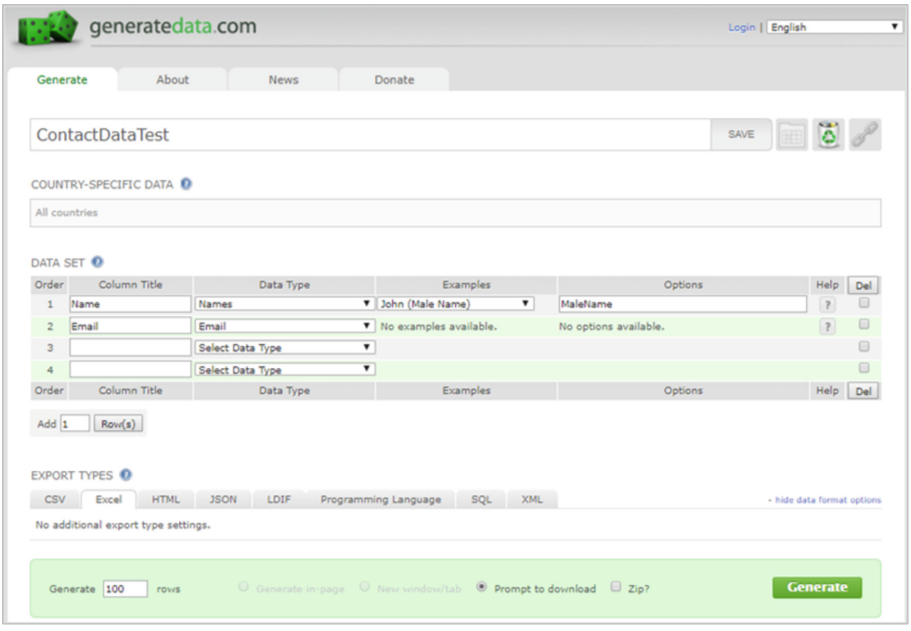
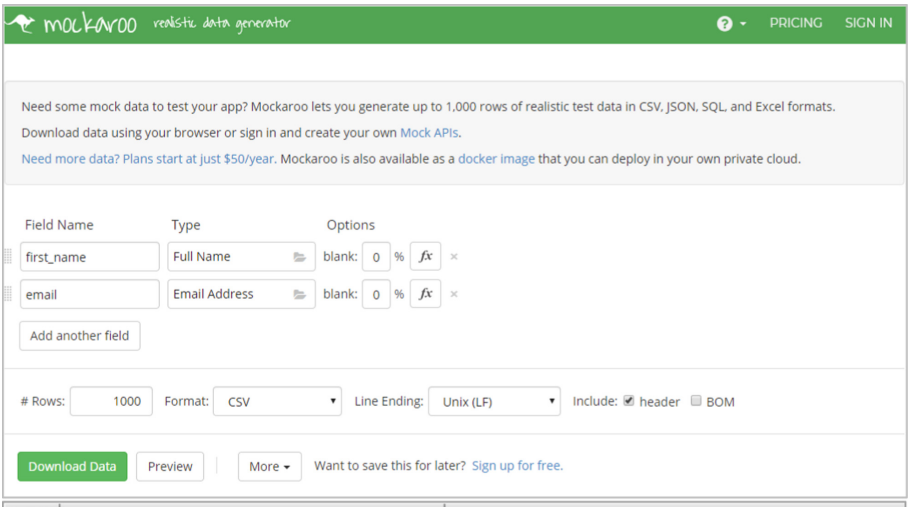


Fig. 8. Generatedata.com screenshot

Comparing experimental results with the Generate data [26], Mocka-roo [27], Yan Data Ellan [28] shows that these tools have many advanced features such as generating data into SQL, XML, Firebase, JSON, and more meaningful data. However, most of them have fees (to pay), do not generate data to cover cases of invalid data, data sizes, wrong format, additional constraints to optimize, adjust generated data as TESTGER-UWE tool. Table 3 presents a comparison of 3 online tools with TESTGER-UWE. Figures 8 and 9 are a screenshot of Generatedata.com and Mockaroo.com and generated data.



	name	email
1	Asian false vampire bat	rravenhill0@google.com
2	Cat, tiger	demes1@cbslocal.com
3	Peccary, white-lipped	lgorton2@barnesandnoble.com
4	White-winged black tern	pstollery3@qq.com
5	Baleen whale	fsidebotton4@mashable.com
6	Zebra, common	ahaggis5@hugedomains.com
7	Blackish oystercatcher	atassell6@barnesandnoble.com
8	Seal, northern fur	dscougal7@pcworld.com

Fig. 9. Mockaroo.com screenshot and generated data example

6 Conclusion and Future Work

In this paper, we proposed a method to generate test data from UWE Content and Presentation. Specifically, our method first converts the content of the UWE Content and Presentation into XM. We also developed a tool named TESTGER-UWE tool,

which generates the specification DSL and call Z3 SMT solver to generate test data. Our proposed approach is not only suitable for unit testing and Data Driven Testing but also can be applied to various testing purposes.

In the future, we will expand our approach to cover other types of data and generate test scripts for unit test or automated tests using Selenium, Katalon.

Acknowledgments. This research is funded by Hanoi University of Science and Technology under Grant number T2018-PC-015.

References

1. Baumeister, H., Koch, N., Mandel, L.: Towards a UML extension for hypermedia design. In: France, R., Rumpe, B. (eds.) UML 1999. LNCS, vol. 1723, pp. 614–629. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-46852-8_43
2. Wirsing, M., et al.: Hyper-UML: specification and modeling of multimedia and hypermedia applications in distributed systems. In: Proceedings of 2nd Workshop. German-Argentinian Bilateral Programme for Scientific and Technological Cooperation, Konigswinter (1999)
3. Schwabe, D., Rossi, G.: The object-oriented hypermedia design model. Commun. ACM **38** (8), 45–46 (1995)
4. Isakowitz, T., Stohr, E.A., Balasubramanian, P.: RMM: a methodology for structuring hypermedia design. Commun. ACM **38**(8), 34–44 (1995)
5. Koch, N., Knapp, A., Zhang, G., Baumeister, H.: UML-based web engineering. In: Rossi, G., Pastor, O., Schwabe, D., Olsina, L. (eds.) Web Engineering: Modelling and Implementing Web Applications. Human-Computer Interaction Series, pp. 157–191. Springer, London (2008). https://doi.org/10.1007/978-1-84628-923-1_7
6. Valverde, F., Valderas, P., Fons, J., Pastor, O.: A MDA-based environment for web applications development: from conceptual models to code. In: International Workshop on Web-oriented Software Technology (IWWOST 2007), in conjunction with ICWE (2007)
7. Baker, P., Dai, Z.R., Grabowski, J., Haugen, P., Schieferdecker, I., Williams, C.: Model-Driven Testing: Using the UML Testing Profile. Springer, Heidelberg (2007)
8. Utting, M., Legeard, B.: Practical Model-Based Testing: A Tools Approach. Morgan Kaufmann, Burlington (2006). ISBN-10: 0123725011
9. Mahmood, S.: A systematic review of automated test data generation techniques. Master thesis, Software Engineering, School of Engineering, Blekinge Institute of Technology (2007)
10. Polamreddy, R.R., Irtaza, S.A.: Software testing: a comparative study model-based testing vs test case-based testing. Master thesis, Software Engineering, School of Engineering, Blekinge Institute of Technology (2012)
11. Wang, C., Pastore, F., Goknil, A., Briand, L., Iqbal, Z.: Automatic generation of system test cases from use case specifications. In: Proceedings of the 2015 International Symposium on Software Testing and Analysis, ISSTA 2015 (2015). https://doi.org/10.1145/2771783.2771812
12. Oluwagbemi, O., Asmuni, H.: An approach for automatic generation of test cases from UML diagrams. Int. J. Softw. Eng. Appl. **9**(8), 87–106 (2015)
13. Anbunathan, R., Anirban, B.: Dataflow test case generation from UML Class diagrams. In: 2013 IEEE International Conference on Computational Intelligence and Computing Research (2013). https://doi.org/10.1109/iccic.2013.6724144

14. Vinaya, S., Ketan, S.: Automatic generation of test cases from UML models. In: International Conference on Technology Systems and Management (ICTSM) (2011)
15. Papadopoulos, P., Walkinshaw, N.: Black-box test generation from inferred models. In: 2015 IEEE/ACM 4th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (2015). https://doi.org/10.1109/raise.2015.11
16. Jain, N., Porwal, R.: Automated test data generation applying heuristic approaches—a survey. In: Hoda, M.N., Chauhan, N., Quadri, S.M.K., Srivastava, P.R. (eds.) Software Engineering. AISC, vol. 731, pp. 699–708. Springer, Singapore (2019). https://doi.org/10.1007/978-981-10-8848-3_68
17. Mahadik, P.P., Thakore, D.M.: Survey on automatic test data generation tools and techniques for object-oriented code. Int. J. Innov. Res. Comput. Commun. Eng. **4**, 357–364 (2016)
18. Korel, B.: Dynamic method for software test data generation. Softw. Test. Verif. Reliab. **2** (4), 203–213 (1992)
19. Latiu, G.I., Cret, O.A., Vacariu, L.: Automatic test data generation for software path testing using evolutionary algorithms. In: Third International Conference on Emerging Intelligent Data and Web Technologies (2012)
20. Varshney, S., Mehrotra, M.: Search based software test data generation for structural. ACM SIGSOFT Softw. Eng. Notes **38**(4), 1–6 (2013)
21. Nayak, N., Mohapatra, D.P.: Automatic test data generation for data flow testing using particle swarm optimization. In: Ranka, S., et al. (eds.) IC3 2010. CCIS, vol. 95, pp. 1–12. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14825-5_1
22. Jiang, S., Zhang, Y., Yi, D.: Test data generation approach for basis path coverage. ACM SIGSOFT Softw. Eng. Notes **37**(3), 1–7 (2012)
23. Nguyen, T.T.L., Tran, D.D., Bui, Q.T., Huynh, Q.T.: Integration MDA techniques in solving a class of web application with similar structure. In: 2015 ANU/SEED-Net Regional Conference for Computer and Information Engineering, Hanoi, 1–2 October 2015, pp. 78–83 (2015). ISBN 978-604-938-689-3
24. Tran, D.D., Huynh, Q.T., Tran, Q.K.: Model transformation with OCL integration in UWE. In: FICTA2018: 7th International Conference on Frontiers of Intelligent Computing: Theory and Applications, 29–30 November 2018
25. http://uwe.pst.ifi.lmu.de/exampleAddressBookWithSearches.html
26. http://generatedata.com/
27. Mockaroo, Random realistic test data generation in CSV, JSON, SQL, and Excel formats. https://mockaroo.com/
28. Yan Data Ellan. http://www.yandataellan.com/
29. Nguyen, D.M., Huynh, Q.T., Nguyen, T.H., Ha, N.H.: Automated test input generation via model inference based on user story and acceptance criteria for mobile application development. Int. J. Softw. Eng. Knowl. Eng. (2019). ISSN 1793-6403
30. Tran, D.-D., Huynh, Q.-T., Bui, T.-M.-A., Nguyen, P.-L.: Development of rules and algorithms for model-driven code generator with UWE. In: SOMET (2019)