Users, Developers and Managers vs. Traditional and Non-traditional Computing Mechanisms in Automated Software Testing: A Systematic Literature Review

Sara Gračić, Vuk Vuković
University of Novi Sad, Faculty of Economics
Subotica, Serbia
saritta4u@gmail.com, vuk.vukovic@ef.uns.ac.rs

Abstract - **Testing is important in producing high quality software and can be done manually and/or automatically. Automation has potential to reduce testing effort (e.g. in developing test cases, generating test data) and speed up testing cycles. This paper aims to present state-of-the-art achievements in automated testing through its diverse application in practice and investigate how users, developers and managers are influenced by this phenomenon. Three scientific databases were chosen and the search resulted in 46 final papers. Majority of sources proposed traditional testing techniques, while one proposed chemical reaction networks. Although automation is present in developing mobile apps, embedded systems, automotive industry etc., most developers do not properly automate testing. Two sources marked user reviews as important, while managers are drawn to potential savings in time, effort and money.**

Keywords-automated software testing; embedded systems; automotive industry; non-traditional computing mechanisms; SLR

I. INTRODUCTION

With the proliferation of software in every aspect of our lives, its high quality is of at-most-importance, especially in safety-critical systems e.g. in healthcare, automotive industry, avionics. Software testing, as an important verification and validation activity [1], ensures that the demanded quality is achieved through detection and removal of bugs/defects [2].

On the other hand, proliferation has also led to increase in software's size and complexity [3], so testing software of that magnitude consumes a lot of time and effort [3] and up to 50% of total software project costs are related to testing [4], [5].

Since automated software testing has potential to reduce high effort invested in "manually developing and maintaining test cases" [6], decrease costs [4], improve testing effectiveness [4], [6] and speed-up testing cycles [6], there has been an increase in interest regarding automation of software testing [4]. Therefore, research in this field has advanced over the years, which has resulted in developing a wide range of available testing techniques and tools [6].

After the Introduction, the rest of the paper is structured as follows: Section II presents research methodology (research questions, chosen scientific databases, search strings, inclusion and exclusion criteria, and how the review was conducted); Section III extracts the review data and discusses findings, while conclusion is given in Section IV.

II. RESEARCH METHODOLOGY

Since systematic literature reviews (SLRs) are designed to provide detailed information about a specific phenomenon [7], they are becoming a vital part of scientific research [8]. SLRs also have to be fair and thorough [7], otherwise their scientific value will be questionable. They represent a mean of identification, evaluation and interpretation of available research [7] related to a topic of interest. A SLR consists of three main phases: planning a review, conducting a review and reporting review findings [7].

A. Planning the Review

Since automation of software testing is becoming very important for both industry and academia, this SLR is focused on summarizing the available knowledge and identifying current trends in the area of automated software testing.

1) Defining Research Questions: Based on the set objective, following research questions were constructed. **RQ1**: How many studies focused on automated software testing were published from 2016 till January 14th 2021? **RQ2**: What were the main topics discussed in these papers? **RQ3**: What are current trends in the area of automated software testing? **RQ4**: How are users, developers and managers influenced by automated software testing?

2) Defining Search Strings for Scientific Databases: To provide answers to research questions, a SLR was conducted to identify topics and trends discussed in academia. Three electronic databases were chosen for conducting SLR: IEEE xplore, AIS eLibrary and Science Direct. Research string was set in the following way: "automated software testing" and applied in the above stated electronic databases.

3) Defining Inclusion and Exclusion Criteria: Publishing period was set to last 6 years - from 2016 till January 14th 2021, to collect the most recent research findings. Only papers published in journals and conferences were included in the

initial population. Grey literature (e.g. blog posts, white papers) did not enter the initial population. Only primary and secondary studies in the domain of automated software testing were examined. Language was not set as an obstacle for entering the initial population, although it may become one in the next stages (if title or abstract are not in English).

B. Conducting the Review

Advanced search was chosen in all databases. After applying search string and criteria "all fields" in AIS eLibrary, "all metadata" in IEEE xplore and "find articles with these terms" in Science Direct, search resulted in initial population of 95 hits, structured in the following order: IEEE explore (29), AIS library (8) and Science Direct (58).

Titles, keywords and abstracts were evaluated to ensure that a study is in function of answering research questions. All papers that were not in the field of interest were eliminated. 1 paper had abstract in Portuguese and English and was relevant to SLR; 3 papers did not have full access [3], [5] and [9]. Only 3 papers from AIS eLibrary, 29 from IEEE xplore and 14 from Science Direct entered the final population. Due to limited number of pages, extraction of results and discussion are joined into Section III.

III. EXTRACTING RESULTS AND DISCUSSING FINDINGS

Authors of this paper aimed to provide a summary of state-of-the-art knowledge regarding automated software testing and investigate how users, developers and managers are involved in this process. Thus, a SLR was conducted and 46 papers were of interest for this research, which gives answer to **RQ1** *(How many studies focused on automated software testing were published from 2016 till January 14th 2021?).*

Findings, extracted from those studies, were categorized in groups based on the context in which automated software testing was presented and are discussed in the following paragraphs, thus providing answers for **RQ2** *(What were the main topics discussed in these papers?)* and **RQ3** *(What are current trends in the area of automated software testing?).*

Software has become a part of our everyday life, which has led to increase in its size, complexity [3] and demand for its high quality has also risen. Consequently, testing effort has been increased and testing participates 50% in total project costs [4], [5]. Therefore, academia and industry have shifted focus towards potential benefits of automated software testing in various fields.

Application of automated software testing on the **Web** was in focus of several papers. Since open source tools have potential for productivity savings, productivity of two open source automated testing tools, measured in time, was compared and time difference was significant [10]. This finding indicates that appropriate choice of testing tool could potentially decrease testing time.

Testing web apps is not without difficulties - performance testing can be problematic due to unpredictable load, response time etc. [11]. Therefore, correlation of performances testing tools was done in terms of usability and performance parameters in [11].

To achieve high level of automation and thoroughness in testing of web API, artificial intelligence techniques were applied for autonomous software failure detection. Bots were used to generate test inputs and correctness of test outputs was evaluated through: "1) patterns learned from previous executions of the SUT" and "2) knowledge gained from analyzing thousands of similar programs". Developed prototype automatically detected bugs in some real-world APIs [12].

General instructions for manually testing Web sites are acceptable, but could those be automated? In order to automate these instructions, semantic usage patterns were presented in [13]. The model recorded "general topics behind the individual steps of interactions". Models were extracted from existing test descriptions, regardless of their form i.e. they could either be in natural language or in the form of system tests. These patterns can also be applied for applications they were not designed for. Applications can be tested automatically for behavioral anomalies [13].

Software testing is also knowledge-intensive, so it could be improved by semantic web technologies, e.g., ontologies, "which have been frequently used in knowledge engineering activities" [1]. A SLR was conducted and benefits of using semantic web enabled software testing for industry and academia were presented; testing activities that would have use of these techniques were pointed out, as well as problems in application of semantic web enabled techniques [1].

Several sources were focused on **fuzzing.** Fuzzing is a popular [14] automated testing technique that "looks for vulnerabilities by causing crashes through the introduction of invalid, unexpected, or random data" as inputs and improves software's robustness and security [15]. It possesses conceptual simplicity, deployment barriers are low and there is abundance of "empirical evidence" that fuzzing is effective in "discovering real-world software vulnerabilities" [14].

Fuzzing was applied in [15] to test Janus WebRTC media server's behavior, which allowed fixing several important software issues. A "unified, general-purpose model of fuzzing together with taxonomy of the current fuzzing literature" was presented in [14].

With the expansion of mobile devices, fuzz tests have been applied to mobile platforms as well. While most studies are focused on GUI and implementation at the application level [16], it was argued in [16] that detecting vulnerabilities in lower levels is important, especially since that would affect many Android users. Therefore, genetic algorithms for efficient fuzz testing for Android app installation process were proposed. Black-box fuzzing tool created "more unique crashes" in less time and detected new and existing bugs [16].

Machine learning plays an important role in automated software testing. An automated text clustering approach with a semi-automated version for clustering errors in term of their root causes was proposed to save effort in triaging and fixing bugs. Applied approach outperformed other baseline methods for classification and clustering [17].

Machine learning was also used to automatically prioritize test cases, based on the level of failure probability. Metrics

about software under test and historical commit messages were used for creating data file for generating decision trees in Weka. The model was used for prioritizing tests according to prediction where defects will occur [18].

Achieving minimum level of coverage for every system build is desirable. However, executing all test cases and generating new ones for all classes for every commit is not feasible, so selection of subset of classes to be tested has to be made. "Knowing a priori the branch coverage that can be achieved with test data generation tools might give some useful indications" [19], so machine learning was used to develop models for predicting branch coverage that would be achieved by test data generation tools in [19].

Multi-Objective Ant Lion Optimization algorithm resolved multi-objective optimization of coverage based test data. The algorithm was validated through comparison with random testing and conventional genetic algorithms [20].

In [4], EvoPSO algorithm, which is based on swarm intelligence paradigm, was implemented in EvoSuite tool for test data generation. Its performance was evaluated on SFIIO dataset and it was showed that EvoPSO was efficient and competitive.

Finding faults is complex and time consuming. So, many Spectrum Based Fault Localization techniques for automated fault localization in single-fault software have been developed. However, they are not always effective for multi-fault software [3]. Therefore, Chaos-based Genetic Algorithm for Multi-fault Localization based on Spectrum Based Fault Localization for automated fault localization in single and multi-fault programs was proposed. Suspicion for every program statement was calculated and ranked. Lower rank meant higher probability of a statement being faulty. Experiments showed that this technique outperformed Spectrum Based Fault Localization techniques [3].

Neuroevolution of Augmenting Topologies algorithm was applied for automated generation of new test suits or coverage improvement of existing test suits. Test suites were automatically generated for white box testing [5].

Testing automation is important in different areas of **industry**. Automated test case generation in *practice and research* was analyzed and "lessons learned from transferring software testing research results to industry" were presented in [6]. Experience in automated smoke testing in a cross-platform *game application* was presented with "specific details and challenges associated with setting up and maintenance of day-to-day automated testing activities" in [21].

Automated testing method for improving testing efficiency for *smart TV* was proposed in [22]. Test platform was "designed to send the python script to the Android smart TV automatically through its Android debug bridge interface". Their method reduced testing time and more errors were tested, when compared to manual testing [22]. Architecture for automated *warship* software testing was proposed in [23].

Automation is highly present in *embedded systems.* Therefore, authors in [24] conducted a SLR. Many approaches were presented. E.g. black-box testing for embedded software with specific test automation tool was conducted. Automated test input generation in Java for embedded device was evaluated. A search-based approach for automated model-in-the-loop testing of continuous controllers was presented. An automated approach for test suites reduction was presented. For one embedded system, automation of test case generation with the help of Genetic algorithms was applied. Authors of [24] stated that automation can be successfully applied not just in execution of test cases, but in test case design and test evaluation as well.

A method for automated testing analysis for *IoT* was proposed in [2]. If automated, software testing for IoT enables test reusability and repeatability, thus decreasing test coverage cost and time. Authors compared "automated software testing for IoT transactions and system peripherals such as sensor, DMA controller, internal counters, physical layer, and virtual layer using IoT specification based software testing" [2].

Software testing, as a time consuming task, should be automated, since that could shorten development time. Although automation has its advantages and disadvantages, management is especially interested in automation's effects on software's cost, quality and time [25], so authors in [25] investigated critical factors related to cost and return of/from automation, which is a very expensive activity. Three different software products were investigated and their experiments showed "positive effects of test automation on cost, quality and time to market".

Automotive software has gained importance, due to software's role in controlling vehicles (e.g. "window controller, smart-key system, and tire pressure monitoring system"). Thus, human effort in testing automated software is quite high and, consequently, the industry is focused on finding ways to ensure that automotive software is of the highest quality, but with reduced human effort [26]. Authors applied concolic testing for automotive software developed by Hyundai Mobis in [26]. They developed a framework MAIST "that automatically generates the test driver, stubs and test inputs to a target task". In concolic testing, MAIST achieved 90.5% branch coverage and 77.8% MC/DC coverage on the integrated body unit (IBU) software. It also reduced the cost of IBU coverage testing through reducing the manual testing for coverage testing by 53.3%.

The same group of authors (from [26]) conducted a new study arguing that applying automated test generation in automotive software is "technically challenging because of false alarms caused by imprecise test drivers/stubs and lack of tool supports for symbolic analysis of bit-fields and function pointers in C" [27]. So, they developed MAESTRO - an automated testing framework. Test driver and stubs for a target task were built automatically, after which test inputs were generated and concolic testing and fuzzing were applied together in an adaptive way. Transformation of "target program that uses bit-fields into a semantically equivalent one that does not use bit-fields" was performed. MAESTRO "supports symbolic function pointers by identifying the candidate functions of a symbolic function pointer through static analysis". Experiments showed that their framework achieved 94.2% branch coverage and 82.3% MC/DC coverage

on the four target modules. It significantly decreased coverage testing cost through reduction of manual coverage testing effort by 58.8%. They concluded that "MAESTRO can achieve high test coverage for automotive software with significantly reduced manual testing effort" [27].

Mobile app's marketplace is characterized by strong competition [28], [29]. Therefore, creating high-quality mobile apps [28], [29] in short release cycles [28] and maintaining them [29] is necessary to ensure their commercial success and acquire new users [29].

Therefore, characteristics of mobile apps were extracted to define efficient and effective testing suitable for mobile apps [30]. One research showed that modifications due to changes in GUI occurred in 55% of modified test methods, modifications related to changes in test logic were registered in 35% and adaptations to modified application logic occurred in 27% of cases. These findings should help developers avoid fragility and reduce maintenance costs for automated test suites for Android apps [31].

TOOGLE was developed to automatically derive visual tests from "existing layout-based counterparts" or repair them when graphical changes happen. Script portability rose to 93%, while translation could repair up to 90% of visual locators in failing tests [32].

3 hybrid apps were tested on 5 different mobile devices and while manual testing showed correct functioning of tested features, automated testing showed differences between platforms [33]. A SLR conducted in [34] found 29 studies in the field of automated testing, which were classified according to testing techniques – the most common was model-based testing (8 studies).

To detect and repair bugs rapidly, researchers and practitioners propose tools for automating test process [28], [29]. But, these tools generate redundant and random inputs that lack "contextual information" and generate "reports difficult to analyze" [28]. These inputs are also insufficient for properly simulating human behavior, thus "leaving feature and crash bugs undetected until they are encountered by users" [29].

Since users can "provide contextual details about errors or exceptions detected by automated testing tools", this enables detecting bugs that would remain uncovered if teams would only rely on testing tools. Therefore, incorporating user feedback into testing is important [29] and BECLOMA - a tool for integrating **user feedback** in testing mobile apps was proposed in [28]. Connection between testing tools information and user reviews provided developers a more detailed testing report, which combined "stack traces with user reviews information referring to the same crash". Therefore, the tool facilitated diagnosis and fixing bugs and eased "the usage of testing tools" and automated "analysis of user reviews from the Google Play Store" [28].

Automating software testing is important, because it reduces test repair effort [35], so a SLR of **test breakage prevention and repair** techniques was conducted and level of automation of these techniques (manual, semi-automated and automated) was also investigated. A technique was classified as automated if it automatically detected "occurrence of breakages", automatically generated "potential test fixes, while validation of potential fixes may" have been manual. 12 papers (30%) from the SLR were focused on automation of test case repair. It was argued that there were very little or no evidence on efficiency of manual, semi-automated and automated techniques. Without evidence, it was questionable whether it was more efficient to repair or write new test cases which made practitioners reluctant in accepting these techniques [35].

Test smells, as poorly designed tests, negatively affect test and code quality, so a SLR was conducted in [36]. It was argued that practitioners often lacked skills for writing automated test scripts, which led to variety of smells; that practitioners were the ones interested in discovering new smells; that detection of 6 fixture-related smells (general fixture, test maverick, lack of cohesion of test methods, dead field, vague header setup, and obscure in-line setup) was automated and that there were metrics that indicated existence of smells.

Environment is changing quickly and for faster software delivery, other parts of the IT department are also crucial. So, authors in [37] argued that skills required for successfully implementing DevOps in an IT team were not enough empirically researched. They identified 36 skills and categorized them in 7 groups (full-stack development, analysis, functional, decision-making, social, testing and advisory skills). It was argued that comprehensive testing skills were needed in an ideal **DevOps** team [37].

A non-traditional computing mechanism – **chemical reaction networks** were used to develop ChemTest which "evaluates test oracles on individual simulation traces and supports functional, metamorphic, internal and hyper test cases. It also allows for flakiness and programs that are probabilistic". Conducted case study showed that 21% of tests were flaky; functional tests found 66.5%, while metamorphic tests found 80.4% of faults [38].

In [39], focus was on **cloud** quality testing and development of strategy for automated SaaS testing in CSB environment, while a novel cloud testing platform for software testing was proposed in [40].

When failures occur, development team has to analyze, fix and repair the problematic process, so **process mining** of production process event logs was proposed to automatically extract unit test cases in [41].

A framework (based on ISO/IEC /IEEE 291129 -2013 standard for software testing process and ISO/IEC /IEEE 291129 -2015 for software testing techniques) for automated test case generation in **gesture recognition systems** was developed in [9] and five parameters (rotation, contrast, scaling, background and noise) were used to generate test cases.

A technique for automated testing that supports white-box, unit, assertion-based and exception testing and dead code was developed and compared to commercial tools [42], while comparison of automated regression tests with continuous integration and manual testing was performed in [43]. General

model for random and systematic testing, based on a set of assumptions, was proposed in [44]. TAO (a testing tool) for specifying and generating test cases and oracles in a declarative way was proposed in [45], while automated testing tools were divided into 6 categories based on testing type in [46].

In [47], SLR showed that majority of investigated studies applied automation in different aspects of testing (generation and execution of test cases, test analysis); however, neither study had fully automated testing process. Out of 15 factors, that influence decision regarding automated testing, regression testing, maturity and economic factors were very frequent, although that did not reflect on their importance [48].

Answer to **RQ4** *(How are users, developers and managers influenced by automated software testing?)* is provided in the following paragraphs. In automated testing, **user feedback** is of at-most-importance, because they can provide developers with detailed information regarding errors and exceptions [28], [29], that would otherwise stay "bellow the radar" and therefore jeopardize quality of the final product.

On the other hand, majority of **developers** and automated testing do not go hand into hand. Firstly, [37] argues that, if implementing DevOps, a comprehensive set of testing skills (36 skills) is needed. Secondly, according to [36], majority of developers lack skills for writing automated test scripts, which leads to a vast number of test smells that distort the quality of the final product. However, those developers are the ones with the highest interest in discovering new smells. Thirdly, since automated test case repair techniques lack evidence to support their efficiency, developers are reluctant in accepting them [35].

Managers are always focused on the business side of automated testing – resources and the final product. They want high quality product with minimal human effort, decreased costs and shorter time to market, since all of the above are necessary conditions for being competitive and becoming a leader. Automated testing has the potential for reusability [2], [46], decrease in time [2], [46], and cost [2], [5], [25], [31], higher quality [25], [26], [27], shorter time to market [25] and reduced human effort [26], [27], so managers are particularly interested in applying automation in software testing.

IV. Conclusion

Testing, as a stage in development process, is vital for producing high quality software. With the proliferation of software in every aspect of our lives, its high quality is of at-most-importance, especially in safety-critical systems e.g. healthcare, automotive industry, avionics. Thus, a SLR, that aimed to provide a summary of knowledge regarding this topic, was conducted and 46 papers were in focus of this research.

Research results have led to several conclusions. Majority of sources proposed traditional testing techniques, while only 1 proposed chemical reaction networks (a non-traditional computing mechanism). Automated testing is present in automotive industry, embedded systems, mobile apps, web, IoT etc.

Most developers do not properly automate test scripts, which leads to smells and poor test and code quality, although they are interested in discovering new smells. Without evidence of efficiency of automated repair techniques, they are reluctant in accepting them. Users can provide developers important, detailed information regarding defects, which would not have been noticed just with test tools. Managers are focused on potential savings in time, effort and money automated testing could offer.

Despite many benefits automated software testing has to offer, managers need to have in mind that automation also has disadvantages, and therefore, has to be carefully conducted, i.e. benefits have to be higher than costs. It can be expected that automation will continue to expand, but in certain circumstances, it will not eliminate manual testing.

References

[1] M. Dadkhah, S. Araban and S. Paydar, "A systematic literature review on semantic web enabled software testing", Journal of Systems and Software, vol. 162, 2020.

[2] M. Padmanabhan, "A Study on Transaction Specification based Software Testing for Internet of Things," *2018 International Conference on Current Trends towards Converging Technologies (ICCTCT)*, Coimbatore, 2018, pp. 1-6, doi: 10.1109/ICCTCT.2018.8550926.

[3] D. Ghosh and J. Singh, "Spectrum-based multi-fault localization using Chaotic Genetic Algorithm", Information and Software Technology, vol.133, 2021.

[4] M. M. D. Shahabi, S. P. Badiei, S. E. Beheshtian, R. Akbari and S. M. R. Moosavi, "On the performance of EvoPSO: A PSO based algorithm for test data generation in EvoSuite," *2017 2nd Conference on Swarm Intelligence and Evolutionary Computation (CSIEC)*, Kerman, 2017, pp. 129-134, doi: 10.1109/CSIEC.2017.7940170.

[5] H. L. P. Raj and K. Chandrasekaran, "NEAT Algorithm for Testsuite generation in Automated Software Testing," *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*, Bangalore, India, 2018, pp. 2361-2368, doi: 10.1109/SSCI.2018.8628668.

[6] R. Ramler, C. Klammer and G. Buchgeher, "Applying Automated Test Case Generation in Industry: A Retrospective," *2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, Västerås, Sweden, 2018, pp. 364-369, doi: 10.1109/ICSTW.2018.00074.

[7] B. Kitchenham, "Procedures for Performing Systematic Reviews", Joint Technical Report, Computer Science Department, Keele University (TR/SE-0401) and National ICT AustraliaLtd. (0400011T.1), 2004.

[8] A. A. Khan, J. Keung, M. Niazi, S. Hussain, and H. Zhang, "Systematic Literature Reviews of Software Process Improvement: A Tertiary Study", *EuroSPI 2017*, *CCIS 748*, pp. 177–190, 2017.

[9] S. M. Hasan, M. Saiful Islam, M. Ashaduzzaman and M. A. Rahaman, "Automated Software Testing Cases Generation Framework to Ensure the Efficiency of the Gesture Recognition Systems," *2019 22nd International Conference on Computer and Information Technology (ICCIT)*, Dhaka, Bangladesh, 2019, pp. 1-6, doi: 10.1109/ICCIT48885.2019.9038214.

[10] A. Sivaji *et al*., "Software Testing Automation: A Comparative Study on Productivity Rate of Open Source Automated Software Testing Tools For Smart Manufacturing," *2020 IEEE Conference on Open Systems (ICOS)*, Kota Kinabalu, Malaysia, 2020, pp. 7-12, doi: 10.1109/ICOS50156.2020.9293650.

[11] D. Saharan, Y. Kumar and R. Rishi, "Analytical Study and Implementation of Web Performance Testing Tools," *2018 International Conference on Recent Innovations in Electrical, Electronics & Communication Engineering (ICRIEECE)*, Bhubaneswar, India, 2018, pp. 2370-2377, doi: 10.1109/ICRIEECE44171.2018.9008408.

[12] A. Martin-Lopez, "AI-Driven Web API Testing," *2020 IEEE/ACM 42nd International Conference on Software Engineering: Companion*

Proceedings (ICSE-Companion), Seoul, Korea (South), 2020, pp. 202-205.

[13] A. Rau, "Topic-Driven Testing," *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, Buenos Aires, 2017, pp. 409-412, doi: 10.1109/ICSE-C.2017.175.

[14] V. J. M. Manès *et al*., "The Art, Science, and Engineering of Fuzzing: A Survey," in *IEEE Transactions on Software Engineering*, doi: 10.1109/TSE.2019.2946563.

[15] A. Amirante, T. Castaldi, L. Miniero, S. P. Romano, P. Saviano and A. Toppi, "Fuzzing Janus for Fun and Profit," *2019 Principles, Systems and Applications of IP Telecommunications (IPTComm)*, Chicago, IL, USA, 2019, pp. 1-7, doi: 10.1109/IPTCOMM.2019.8920918.

[16] V. Hatas, S. Sen and J. A. Clark, "Efficient Evolutionary Fuzzing for Android Application Installation Process," *2019 IEEE 19th International Conference on Software Quality, Reliability and Security (QRS)*, Sofia, Bulgaria, 2019, pp. 62-68, doi: 10.1109/QRS.2019.00021.

[17] X. Nguyen, P. Nguyen and V. Nguyen, "Clustering Automation Test Faults," *2019 11th International Conference on Knowledge and Systems Engineering (KSE)*, Da Nang, Vietnam, 2019, pp. 1-7, doi: 10.1109/KSE.2019.8919435.

[18] L. Butgereit, "Using Machine Learning to Prioritize Automated Testing in an Agile Environment," *2019 Conference on Information Communications Technology and Society (ICTAS)*, Durban, South Africa, 2019, pp. 1-6, doi: 10.1109/ICTAS.2019.8703639.

[19] G. Grano, T. V. Titov, S. Panichella and H. C. Gall, "How high will it be? Using machine learning models to predict branch coverage in automated testing," *2018 IEEE Workshop on Machine Learning Techniques for Software Quality Evaluation (MaLTeSQuE)*, Campobasso, 2018, pp. 19-24.

[20] M. Singh, V. M. Srivastava, K. Gaurav and P. K. Gupta, "Automatic test data generation based on multi-objective ant lion optimization algorithm," *2017 Pattern Recognition Association of South Africa and Robotics and Mechatronics (PRASA-RobMech)*, Bloemfontein, 2017, pp. 168-174, doi: 10.1109/RoboMech.2017.8261142.

[21] M. Mozgovoy, "Multiplatform Automated Software Testing: Personal Experience of a Maintainer," *2019 4th International Conference and Workshops on Recent Advances and Innovations in Engineering (ICRAIE)*, Kedah, Malaysia, 2019, pp. 1-4,

[22] K. Cui, K. Zhou, H. Song and M. Li, "Automated Software Testing Based on Hierarchical State Transition Matrix for Smart TV," in *IEEE Access*, vol. 5, pp. 6492-6501, 2017.

[23] X. Han, N. Zhang, W. He, K. Zhang and L. Tang, "Automated Warship Software Testing System Based on LoadRunner Automation API," *2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, Lisbon, 2018, pp. 51-55.

[24] V. Garousi, M. Felderer, C. M. Karapıcak and U. Yılmaz, "Testing embedded software: a survey of the literature", Information and Software Technology, vol. 104, pp. 14-45, 2018.

[25] D. Kumar and K. K. Mishra, "The Impacts of Test Automation on Software's Cost, Quality and Time to Market", Procedia Computer Science, vol. 79, pp. 8-15, 2016.

[26] Y. Kim, D. Lee, J. Baek and M. Kim, "Concolic Testing for High Test Coverage and Reduced Human Effort in Automotive Industry," *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, Montreal, QC, Canada, 2019, pp. 151-160, doi: 10.1109/ICSE-SEIP.2019.00024.

[27] Y. Kim, D. Lee, J. Baek and M. Kim, "MAESTRO: Automated Test Generation Framework for High Test Coverage and Reduced Human Effort in Automotive Industry", Information and Software Technology, vol.123, 2020.

[28] L. Pelloni, G. Grano, A. Ciurumelea, S. Panichella, F. Palomba and H. C. Gall, "BECLoMA: Augmenting stack traces with user review information," *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, Campobasso, 2018, pp. 522-526, doi: 10.1109/SANER.2018.8330252.

[29] G. Grano, A. Ciurumelea, S. Panichella, F. Palomba and H. C. Gall, "Exploring the integration of user feedback in automated testing of Android applications," *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, Campobasso, 2018, pp. 72-83, doi: 10.1109/SANER.2018.8330198.

[30] D. B. Silva, M. Medeiros Eler, V. H. S. Durelli and A. T. Endo, "Characterizing mobile apps from a source and test code viewpoint", Information and Software Technology, vol. 101, pp. 32-50, 2018.

[31] R. Coppola, M. Morisio and M. Torchiano, "Maintenance of Android Widget-Based GUI Testing: A Taxonomy of Test Case Modification Causes," *2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, Västerås, Sweden, 2018, pp. 151-158, doi: 10.1109/ICSTW.2018.00044.

[32] R. Coppola, L. Ardito, M. Torchiano and E. Alégroth, "Translation from layout-based to visual android test scripts: An empirical evaluation", Journal of Systems and Software, vol. 171, 2021.

[33] A. A. Menegassi and A. T. Endo, "An evaluation of automated tests for hybrid mobile applications," *2016 XLII Latin American Computing Conference (CLEI)*, Valparaiso, 2016, pp. 1-11, doi: 10.1109/CLEI.2016.7833337.

[34] S. Zein, N. Salleh and J. Grundy, "A Systematic Mapping Study of Mobile Application Testing Techniques", Journal of Systems and Software, vol. 117, pp.334-356, 2016.

[35] J. Imtiaz, S. Sherin, M. U. Khan and M. Z. Iqbal, "A systematic literature review of test breakage prevention and repair techniques", Information and Software Technology, vol. 113, pp.1-19, 2019.

[36] V. Garousi and B. Küçük, "Smells in software test code: A survey of knowledge in industry and academia", Journal of Systems and Software, vol. 138, pp.52-81, 2018.

[37] A. Wiedemann and M. Wiesche, "Are you ready for DevOps? Required skill set for DevOps teams", Research Papers, 123, 2018.

[38] M. C. Gerten, J. I. Lathrop, M. B. Cohen and T. H. Klinge, "ChemTest: An Automated Software Testing Framework for an Emerging Paradigm," *2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, Melbourne, Australia, 2020, pp. 548-560.

[39] V. Paulsson, V. C. Emeakaroha, J. Morrison and T. Lynn, "Cloud Service Brokerage: A systematic literature review using a software development lifecycle", *Twenty-second Americas Conference on Information Systems*, San Diego, 2016.

[40] M. C. Calpur and C. Yilmaz, "Towards Having a Cloud of Mobile Devices Specialized for Software Testing," *2016 IEEE/ACM International Conference on Mobile Software Engineering and Systems (MOBILESoft)*, Austin, TX, 2016, pp. 9-10, doi: 10.1109/MobileSoft.2016.014.

[41] D. Lübke, "Extracting and Conserving Production Data as Test Cases in Executable Business Process Architectures", Procedia Computer Science, vol. 121, pp. 1006-1013, 2017.

[42] R. Khalid, "Towards an automated tool for software testing and analysis," *2017 14th International Bhurban Conference on Applied Sciences and Technology (IBCAST)*, Islamabad, 2017, pp. 461-465.

[43] J. Alberto, J. Oliveira, S. Reis, V. Cunha, S. Pelegrini and E. P. S. Nunes, "Automação de Testes: Um Estudo de Caso", *2020, ISLA 2020 Proceedings*, 8.

[44] M. Böhme and S. Paul, "A Probabilistic Analysis of the Efficiency of Automated Software Testing," in *IEEE Transactions on Software Engineering*, vol. 42, no. 4, pp. 345-360, 1 April 2016.

[45] H-F. Guo, "A Semantic Approach for Automated Test Oracle Generation", Computer Languages, Systems & Structures, vol. 45, pp. 204-219, 2016.

[46] S. K. Alferidah and S. Ahmed, "Automated Software Testing Tools," *2020 International Conference on Computing and Information Technology (ICCIT-1441)*, Tabuk, Saudi Arabia, 2020, pp. 1-4, doi: 10.1109/ICCIT-144147971.2020.9213735.

[47] B. Uzun and B. Tekinerdogan, "Model-Driven Architecture Based Testing: A Systematic Literature Review", Information and Software Technology, vol. 102, pp. 30-48, 2018.

[48] V. Garousi and M. V. Mäntylä, "When and what to automate in software testing? A multi-vocal literature review", Information and Software Technology, vol. 76, pp. 92-117, 2016.