

Artificial Intelligence in Software Test Automation: A Systematic Literature Review

Anna Trudova, Michal Dolezel^a and Alena Buchalceva^b

Department of Information Technologies, University of Economics, Prague, W. Churchill Sq. 4, Prague, Czech Republic

Keywords: Software Testing, Test Automation, Test Tools, Artificial Intelligence, Literature Study.

Abstract: Artificial intelligence (AI) has made a considerable impact on the software engineering field, and the area of software testing is not an exception. In theory, AI techniques could help to achieve the highest possible level of software test automation. The goal of this Systematic Literature Review (SLR) paper is to highlight the role of artificial intelligence in the software test automation area through cataloguing AI techniques and related software testing activities to which the techniques can be applied. Specifically, the potential influence of AI on those activities was explored. To this end, the SLR was performed with the focus on research studies reporting the implementation of AI techniques in software test automation. Out of 34 primary studies that were included in the final set, 9 distinct software testing activities were identified. These activities had been reportedly improved by applying the AI techniques mostly from the machine learning and computer vision fields. According to the reviewed primary studies, the improvement was achieved in terms of reusability of test cases, manual effort reduction, improved coverage, improved fault and vulnerability detection. Several publicly accessible AI-enhanced tools for software test automation were discovered during the review as well. Their short summary is presented.


1 INTRODUCTION


The growing complexity of today's software systems results in an increased need for sophisticated testing techniques. Performing software testing activities manually appears to be ineffective in terms of demanding manpower consumption, low execution speed and inadequate test coverage. Those are precisely the problems which test automation could address and, in most cases, also solve. Software test automation is defined by Dustin et al. (1999) as "*management and performance of test activities, to include the development and execution of test scripts so as to verify test requirements, using an automated test tool*" (p. 4). In principle, however, test automation should be considered as a broader concept, including not only the automated test scripting and execution, but also other activities across the whole software testing process (Garousi & Elberzhager, 2017).

It is known that the software test automation has its limitations and problems (Rafi et al., 2012). As an

example, fragile automation scripts or ineffective fault detection may be mentioned. However, the limitations and problems of test automation are conceptually similar to certain issues which already have been solved by the application of artificial intelligence (AI) techniques (Last, Kandel & Bunke, 2004). On the way towards this promising vision, the book *Artificial intelligence methods in software testing* (Last et al., 2004) incorporated a set of articles and papers on the relatively new application of artificial intelligence algorithms in software testing. Generally speaking, the proposed approaches were as follows:

1. fuzzy logic for the generalization of cause-effect software testing;
2. Info-Fuzzy Networks and Artificial Neural Networks for test case generation and reduction;
3. AI planning for regenerating regression tests affected by software change;
4. case-based reasoning and C4.5 for determination of risky modules in software.

^a  <https://orcid.org/0000-0002-5963-5145>

^b  <https://orcid.org/0000-0002-8185-5208>

Another publication provided an overview of the AI techniques usable in software testing (Houranim, Hammad & Lafi, 2019). Authors of the publication focused their findings on the software testing domain, but they paid only a limited attention to software test automation. The following AI techniques in connection with test automation were mentioned: Huber Regression, Support Vector Regression (SVR), multi-layer perceptron, Hybrid Genetic Algorithms (HGA) and Natural Language Processing (NLP).

The publications mentioned above represent only a relatively small selection of possibly promising applications of AI techniques in the software testing domain. The papers included in *Artificial intelligence methods in software testing* (Last et al., 2004) pointed out to the next step that should be taken. This step was defined as a need to move forward with practical tools that implements AI algorithms not only for software testing in general, but for software test automation in particular.

At the time of writing this paper, we found no publications that would provide a full overview of AI techniques applications in software test automation. This paper intends to fill this gap. Therefore, the aim of this paper is to identify in what manner artificial intelligence is impacting the software test automation field, and to systematize the AI techniques that can be applied to the stated field. Such knowledge can enable a better understanding of given areas, their conceptual interconnection, and provide the practitioners with practical examples of AI techniques applied to various test automation activities. This paper is primarily intended for specialists from the quality engineering field. Due to that fact, it aims to give a practical, bird-eyes perspective on AI; the paper does not cover specific details with regard to the implementation details of various AI techniques.

The rest of this paper is organized as follows. Section 2 describes the systematic review process, Section 3 presents the SLR results together with answering the research questions. Conclusions are presented in Section 4.

2 SYSTEMATIC LITERATURE REVIEW

In order to accurately perform the Systematic Literature Review (SLR) focused on artificial intelligence in software test automation, the systematic process was followed according to the SLR guidelines proposed by Kitchenham and

Charters (2007). A Systematic Literature Review is a “means of identifying, evaluating and interpreting all available research relevant to a particular research question, or topic area, or phenomenon of interest” (Kitchenham & Charters, 2007, p. 3). The following subsections reflect and document the process of how the review was conducted.

2.1 Research Questions

In order to describe the role of artificial intelligence and its techniques in software test automation, the following research questions were stated:

RQ1: Which software testing activities can be improved by applying AI techniques?

RQ2: What AI techniques can be applied for improving testing activities identified during answering the RQ1?

RQ3: What are the reported benefits of AI techniques usage in software test automation?

RQ4: What AI-enhanced software tools can be pragmatically used by practitioners for software test automation activities?

2.2 Search Strategy

Several digital libraries were used as a source of the research papers, including IEEExplore (<https://ieeexplore.ieee.org>), ACM Digital Library (<https://dl.acm.org>), ScienceDirect (www.sciencedirect.com), and SpringerLink (<https://link.springer.com>). These libraries were selected due to the quality, accessibility and relevance of their content for the field of software engineering.

Search queries for each of the libraries are stated in Table 1. The following list summarizes the keywords that were identified as relevant to answer the research questions: *artificial intelligence, machine learning, computer vision, natural language processing, test automation, automated test, automated testing, software engineering, software*. The full queries can be found in Table 1.

The syntax and the keywords themselves were adapted depending on the searching-related features and limitations of each digital library. Notably, the main difference was a varying usage of specific syntax (asterisk or double quotes) for different databases.

The presented search results are accurate as of 15th September 2019. Number of total results without removing duplicate papers was 2 548.

Table 1: Search queries for digital libraries with the number of results found.

Digital Library	Search query	Number of results
ACM Digital Library	("computer vision" OR "natural language processing" OR "AI" OR "artificial intelligence" OR "ML" OR "machine learning" OR "NLP") AND ((test* AND (automated OR automation)) AND ("software engineering" OR "software"))	855
IEEEExplore	("computer vision" OR "natural language processing" OR "AI" OR "artificial intelligence" OR "ML" OR "machine learning" OR "NLP") AND ("test* automat*" OR "automat* test*") AND ("software engineering" OR "software")	455
ScienceDirect	("computer vision" OR "natural language processing" OR "artificial intelligence" OR "machine learning") AND ("test automation" OR "automated test" OR "automated testing") AND ("software engineering" OR "software")	426
SpringerLink	("computer vision" OR "natural language processing" OR "AI" OR "artificial intelligence" OR "ML" OR "machine learning" OR "NLP") AND (("test* AND automat*") OR ("automat* AND test*")) AND ("software engineering" OR "software")	812

2.3 Inclusion and Exclusion Criteria

Obtained publications were filtered according to the inclusion and exclusion criteria that are defined below. Some of the criteria are based on the fact that this paper is meant for quality engineers and, as a result of that matter, does not consider the activities related to software programming and code maintenance.

The inclusion criteria were:

IC1: publications written in English

IC2: only primary studies

IC3: publications from the software engineering domain

IC4: publications that describe the application of artificial intelligence techniques

The following exclusion criteria were specified:

EC1: publication types such as encyclopaedia, book, book chapter, conference abstract, editorials, book review, conference info

EC2: papers issued before publication of Last et al. (2004), where authors collected a representative set of papers on the topic of application of artificial intelligence techniques in software testing

EC3: publications related to training, validating and testing algorithms

EC4: publications regarding unit testing and fault localization techniques

The above stated criteria were thoroughly applied in several phases. During the first phase of the review, duplicate and incomplete publications were excluded. In addition to that, all publications were filtered by year, language and type of publication with the help of Mendeley reference management software. After duplicates were removed and the criteria (IC1, EC1, EC2) were applied, the amount of found publications was significantly reduced from 2 548 to 1 814. The

next phase involved filtering papers based on reading their titles and abstracts. Publications were evaluated by multiple exclusion and inclusion criteria that were not applied in the previous phase: IC2, IC3, IC4, EC3 EC4. As the outcome, the number of papers that were included was cut down to 227. In some cases, it was not sufficient to read only the title and abstract to identify whether a certain publication is relevant to the research or not. Therefore, in order to make a decision regarding inclusion or exclusion of aforesaid publications, the introduction and conclusion were read in addition to their titles and abstracts. The next phase of the review involved reading of the articles' full text with the intention of identifying whether each individual paper should be included into the final set or not, respecting all stated inclusion and exclusion criteria. During this phase, several papers written by the same authors regarding the same subject were discovered, although they were not complete duplicates. In order not to compromise the review's results, only the more recent publication or, in some cases, the more descriptive one was taken into final set. Once filtered based on the inclusion and exclusion criteria specified above, the set of 39 papers remained (Fig. 1). The quality of that publications was subsequently analysed and assessed according to the SLR process (Kitchenham & Charters, 2007). A more detailed information regarding the quality assessment is presented in the following Section 2.4.

2.4 Quality Assessment

For the purpose of assuring that the previously selected 39 publications are relevant and unbiased, a quality assessment was performed. To address the problem of the papers' quality, a checklist was used as it is a standardized way of performing the quality assessment.

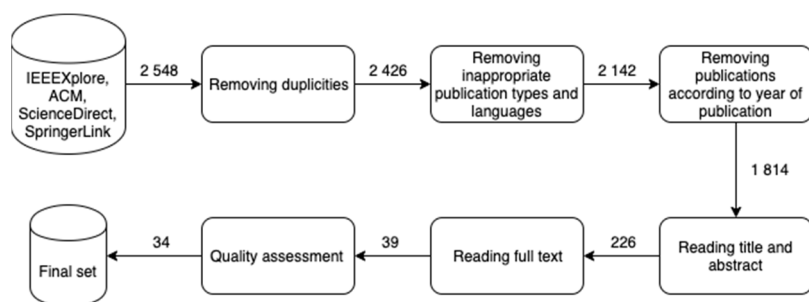


Figure 1: SLR process.

The quality checklist proposed and applied consisted of the following questions:

Q1: Are the study aims clearly stated?

Q2: Is there an adequate description of the study context?

Q3: Is there a clear statement of findings/achieved results?

Q4: Are the findings/achieved results based on multiple projects? That means, whether the solution proposed by the authors was successfully verified by its application to at least two cases in a particular context (e.g. multiple web applications)

Q5: Do the researchers discuss the validity/reliability of their results?

Answers to the stated questions were binary, with the answer being either “yes” or “no”. For each affirmative answer, the paper was given a point, in other case the point was not granted. The overall score for the paper is based on the count of points gained. The maximum achievable score was 5 points, and with that in mind, if the score was lower than 3, the paper was excluded from the final set. Table 2 summarizes the results of the quality assessment for the papers included during the previous phase. The lines marked in light grey represent the publications excluded due to their low score. Each publication has an identifier assigned to it, being used further in this paper.

After the quality assessment was performed, the final set of selected publications included 34 publications.

3 SLR RESULTS

This section presents the outcomes of the Systematic Literature Review and provides the answers to the research questions posed (RQ1, RQ2, RQ3, RQ4).

3.1 RQ1: Which Software Testing Activities Can Be Improved by Applying AI Techniques?

In the course of this literature study, 9 distinct activities were identified. These activities constitute software testing activities which have a potential to be automated and improved by applying AI techniques. The activities were identified and analysed based on the resulting set of 34 papers. A brief summary of the activities is presented in Table 3.

Some of the papers proposed approaches applicable across several testing activities. Hence, based on that fact, the publications were mentioned multiple times in all the activities they impact.

Figure 2 shows the count of papers that mentioned individual testing activities. Based on that count, it is possible to make an assumption that even the AI techniques are reportedly suitable to be used all across the testing process, some of the activities (e.g. test case or test oracle generation) attract more attention than the others.

As mentioned earlier, several software testing activities were identified during the SLR process. These are described below in more detail as they are important for the remaining research questions.

Test Case Generation. Test case generation can be defined as a process of creation of a sequence of test operations or test steps for the particular system under test (Hu et al., 2018; Li & Lam, 2005; Mariani et al., 2012; Papadopoulos & Walkinshaw, 2015; Srivastava & Baby, 2010).

Test Oracle Generation. This activity can also be titled test evaluation. Test oracles provide the feedback on the obtained outputs from the system under test. They determine whether the outputs correspond with the expected ones (Braga et al., 2018; Jin et al., 2008; Shahamiri et al., 2011).

Test Execution. The core of this activity is represented by the execution of test cases and by recording the results of those test runs. In certain cases, test execution and some other activities such as

Table 2: Quality assessment. Lines marked grey represent the excluded publications.

Identifier	Publication	Q1	Q2	Q3	Q4	Q5	Score
R1	Méndez-Porras et al. (2015)	yes	yes	no	no	no	2
R2	Sharifipour et al. (2018)	yes	yes	yes	yes	yes	5
R3	Shahamiriet al. (2011)	yes	yes	yes	no	yes	4
R4	Papadopoulos and Walkinshaw (2015)	yes	yes	yes	yes	yes	5
R5	Wotawa (2016)	yes	yes	no	no	no	2
R6	Lu et al. (2008)	yes	yes	no	no	no	2
R7	King et al. (2018)	yes	yes	yes	yes	yes	5
R8	Srivastava and Baby (2010)	no	yes	yes	no	yes	3
R9	Paradkar et al. (2007)	yes	yes	yes	no	no	3
R10	Wang et al. (2007)	yes	yes	no	no	no	2
R11	Jin et al. (2008)	yes	yes	yes	no	no	3
R12	Mariani et al. (2012)	yes	yes	yes	yes	yes	5
R13	Liu et al. (2017)	yes	yes	yes	yes	yes	5
R14	Li et al. (2011)	yes	yes	yes	no	no	3
R15	Li et al. (2009)	yes	yes	yes	no	yes	4
R16	Shen et al. (2009)	yes	yes	yes	no	yes	4
R17	Li and Lam (2005)	yes	yes	yes	no	no	3
R18	Souza et al. (2011)	yes	yes	yes	yes	yes	5
R19	Rosenfeld et al. (2018)	yes	yes	yes	yes	yes	5
R20	Gu et al. (2017)	yes	yes	yes	yes	yes	5
R21	Bhattacharyya and Amza (2018)	yes	yes	yes	no	no	3
R22	Carino and Andrews (2015)	yes	yes	yes	yes	yes	5
R23	Santiago et al. (2018)	yes	yes	yes	yes	yes	5
R24	Stocco et al. (2018)	yes	yes	yes	yes	yes	5
R25	Thummalapenta et al. (2012)	yes	yes	yes	no	no	3
R26	Bozic and Wotawa (2018)	yes	yes	yes	no	no	3
R27	Hewett and Kijisanayothin (2009)	yes	yes	yes	no	yes	4
R28	Hillah et al. (2016)	yes	yes	yes	yes	yes	5
R29	White et al. (2019)	yes	yes	yes	yes	yes	5
R30	Moghadam (2019)	yes	yes	yes	yes	no	4
R31	Sant et al. (2005)	yes	yes	yes	no	yes	4
R32	Chen et al. (2017)	yes	yes	yes	yes	yes	5
R33	Braga et al. (2018)	yes	yes	yes	yes	yes	5
R34	Vieira et al. (2006)	yes	yes	no	no	no	2
R35	Chang et al. (2010)	yes	yes	yes	yes	yes	5
R36	Fard et al. (2014)	yes	yes	yes	yes	yes	5
R37	Pan et al. (2019)	yes	yes	yes	yes	no	4
R38	Hu et al. (2018)	yes	yes	yes	yes	yes	5
R39	Choi et al. (2013)	yes	yes	yes	yes	yes	5

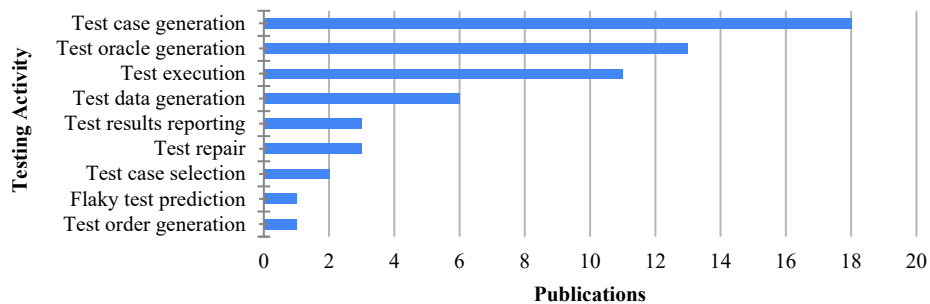


Figure 2: Number of publications by testing activity.

data generation are performed together (Bhattacharyya & Amza, 2018; Chang et al., 2010; Gu et al., 2017; Hillah et al., 2016).

Test Data Generation. Test data generation is a process of creation of the data for test cases. As an example, input values that should be relevant to each particular test can be mentioned (Liu et al., 2017; Sharifipour et al., 2018; White et al., 2019).

Table 3: Software testing activities that can be improved by applying the AI techniques.

Testing	Publication identifier
Test case generation	R4, R8, R9, R12, R15, R16, R17, R19, R22, R23, R25, R26, R28, R29, R30, R31, R36, R38
Test oracle generation	R3, R4, R9, R11, R12, R19, R26, R28, R29, R32, R33, R35, R36
Test execution	R12, R19, R20, R21, R26, R28, R30, R32, R35, R37, R39
Test data generation	R2, R13, R14, R28, R29, R39
Test results reporting	R19, R20, R32
Test repair	R24, R25, R37
Test case selection	R18, R28
Flaky test prediction	R7
Test order generation	R27

Test Results Reporting. Test results reporting is the collection of test outputs in the form of a report. This activity is performed after test execution and test evaluation activities have been finished. Such reports can possibly contain the following information: executed steps, execution status, occurred failures identification, defect reports, etc. (Chen et al., 2017; Gu et al., 2017; Rosenfeld et al., 2018).

Test Repair. Test repair is, in essence, a maintenance activity. Within the course of this activity, test scripts are adjusted to changed conditions. The need for it lays in a fact that test scripts are fragile and vulnerable to the changes introduced by developers in a newer version of the tested software (Pan et al., 2019; Stocco et al., 2018).

Test Case Selection. This activity focuses on the selection of test cases from a test suite. The selection is made based on criteria individually defined for each test case execution. Test case selection also involves removal of the duplicate, redundant or inexecutable test cases from the test set, which is typically generated by the tools (Souza et al., 2011).

Flaky Test Prediction. Flaky test is characterized as such when it reports false positive or false negative test result, when adjustment was made to the test scripts and/or to the code of the system under test (King et al., 2018). If the tests expressing similar characteristics could be identified and repaired, the overall stability and reliability of the tests can be significantly improved.

Test Order Generation. This activity is concentrated on determination of the number and order of the components under test during the component integration testing. The aim is to minimize the number of mocked components required for testing, and to select their appropriate orchestration (Hewett & Kijsanayothin, 2009).

3.2 RQ2: What AI Techniques Can Be Applied for Improving Testing Activities Identified during Answering the RQ1?

During the SLR process, several promising AI techniques were identified. The findings are presented in Table 4, where the identified techniques are introduced together with the publications, mentioning and using the technique. Figure 3 summarizes all identified artificial intelligence techniques and the publications where usage of those techniques was reported. The mentioned artificial intelligence techniques are ordered according to the count of publications where these techniques were used. For the sake of readability of the diagram, a few algorithms and methods from Table 4 were aggregated. Specifically, those belonging to the computer vision (CV) field were combined into the group labelled as “CV techniques” by grouping namely: non-maximum suppression method (NMS), Scale Invariant Feature Transform (SIFT), Features from Accelerated Segment Test (FAST), Fast Normalized Cross Correlation (FNCC) algorithms, contour detection, Scale-Invariant Feature Transform (SIFT), Optical Character Recognition (OCR).

The visible trend from Figure 3 is that more than half of the reported AI techniques was reported only by a single publication. However, if the individual techniques were grouped by AI subfields, it would be apparent that most used techniques are from the machine learning and computer vision fields. From the machine learning field, the most popular techniques were different types of networks (e.g. Artificial Neural Network, Convolutional Neural Networks, Recurrent neural network, Bayesian Network) and Q-learning (Mariani et al., 2012), which were used in 23% and 8% of the publications

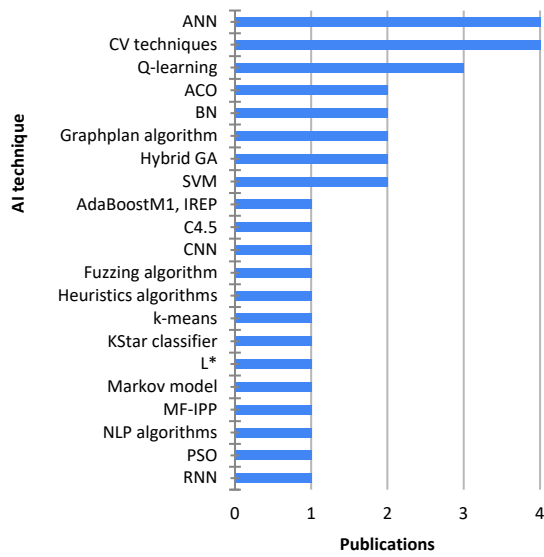


Figure 3: Count of publications by AI techniques.

respectively. The application of computer vision techniques was reported by 11% of the publications in the final set, where the individual techniques from

these field include template matching algorithms, contour detection and OCR.

Table 4 maps the identified AI techniques to testing activities that were identified in RQ1. Unfortunately, it was not possible to identify the exact AI technique used in the Thummalapenta et al. (2012) publication. Therefore, in Table 4 it is mentioned as “Algorithm from NLP field”.

3.3 RQ3: What Are the Reported Benefits of AI Techniques Usage in Software Test Automation?

This section describes the benefits of AI techniques usage in the field of software test automation. The identified benefits may be perceived as a main motivation for applying artificial intelligence in software test automation. To answer the research question, it was important to differentiate the research contributions of the selected publications to software engineering field in general from the practical value AI techniques can bring to the testing activities. The reported benefits were grouped into larger categories

Table 4: Mapping AI techniques and testing activities (x = technique is applicable).

AI technique \ Testing activity	Publications identifier	Test case generation	Test oracle generation	Test execution	Test data generation	Test results reporting	Test repair	Test case selection	Flaky test prediction	Test order generation
Non-maximum suppression method (NMS)	R32					x				
SIFT, FAST, and FNCC algorithms	R24, R37						x			
Contour detection, OCR	R37						x			
Bayesian Network	R7, R28	x						x	x	
Particle swarm optimization (PSO)	R18							x		
Hybrid genetic algorithms	R14, R16	x			x					
Ant colony optimization (ACO)	R2, R8	x			x					
Artificial Neural Network (ANN)	R3, R11, R22, R23	x	x							
Graphplan algorithm	R9, R26	x	x							
Support vector machine (SVM)	R36, R38	x	x							
AdaBoostM1 and Incremental Reduced Error Pruning (IREP) algorithms	R33		x							
Convolutional Neural Networks (CNN)	R29		x		x					
Template-matching algorithm	R32, R35		x							
Decision tree algorithm (C4.5)	R4	x								
Markov model	R31	x								
MF-IPP (Multiple Fact Files Interference Progression Planner)	R15	x								
Algorithm from NLP field	R25	x								
Q-learning	R12, R17, R30	x								
Recurrent neural network (RNN)	R13				x					
L*	R39			x	x					
Fuzzing algorithm	R20			x						
k-means	R21			x						
KStar classifier	R19			x						
Heuristics algorithms	R27									x

Table 5: Reported benefits of using AI techniques.

Benefit	Benefit description	Publications identifier
Manual effort reduction	Some of the testing activities are already automated but still require user guidance, some of them are only semi-automatic, with more human interventions. Application of AI techniques saves the manual effort in terms of reduction of time and cost required for the creation of the tests, execution and their maintenance.	R7 R9, R20, R24, R25, R27, R28, R30, R32, R33, R35, R37, R38, R39
Improved code coverage	Reported benefits regarding the coverage can be described as an ability to cover either a significant part of the statements, branches, transitions or to fully cover them. In the publications, the improvement in coverage was measured against the already existing approaches.	R2, R8, R12, R22, R29, R31, R36, R39
Improved fault and vulnerability detection effectiveness	Generally speaking, generated test cases or oracles are more efficient and effective at identifying flaws in software in comparison with the existing approaches.	R4, R9, R12, R22, R26, R32, R36, R38
Reusability of created test cases and test oracles	Reusability of generated test cases and oracles in the context of the publication could be perceived as an independence of one or more conditions: specific GUI library, application, operating system, source code, system model.	R23, R29, R30, R35, R38
Test breakage repair	Papers were reporting the effectiveness of breakage repair capabilities by correcting the significant amount of breakages, outperforming existing solutions.	R24, R25, R37
Avoidance of redundant actions during the test execution	To improve the execution time and accuracy of the test, the AI techniques contribute to avoidance of system under test restarts and of unnecessary activity transitions.	R8, R20, R39
Improvement of existing solutions	Some of them are improvements that AI-based approach can bring to existing test case generation, selection and data generation techniques: generated text inputs are depending on the context of the system under test and not generated randomly; avoidance of the combinatorial explosion during the generation; selection of test cases based not on one but multiple objectives.	R13, R15, R18
Improved test adequacy of the generated test cases	Generated test cases are able to achieve the required test adequacy, which exceeds equivalent test adequacy for other non-AI approaches. The adequacy is based on the states covered, practicability and non-redundancy of the generated test cases.	R4, R17

and are shown in Table 5 together with their short description.

It is pertinent to note that the relationship between the AI techniques and the benefits can be described as many-to-many relationship type. Taking this fact into consideration, in more than 70% of all papers at least one of the first tree benefits from the table below (Table 5) was observed.

3.4 RQ4: What AI-enhanced Software Tools Can Be Pragmatically Used by Practitioners for Software Test Automation Activities?

During the review, 14 software tools presented in Table 6 were discovered. As it is obvious from the table, only half of them is publicly accessible on the web. Information regarding the year of relevant research publication, in which the tool was used, and

the year of last tool update are presented in Table 6 as well. The latter information is presented only for those tools that are publicly accessible.

The table also shows that only a few tools (3) have been under active development after the papers, which reported on the tools, were published. These are as follows: AutoBlackTest, Sikuli Test and SwiftHand.

In total, seven publicly accessible tools were identified and are shown in Table 6. A detailed description for each of them is presented below.

Model-Inference Driven Testing (MINTest) can be used for the test case generation using the C4.5 algorithm (Papadopoulos & Walkinshaw, 2015). At its webpage ("MIN Test Framework", 2012), it is described as a test framework for unit and integration levels of testing. Its implementation is intended for Linux operating system (OS).

Table 6: AI-enhanced software tools, their public accessibility with years of publication and tool update.

Tool name	Publication identifier	Is tool publicly available?	Publication year	Last update year
MINTest - Model-Inference driven Testing	R4	yes	2015	2012
AutoBlackTest - Automatic Black-Box Testing	R12	yes	2012	2016
DAS - Dynamic Ant Simulator	R17	no	2005	-
ACAT - Activities Classification for Application Testing	R19	no	2018	-
AimDroid	R20	yes	2017	2017
Vista	R24	yes	2018	2018
ATA - Automating Test Automation	R25	no	2012	-
MIDAS	R28	no	2016	-
UI X-RAY	R32	no	2017	-
Sikuli Test	R35	yes	2010	2019
Testilizer	R36	yes	2014	2014
METER - Mobile Test Repair	R37	no	2019	-
AppFlow	R38	no	2018	-
SwiftHand	R39	yes	2013	2015

Automatic Black-Box Testing (AutoBlackTest) implements the reinforcement learning, namely Q-Learning. The tool serves for the automatic graphical user interface (GUI) test case generation (Mariani et al., 2012). According to the AutoBlackTest GitHub repository (Shekhar, Murphy-Hill, & Oliviero, 2016), it runs only with IBM Rational Functional Tester on Windows OS. Based on available information, it is not possible to say whether the tool is usable on a higher version of Windows OS than 8.1 and JRE above 1.6.

AimDroid is a tool that was designed for the GUI testing of Android applications. The automated testing of the application is made by the exploration of its activities. The tool handles test execution and test results reporting back to the user. As an AI-enhancement, the fuzzing algorithm was used (Gu et al., 2017). One of the limitations and a concern regarding the usage of AimDroid is the fact that the device should be rooted: the user of the device is granted root privileges (Institute of Computer Software of Nanjing University, 2017).

Vista leverages the computer vision techniques for the purpose of GUI test breakage repair. It records a successfully running test in its web-based GUI. Once the test starts to fail on a later version of the application, Vista is able to compare the current state of the application with the recorded one and can perform the repair of the test scripts (Stocco et al., 2018). The tool currently supports the repair of the scripts written in Java, in particular Selenium scripts (Stocco, 2018).

Sikuli Test is an automated tool that enables usage of visual notation (e.g. by using a picture of an element for the sake of element identification on the screen)

while writing the GUI test. The tool uses computer vision in order to make automated testing easier for the user. Sikuli Test is designed to be platform independent. So, it can be used for testing of not only desktop applications, but also web and mobile (Android) applications (Chang et al., 2010). It seems that the tool is currently under active development as SikuliX (Hocke, n.d).

Testilizer is capable of generating test cases from existing Selenium test scripts for web applications using SVM (Vapnik, 2013). The Selenium tests are the starting point for the tool, which is able to generate new test cases for the previously not reached states of the application (Fard et al., 2014). Crawljax is required as a prerequisite installed on the machine, where the tests should run (Fard & Mesbah, 2014).

SwiftHand supports the GUI test automation of Android applications. It uses L* algorithm (Irfan, Oriat, & Groz, 2010) for exploration of the model of the application-under-test's GUI. Subsequently, SwiftHand uses it to generate needed inputs in order to examine the previously not visited states of the application (Choi et al., 2013). SwiftHand can be run on Linux OS or OSX (Choi, 2015). The GitHub repository (Choi, 2015) of the tool provides a detailed information on how to install and run the tool.

4 DISCUSSION AND CONCLUSION

The overall goal of this paper was to raise awareness regarding the potential benefits that AI could bring into the software test automation field. A Systematic

Literature Review (SLR) study was conducted for the purpose of fulfilling the goal. As the main outcome of the SLR process, 34 resulting publications were closely analysed and found relevant to the stated research questions.

Based on the discovered publications, nine software testing activities were identified as activities which could be improved by the application of AI techniques. The testing activities are as follows: test case generation, test oracle generation, test execution, test data generation, test results reporting, test repair, test case selection, flaky test prediction, test order generation. The analysed papers addressed mostly test case generation.

According to the collected data, most commonly used AI techniques appears to be from the field of machine learning, specifically different types of neural networks: Artificial Neural Network, Recurrent Neural Network, Bayesian Network; Q-learning; L* etc. Bayesian Network and techniques from the Computer Vision field belong among the techniques that were used across more testing activities more frequently than others.

Eight benefits of AI usage in software test automation were discovered during the SLR. With respect to the fact that the artificial intelligence techniques described in the publications can contribute to multiple benefits, 73% of all papers reported at least one of the following benefits: manual effort reduction, improved coverage, improved fault and vulnerability detection.

In order to provide test practitioners with practical examples of AI application in the test automation field, several AI-enhanced tools were identified. From these tools, only the publicly accessible ones were described here in more details: MINTest, AutoBlackTest, AimDroid, Vista, Sikuli Test, Testilizer and SwiftHand. Importantly, some of the tools mentioned above appear to be already outdated. Therefore, pragmatically speaking, they may not be practically usable as the software engineering field and artificial intelligence techniques evolve quickly. As two examples, we mention MINTest and Testilizer, which have not been updated for seven and five years respectively.

During the review process, an observation was made that most of the publications included in our SLR were concentrated on solving one or two particular problems that can arise during software test automation activities. That means, an integrative and simply-to-use toolset for test automation practitioners starving for AI-driven test automation does not seem to be readily available yet. However, one should note that this review was concentrated on predominantly

academic sources. Therefore, it did not include much grey-literature, which may be perceived as its main limitations. In fact, probing into commercial AI-powered tools by means of multivocal literature reviewing (Garousi, Felderer, & Mäntylä, 2016) might bring additional insights.

Furthermore, as the paper focused mostly on the benefits of AI usage in software test automation, future work should also consider limitations and risks that AI might bring into this context. As an example, it is reasonable to expect that high initial investments into AI technologies, together with a need for special training, may significantly hinder the AI adoption process in the software industry. To cope with these dilemmas, future empirical work should ideally take a practice-based view. Notably, mapping specific motives and approaches driving the deployment and usage of AI-powered test automation tools in real-world companies appears to be an ideal way forward.

ACKNOWLEDGEMENT

This work has been supported by an internal grant funding scheme (F4/23/2019) administered by the University of Economics, Prague.

REFERENCES

- Bhattacharyya, A., & Amza, C. (2018). PReT: A tool for automatic phase-based regression testing. *CloudCom, 2018*, 284–289.
- Bozic, J., & Wotawa, F. (2018). Planning-based security testing of web applications. *AST@ICSE 2018*, 20–26.
- Braga, R., Neto, P. S., Rabêlo, R., Santiago, J., & Souza, M. (2018). A machine learning approach to generate test oracles. *SBES 2018*, 142–151.
- Carino, S., & Andrews, J. H. (2015). Dynamically Testing GUIs Using Ant Colony Optimization. *ASE 2015*, 138–148.
- Chang, T.-H., Yeh, T., & Miller, R. C. (2010). GUI testing using computer vision. *CHI 2010*, 1535.
- Chen, C.-F., Pistoia, M., Shi, C., Girolami, P., Ligman, J. W., & Wang, Y. (2017). UI X-Ray. *IUI 2017*, 245–255.
- Choi, W., Necula, G., & Sen, K. (2013). Guided GUI testing of android apps with minimal restart and approximate learning. *OOPSLA 2013*, 623–640.
- Choi, W. 2015. *wtchoi/SwiftHand*. Retrieved October 27, 2019, from <https://github.com/wtchoi/SwiftHand>
- Dustin, E., Rashka, J., & Paul, J. (1999). Automated Software Testing: Introduction, Management, and Performance. *Addison-Wesley Professional*.
- Fard, A. M., Mirzaaghaei, M., Mesbah, A. (2014a). Leveraging existing tests in automated test generation for web applications. *ASE 2014*, 67–78

- Fard, A. M., Mesbah, A. (2014b). *saltlab/Testlizer*. Retrieved October 27, 2019 from <https://github.com/saltlab/Testlizer>
- Garousi, V., Felderer, M., & Mäntylä, M. V. (2016). The need for multivocal literature reviews in software engineering: complementing systematic literature reviews with grey literature. *EASE 2016: 26:1 – 26:6*
- Garousi, V., & Elberzhager, F. (2017). Test Automation: Not Just for Test Execution. *IEEE Software 34(2)*, 90–96.
- Gu, T., Cao, C., Liu, T., Sun, C., Deng, J., Ma, X., & Lü, J. (2017). AIMDROID: Activity-insulated multi-level automated testing for android applications. *ICSME 2017*, 103–114.
- Hewett, R., & Kijisanayothin, P. (2009). Automated test order generation for software component integration testing. *ASE 2009*, 211–220.
- Hillah, L. M., Maesano, A.-P., Maesano, L., De Rosa, F., Kordon, F., & Wuillemin, P.-H. (2016). Service functional testing automation with intelligent scheduling and planning. *SAC 2016*, 1605–1610.
- Hocke, R., n.d. *SikuliX by RaiMan*. Retrieved October 27, 2019, from <http://sikulix.com/>
- Hourani, H., Hammad, A., & Lafī, M. (2019). The Impact of Artificial Intelligence on Software Testing. *JEEIT 2019*, 565–570.
- Hu, G., Zhu, L., & Yang, J. (2018). AppFlow: using machine learning to synthesize robust, reusable UI tests. *ESEC/SIGSOFT FSE 2018*, 269–282.
- Institute of Computer Software of Nanjing University. (2017). *AimDroid: Activity-Insulated Multi-level Automated Testing for Android Applications*. Retrieved October 27, 2019, from <https://icsnju.github.io/AimDroid-ICSME-2017/>
- Irfan, M. N., Oriat, C., & Groz, R. (2010). Angluin style finite state machine inference with nonoptimal counterexamples. *MIIT 2010*, 11–19.
- Jin, H., Wang, Y., Chen, N., Gou, Z., & Wang, S. (2008). Artificial Neural Network for Automatic Test Oracles Generation. *CSSE (X) 2008*, 727–730.
- King, T. M., Santiago, D., Phillips, J., & Clarke, P. J. (2018). Towards a Bayesian Network Model for Predicting Flaky Automated Tests. *QRS Companion 2018*, 100–107.
- Kitchenham, B., & Charters, S. (2007). Guidelines for performing Systematic Literature Reviews in Software Engineering. *EBSE-2007-01*.
- Last, M., Kandel, A., Bunke, H. (2004). *Artificial Intelligence Methods in Software Testing Series in Machine Perception and Artificial Intelligence, Volume 56, 2004*. World Scientific Publishing Co.
- Li, H., & Lam, C. P. (2005). An ant colony optimization approach to test sequence generation for state-based software testing. *QSIC 2005*, 255–262.
- Li, L., Wang, D., Shen, X., & Yang, M. (2009). A method for combinatorial explosion avoidance of AI planner and the application on test case generation. *CiSE 2009*, 1–4.
- Li, X., Wang, T., Wang, F., & Wang, M. (2011). A novel model for automatic test data generation based on predicate slice. *AIMSEC 2011*, 1803–1805.
- Liu, P., Zhang, X., Pistoia, M., Zheng, Y., Marques, M., & Zeng, L. (2017). Automatic Text Input Generation for Mobile Testing. *ICSE 2017*, 643–653.
- Lu, Y., Yan, D., Nie, S., & Wang, C. (2008). Development of an Improved GUI Automation Test System Based on Event-Flow Graph. *CASE (2) 2008*, 712–715.
- Méndez-Porras, A., Nieto Hidalgo, M., García-Chamizo, J. M., Jenkins, M., & Porras, A. M. (2015). A top-down design approach for an automated testing framework. *UCAml 2015*, 37–49.
- Moghadam, M. H. (2019). Machine Learning-assisted Performance Testing. *ESEC/SIGSOFT FSE 2019*, 1187–1189.
- MIN Test Framework. (2012). MIN Test Framework. Retrieved October 27, 2019, from <http://min.sourceforge.net/>.
- Pan, M., Xu, T., Pei, Y., Li, Z., Zhang, T., & Li, X. (2019). GUI-guided Repair of Mobile Test Scripts. *ICSE (Companion Volume) 2019*, 326–327.
- Papadopoulos, P., & Walkinshaw, N. (2015). Black-box test generation from inferred models. *RAISE@ICSE 2015*, 19–24.
- Paradkar, A. M., Sinha, A., Williams, C., Johnson, R. D., Outturson, S., Shriver, C., & Liang, C. (2007). Automated functional conformance test generation for semantic web services. *ICWS 2007*, 110–117.
- Rafi, D. M., Moses, K. R. K., Petersen, K., & Mäntylä, M. V. (2012). Benefits and limitations of automated software testing: Systematic literature review and practitioner survey. *AST 2012*, 26–42
- Rosenfeld, A., Kardashov, O., & Zang, O. (2018). Automation of Android Applications Functional Testing Using Machine Learning Activities Classification. *MOBILESoft@ICSE 2018*, 122–132.
- Sant, J., Souter, A., & Greenwald, L. (2005). An exploration of statistical models for automated test case generation. *ACM SIGSOFT Software Engineering Notes 30(4)*, 1–7.
- Santiago, D., Clarke, P. J., Alt, P., & King, T. M. (2018). Abstract flow learning for web application test generation. *A-TEST@ESEC/SIGSOFT FSE 2018*, 49–55.
- Shahmiri, S. R., Kadir, W. M. N. W., Ibrahim, S., & Hashim, S. Z. M. (2011). An automated framework for software test oracle. *Inf. Softwa. Technol.*, 53(7), 774–788.
- Sharifipour, H., Shakeri, M., & Haghghi, H. (2018). Structural test data generation using a memetic ant colony optimization based on evolution strategies. *Swarm Evol. Comput.* 40, 76–91.
- Shekhar, S., Murphy-Hill, E., & Oliviero, R., 2016. *ICSE-2011-AutoBlackTest*. Retrieved October 27, 2019, from <https://github.com/SoftwareEngineeringToolDemos/ICSE-2011-AutoBlackTest>.
- Shen, X., Wang, Q., Wang, P., & Zhou, B. (2009). Automatic generation of test case based on GATS algorithm. *GrC 2009*, 496–500.

- Souza, L. S. d., Miranda, P. B. C. d., Prudencio, R. B. C., & Barros, F. d. A. (2011). A Multi-objective Particle Swarm Optimization for Test Case Selection Based on Functional Requirements Coverage and Execution Effort. *ICTAI 2011*, 245–252.
- Srivastava, P. R., & Baby, K. (2010). Automated software testing using metaheuristic technique based on an Ant Colony Optimization. *ISED 2010*, 235–240.
- Stocco, A., Yandrapally, R., & Mesbah, A. (2018a). Visual web test repair. *ESEC/SIGSOFT FSE 2018*, 503–514.
- Stocco, A. (2018b). *Saltlab/vista*. Retrieved October 27, 2019, from <https://github.com/saltlab/vista>
- Thummalapenta, S., Singhanian, N., Devaki, P., Sinha, S., Chandra, S., Das, A. K., & Mangipudi, S. (2012). Efficiently scripting change-resilient tests. *SIGSOFT FSE 2012*, 41
- Vapnik, V.N. (2013). *The nature of statistical learning theory*. Springer science & business media.
- Vieira, F. E., Martins, F., Silva, R., Menezes, R., & Braga, M. (2006). Using Genetic Algorithms to Generate Test Plans for Functionality Testing. *ACM Southeast Regional Conference 2006*, 140-145
- Wang, Y., Bai, X., Li, J., & Huang, R. (2007). Ontology-based test case generation for testing web services. *ISADS 2007*, 43–50.
- White, T. D., Fraser, G., & Brown, G. J. (2019). Improving Random GUI Testing with Image-based Widget Detection. *ISSTA 2019*, 307–317.
- Wotawa, F. (2016). On the Automation of Security Testing. *ICSSA 2016*, 11–16.

