OPEN ACCESS
Saudi Journal of Engineering and Technology
Abbreviated Key Title: Saudi J Eng Technol
ISSN 2415-6272 (Print) |ISSN 2415-6264 (Online)
Scholars Middle East Publishers, Dubai, United Arab Emirates
Journal homepage: https://saudijournals.com

Review Article

Comparative Analysis of Modeling and Software Testing Tools: A Survey

Hayatullahi Bolaji Adeyemo*

Department of Computer Science Usmanu Danfodiyo University, Sokoto, Nigeria

DOI: 10.36348/sjet.2021.v06i10.007 | **Received:** 17.08.2021 | **Accepted:** 21.09.2021 | **Published:** 30.10.2021

***Corresponding author:** Hayatullahi Bolaji Adeyemo

Abstract

There are varieties of paradigms for testing software systems with the aim of uncovering faults and improving the quality of software products. One of the popular paradigms is model-based testing paradigm, where system's behaviours are captured and denoted in a model for adequate abstract representation. Evidence has shown that automated testing can incredibly improve the testing efficiency. Central to such test automation is the use of appropriate testing tools. With the existence of a wide spectrum of software testing tools, it is difficult to decide which of the tools to choose and where to start the testing processes. It should be noted that irrespective of how similar the operational processes of software tools are; they are distinct in their respective overall composition. In order to make a choice of which tool is most appropriate to suit a system testing requirements, tester needs to understand some vital information related to the candidate tools such as availability, ease of use, programming skills needed, language and platform support, among others. In this paper, some of the most popular model-based testing tools on the market, both open source and commercial, are analyzed. The promising future of model-based testing is also proposed. This survey will help software testers, beginners or experts, to lay their hands on the appropriate model-based testing tools.

Keywords: Model-based testing, software testing tools, test automation, UML, modelling, survey.

I. INTRODUCTION

Recent advances in modeling and testing of software systems have led to the emergence of a better way to investigate the quality of a software system using ultramodern techniques and tools appropriate for the modern day software systems. These techniques have been referred to as model-based testing (MBT) and have surpassed the traditional way of testing software systems in terms of efficiency and effectiveness.

Migration from manual testing to automated testing is becoming more popular than ever before. Automated testing is the central backbone to model-based testing as the latter leverages the power of test automation. One of the reasons why test automation is becoming popular among software testing practitioners is that as soon as the tests are created, they can be executed over and over without incurring superfluous efforts/cost (Mlynarski, Güldali, Weißleder, & Engels, 2012). To get started with MBT, having idea of what tools are required is a key to unlocking the difficulties of a successful MBT activities. The tools include modeling tools, MBT tools – to generate test cases.

Software development lifecycle processes remain incomplete without software testing, irrespective of the paradigm in use. The processes of software testing are arguably the most expensive software development activities. Software testing remains crucial in development processes irrespective of the paradigm, scale, techniques, principles and models.

In recent time, as the complexity of software systems is increasing there is need for more sophisticated tools to cope with the increased complexity of the systems. Testing broadly involves tools, cases, procedures, and processes. Software testing tools are designed in an attempt to accelerate the testing processes. The purpose of this work is to equip testers with some guidance when choosing and using testing tools to carry out testing activities.

Test automation becomes more appealing for its pass-fail results especially when you have a lot of tasks ahead and/or the tasks involve repetition of some subtasks. Even though, automation has its own challenges, it has an enormous set of values. Automation can be seen as a powerful testing tool to address every test objective but not every test case.

The rest of this paper is structured as follows. Section II presents the background of automated testing, software testing tools and model-based testing. Section III discusses the future of model-based testing. Section IV describes model-based testing tools while the comparative analysis of the tools is presented in Section V. The paper concludes and presents future work in Section VI.

II. BACKGROUND

In this section, test automation is briefly discussed taking into consideration the reason why automated testing is gaining more adoption among researchers and practitioners. The concept of using tools to support software testing activities is also discussed concisely. Also, model-based testing along with its procedures and benefits is presented.

A. Automated Testing

In testing software systems, a tester interacts with the software artefact to ensure the conformance of the functionality as well as the documentation of both expected and unexpected behaviours. This is known as manual testing. Automated testing is the process of controlling the test execution and comparing the results to the expected behaviours with little or no intervention of human.

Recently, there is a rapid shift from manual testing to automated testing in software testing industries. Since there is no guarantee that the delivered software products would have no defects, it becomes necessary to test the products repeatedly throughout every phase of the development in order to improve quality. These tasks can be tedious, especially when complex systems are under test. The repetitive nature of testing can render manual testing prone to human errors, apart from its time-consuming disadvantage.

The best approach to improve the efficiency, effectiveness, and coverage of testing activities is through test automation. Large software development firms have since believed that software test automation is critical for the success of their activities. Although, smaller development firms consider it as too expensive to implement or to get started, automated testing gives a huge payoff in the long run for big software development organizations. There are different types of testing that can be automated depending on the nature of the application. These include unit testing, functional testing, regression testing, integration testing, etc. Automated testing is essentially done through automated specialized tools. Lately, Artificial Intelligence (AI) is finding its way into software testing automation.

B. Software Testing Tools

Software testing tools involve any tool that helps in automating any of the testing activities ranging from test planning, test design, test implementation & execution and test reporting. These activities need to be repeated during the development lifecycle. Since the software is prone to changes and modification during the development, every time the software is changed the testing activities need to be repeated. It is evident that carrying out this repetition manually is not only expensive and error-prone but also time-consuming (Emami, Sim, & Sim, 2011). One can see pretty easily that the use of testing tools is tantamount to automating test automation. To this effect, some of the benefits ("Benefits of using tools to support Software testing," n.d.) of tool supports to software testing include (but not limited to): reduction of task repetition, improved consistency, test reusability, faster defect detection, enhanced test coverage and improved overall testing efficiency.

C. Model-based Testing

In model-based testing (MBT), the behaviour of interest of the system under test is represented and captured by models, which are in turn used to generate test cases, and execution (Veanes *et al*., 2008). Since construction of models is part of the activities in software development, the mindset of having an MBT is concretized right from the beginning of the development life cycle. Having this in mind helps both the developers and testers to focus on developing testable applications. It is noteworthy that test automation is the solid base upon which model-based testing is basically built (Dias Neto, Subramanyan, Vieira, & Travassos, 2007; Pajunen, Takala, & Katara, 2011). As a consequence, automatic test cases are generated from the models. Whenever the requirements of the systems change, the models essentially get modified and hence adequate amendment of the test cases. This, in essence, helps to reduce maintenance of test suites. MBT has been known to increase coverage through automated scripts used in combination with some famous testing tools. According to Capegimini's Continuous Testing Report (SauceLabs, n.d.), the trend of software testing indicates MBT would be leading in the next few years. Cost, implementation time reduction, improved testing coverage, and increased quality are known to be among the reasons why MBT is popularly adopted (Gurbuz & Tekinerdogan, 2018). With the incorporation of AI and machine learning, MBT will be more cost effective and time-saving. MBT approaches have also been used in security testing (Felderer, Zech, Breu, Büchler, & Prestschner, 2016), safety (Gurbuz & Tekinerdogan, 2018). MBT has a set of systematic steps in carrying out the testing activities. Fig-1 presents the model-based testing workflow. The requirements of the system under test are collected from which a model of the system is generated. There are various types of models to be used in this process. The choice of a particular model indeed depends on the nature of the system and the type of testing paradigm to be employed. Test generation criteria is chosen based on the model representation and test suite is generated creating test scripts as reusable results. The execution of

the test suite is automated against the software under test and the result of the execution is then analysed.

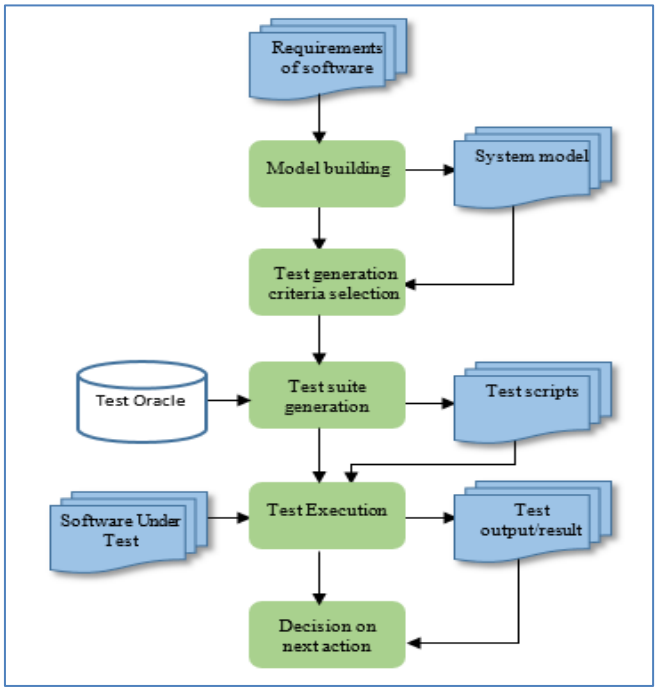

Fig-1: Model-based testing workflow

III. FUTURE OF MODEL-BASED TESTING

Although, in most context, there is no genuine substitute to make software testing completely automated. But model-based testing has tremendously improved the ease of testing by making most of the tasks automated (Mlynarski *et al*., 2012). Central to model-based testing is the accurate encoding of the behaviours of the system under test using appropriate models. Crucial to a successful MBT is the selection of appropriate level of abstraction. Having an explicit model representing some of the system's behaviours of interest, can go a long way to assist in structuring the process of creating, reusing, reproducing, and executing test cases (Gurbuz & Tekinerdogan, 2018). Understanding the current state-of-the-affair will eventually help in postulating the direction of research in the domain. MBT has been extensively useful in different application domains as follows: automotive, medical, power consumption, avionics, robotics (Micskei, Szatmári, Oláh, & Majzik, 2012), and railways (Herzner, Schlick, Schütz, Brandl, & Krenn, 2010). These domains point to the current scope of its applicability. More emphasis is given to the adoption of MBT for safety (Gurbuz & Tekinerdogan, 2018).

The challenge here is that in an attempt to consider the improvements and changes in the system under test, there is need to have a representation of the system model's evolution. There is need to have some revision on the test models. Model-based testing tools can help to overcome these challenges. Tools at different stages of testing need to be integrated to one another. There usually exist some challenges of achieving such integrations. It may be that the tools are from different vendors and there is no interface provided to integrate them.

IV. MODEL-BASED TESTING TOOLS

The problem of writing test scripts and generating test cases contributes to the cost and inefficiency of testing process. Model-based testing aims at assisting in improving the process through automation and reducing the cost. MBT is becoming more popular than ever before, as a result of advanced tool and support for methodological techniques it offers. Model-based testing tools have been found to be helpful in addressing the above-mentioned problems. MBT tools are continuously improving and thereby making the learning curve of the MBT adoption become shallower. This becomes evident, according to a survey (Kramer, 2020), with 75% dropping of the total number of hours needed to accomplish optimal value out of MBT from 2014 – 2019.

This survey is not the first one in the field of software testing tools. (Yang, Li, & Weiss, 2009) presented a survey of some testing tools focusing on the coverage measurement. The study helps in selecting appropriate testing tools depending on the testing requirements. (Emami *et al*., 2011) carried out a preliminary study on different software testing tools that are open source. This, according to the authors, will support a broad span of testing activities. The authors of (Arif & Ali, 2019) discussed a variety of mobile application testing techniques and tools identifying some of the challenges of the tools. Similarly, (Nachiyappan & Justus, 2015) presented a comparative survey of cloud computing tools, challenges and trends.

The selection process of model-based testing tools starts with searching different sources such as review papers, software tools repositories, and so on. In an attempt to obtain a list of relevant testing tools, both manual and automated searches are used. Manual search involves checking conferences and journals, which focus on presentation of tools such as *International Conference on Open Source software and Development Tools* and *International Journal on Software Tools for Technology Transfer***.** While searching from search engines is regarded as automated search. After identifying a list of potential candidate tools for the analysis, a simple step was followed to include such tools in the final list to be considered. Fig-2 shows the steps in a flowchart.

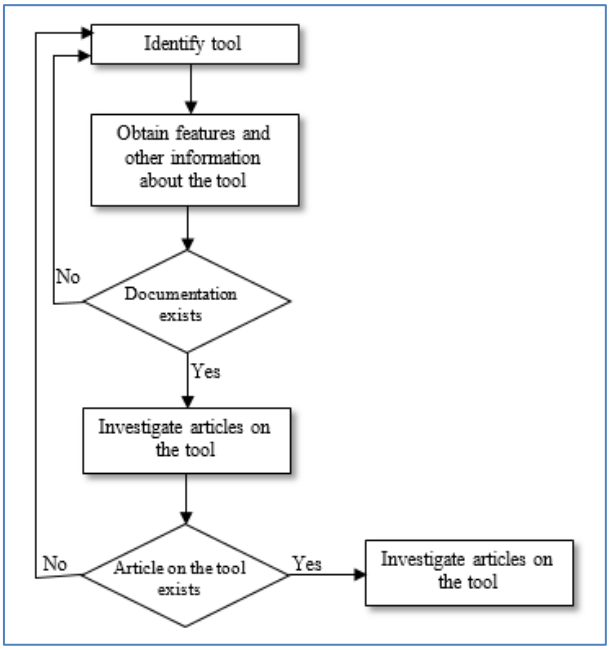

Fig-2: Tool selection flowchart

V. COMPARATIVE ANALYSIS OF MODEL-BASED TESTING TOOLS

In this work, a number of MBT tools are presented in terms of some factors such as developer, platform and language support, cost, ease of use, programming skills, report generation and data source. Table-1 presents a concise comparison of the surveyed model-based testing tools.

Table-1: Comparison of Model-based Testing Tools

Tools	Developer	Language support	Platform support	Availability/ cost	Ease of utilization	Programming skills	Report generation	Data source (AI)
fMBT	Intel	C++, AAL/ Python	Linux	Open source	Relatively easy to use	Yes	Yes	Yes
GraphWalker	Python Software Foundation	Java FSM	Windows Linux Mac OS	Open source	Easy to use	Java, Python	Yes	Yes
JSXM	Dimitris Dranidis	XML, Java	Windows Linux	Open source	X	Yes	X	SXM
MaTeLo	All4Tec	X	Windows Linux, Mobile	Commercial	Easy	No	Yes	Many
MBTsuite	sepp.med gmbh	X	X	Commercial	X	No	Yes	Yes
Modbat	Cyrille Artho and others (Artho *et al.*, 2013)	Scala, Java	All platforms	Open source	Easy	Yes	Yes	Java or Scala file
MoMuT	AIT & TU Graz	UML & Timed automata	Window, Linux	Open source, Academic	Fairly easy	Yes	Partially	X
TestCast	Elvior LLC	UML, Java	Windows, Linux	Commercial	Relatively easy	Yes	Yes	UML

For each tool surveyed, the language support comprises the modelling paradigm used to abstract the behaviour of the system under test and also the coding language upon which the tool was implemented. Some of the entries are denoted as "X", which means the appropriate information to fit the cell is unavailable or not specified.

fMBT ("fMBT | 01.org," n.d.) is an automated tool for generating and executing tests. Among the features of fMBT are model editor, test data generator and tools for studying logs. In case there are different interfaces, fMBT offers generic adapters for integrating those interfaces. It has the ability to run offline and online testing on Linux platforms. As a result, generation and execution of tests are launched via command line and can be coded in C++, JavaScript, Python, and shell script.

GraphWalker ("GraphWalker," n.d.) is a useful MBT tool whose operation is based on a concept of graphs. Models are created in an editor provided by the tool called Studio. Apart from model creation, the editor is also used in editing and verifying the models' correctness. An interesting feature of GraphWalker is its ability to be integrated to Java projects as it can be used in testing as a maven dependency. A typical model created by GraphWalker is basically a graph consisting of vertices and edges. While the vertices symbolize assertions and verifications, edges denote transitions and/or actions. Finite State Machines, FSM are also supported models as they are considered as a type of model by GraphWalker. The tool navigates through the model to generate its corresponding test data. It can be used for online and offline test generation. GraphWalker has been proved to be useful in testing other artefacts written in other language than Java or Python ("Practical Model-Based Testing — Say 'Hello MBT' | by Ofer Rivlin | CyberArk Engineering | Medium," n.d.).

JSXM ("JSXM: A Tool for Automated Testing," n.d.) is another MBT tool built on top of Stream X-machine (XSM), which is a modelling formalism of system states with the ability to model both control and data of a system. Memory and processing functions are incorporated into finite state machines to obtain SXMs (Dranidis, Bratanis, & Ipate, 2012).

MaTeLo ("Model Based Testing Tool - Discover MaTeLo | MaTeLo - Model Based Testing," n.d.) is a handy graphical model-based testing tool, which generates reusable test cases from the behavioral design of the application under test. The tool also imports the requirements specification, which are used to examine the coverage of the test cases generated after executing them against the application.

Apart from quick generation of test cases, MBTsuite ("Home | MBTsuite - The Model Based Testing Tool," n.d.) tool – a model-based testing tool, also continuously track the test cases and update them effortlessly. The tool carries out its test data generation from UML, which is a graphical model of test design based on different coverage criteria. The test cases generated by MBTsuite tool are not platform-dependent and as such can be transformed into various formats to be used by any test automation or management tool.

Modbat ("Modbat," n.d.) is a specialized MBT tool for event-driven systems. The system behaviour is transformed into an abstract model (extended finite-state machines, EFSM), which is consequently used to generate test data (Artho *et al*., 2013). The tool has a strong compatibility with Java even though it is based on Scala.

MoMuT ("MoMuT – Model-based Mutation Testing," n.d.) is a tool for generating test cases based on MBT paradigm with concentration on fault-based testing. It has support for some modelling languages such as UML (State Charts and Action Systems), Assume-Guarantee Contracts and Timed Automata (Krenn *et al*., 2015).

TestCast ("Elvior | TestCast MBT | Model based testing," n.d.) also generates test cases from UML State Machines. The tool has different editions and testers make selection based on the features needed. TestCast is a particularly useful MBT tool with applications in API testing, web/mobile testing, and distributed systems testing. The systems' behaviours of interest are modelled as Input/Output Transition Systems. The tool has community support to help users in fixing any issues that may arise in the course of using the tool.

There are multitude of languages and models that support MBT, one of the most popular is UML. This is also depicted in the results of the analysis shown in Table-1.

Out of the eight MBT tool analysed, three (3) of them are commercial while the remaining ones are either open source or academic.

VI. CONCLUSION AND FUTURE WORK

Software testing tools are becoming indispensable in both academia and industries. The difficulty of choosing an appropriate testing tool still remains challenging. By its very nature, model-based testing tools are distinct in their respective overall composition. Each tool has its unique way of modelling the behaviour of interest of the system under test.

In this paper, some selected model-based testing tools are analytically compared. The selected tools are analysed in terms of language & platform

support, models for test case generation, availability (commercial or open source), ease of utilization, [required] programming skills, report generation capability, and data source. Even though, incorporation of AI into the future trend of MBT is promising; the involvement of human elements is still indispensable. Humans are needed in successfully executing MBT processes.

VII. ACKNOWLEDGMENT

Usmanu Danfodiyo University is greatly acknowledged for providing the useful facilities used in successfully carrying out this research.

VIII. REFERENCES

- Arif, K. S., & Ali, U. (2019). Mobile application testing tools and their challenges: A comparative study. *2019 2nd International Conference on Computing, Mathematics and Engineering Technologies, ICoMET 2019*, 3–8. https://doi.org/10.1109/ICOMET.2019.8673505
- Artho, C. V., Biere, A., Hagiya, M., Platon, E., Seidl, M., Tanabe, Y., & Yamamoto, M. (2013). Modbat: A model-based API tester for event-driven systems. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *8244 LNCS*, 112–128. https://doi.org/10.1007/978-3-319-03077-7_8
- Benefits of using tools to support Software testing. (n.d.). Retrieved June 13, 2021, from http://www.sustainable-hyderabad.in/benefits-of-using-tools-to-support-testing/
- Dias Neto, A. C., Subramanyan, R., Vieira, M., & Travassos, G. H. (2007). A survey on model-based testing approaches: A systematic review. *Proc. - 1st ACM Int. Workshop on Empirical Assessment of Software Engineering Languages and Technologies, WEASELTech 2007, Held with the 22nd IEEE/ACM Int. Conf. Automated Software Eng., ASE 2007*, 31–36. https://doi.org/10.1145/1353673.1353681
- Dranidis, D., Bratanis, K., & Ipate, F. (2012). JSXM: A tool for automated test generation. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *7504 LNCS*(December 2015), 352–366. https://doi.org/10.1007/978-3-642-33826-7_25
- Elvior | TestCast MBT | Model based testing. (n.d.). Retrieved June 14, 2021, from https://elvior.com/model-based-testing/
- Emami, S. A., Sim, J. C. L., & Sim, K. Y. (2011). A survey on open source software testing tools: a preliminary study in 2011. *Fourth International Conference on Machine Vision (ICMV 2011): Computer Vision and Image Analysis; Pattern Recognition and Basic Technologies*, *8350*, 83502Y. https://doi.org/10.1117/12.920508
- Felderer, M., Zech, P., Breu, R., Büchler, M., & Prestschner, A. (2016). Model-based security testing: a taxonomy and systematic classification. *Software Testing, Verification and Reliability*, *26*, 119–148. https://doi.org/10.1002/stvr
- fMBT | 01.org. (n.d.). Retrieved June 14, 2021, from https://01.org/fmbt
- GraphWalker. (n.d.). Retrieved June 14, 2021, from http://graphwalker.github.io/
- Gurbuz, H. G., & Tekinerdogan, B. (2018). Model-based testing for software safety: a systematic mapping study. *Software Quality Journal*, *26*(4), 1327–1372. https://doi.org/10.1007/s11219-017-9386-2
- Herzner, W., Schlick, R., Schütz, W., Brandl, H., & Krenn, W. (2010). Towards generation of efficient test cases from UML/OCL models for complex safety-critical systems. *Elektrotechnik Und Informationstechnik*, *127*(6), 181–186. https://doi.org/10.1007/s00502-010-0741-2
- Home | MBTsuite - The Model Based Testing Tool. (n.d.). Retrieved June 14, 2021, from https://mbtsuite.com/
- JSXM: A Tool for Automated Testing. (n.d.). Retrieved June 14, 2021, from http://jsxm.org/
- Kramer, A. (2020). *2019 Model-based Testing User Survey : Results*.
- Krenn, W., Schlick, R., Tiran, S., Aichernig, B., Jöbstl, E., & Brandl, H. (2015). MoMut::UML model-based mutation testing for UML. In *2015 IEEE 8th International Conference on Software Testing, Verification and Validation, ICST 2015 - Proceedings*. Institute of Electrical and Electronics Engineers Inc. https://doi.org/10.1109/ICST.2015.7102627
- Micskei, Z., Szatmári, Z., Oláh, J., & Majzik, I. (2012). A concept for testing robustness and safety of the context-aware behaviour of autonomous systems. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *7327 LNAI*, 504–513. https://doi.org/10.1007/978-3-642-30947-2_55
- Mlynarski, M., Güldali, B., Weißleder, S., & Engels, G. (2012). *Model-Based Testing: Achievements and Future Challenges*. *Advances in Computers* (Vol. 86). Elsevier Inc. https://doi.org/10.1016/B978-0-12-396535-6.00001-6
- Modbat. (n.d.). Retrieved June 14, 2021, from https://people.kth.se/~artho/modbat/
- Model Based Testing Tool - Discover MaTeLo | MaTeLo - Model Based Testing. (n.d.). Retrieved June 14, 2021, from https://matelo.all4tec.com/
- MoMuT – Model-based Mutation Testing. (n.d.). Retrieved June 14, 2021, from https://momut.org/
- Nachiyappan, S., & Justus, S. (2015). Cloud testing tools and its challenges: A comparative study. *Procedia Computer Science*, *50*, 482–489.

https://doi.org/10.1016/j.procs.2015.04.018

- Pajunen, T., Takala, T., & Katara, M. (2011). Model-based testing with a general purpose keyword-driven test automation framework. *Proceedings - 4th IEEE International Conference on Software Testing, Verification, and Validation Workshops, ICSTW 2011*, 242–251. https://doi.org/10.1109/ICSTW.2011.39
- Practical Model-Based Testing — Say "Hello MBT" | by Ofer Rivlin | CyberArk Engineering | Medium. (n.d.). Retrieved June 14, 2021, from https://medium.com/cyberark-engineering/practical-model-based-testing-say-hello-mbt-b16292ffff06
- SauceLabs. (n.d.). Continuous Testing Cloud. Retrieved from https://saucelabs.com/
- Veanes, M., Campbell, C., Grieskamp, W., Schulte, W., Tillmann, N., & Nachmanson, L. (2008). Model-based testing of object-oriented reactive systems with spec explorer. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *4949 LNCS*, 39–76. https://doi.org/10.1007/978-3-540-78917-8_2
- Yang, Q., Li, J. J., & Weiss, D. M. (2009). A survey of coverage-based testing tools. *Computer Journal*, *52*(5), 589–597. https://doi.org/10.1093/comjnl/bxm021