Autonomous Software Testing Model

ABSTRACT

Recent work has shown a shifting paradigm from Test Automation to Autonomous Software Testing (AUST). While Test Automation aims to automate the important tasks in a software testing process, Autonomous Software Testing aims to make the process as full control, decision making, and operation by the machine. That has been made possible due to the advances in the area of artificial intelligence (AI) and machine learning (ML). In this paper, we propose a model for assessing the autonomy of software testing approaches. The model, namely Autonomous Software Testing Model (ASTM), defines six levels ranging from fully manual (Level 0) to fully autonomous testing (Level 5). Using this model as a basis, we evaluate the levels of autonomy of notable AI-enabled test automation approaches proposed by the research community and tools introduced by the software industry. We show that these approaches and tools are today at a low level of autonomy, which suggests opportunities for research and practice towards higher levels of autonomy.

1 INTRODUCTION

Software testing plays an important role in ensuring the high quality of a software product. Software testing validates if the software is functioning appropriately and meeting requirements before it is released into production. To improve the quality of software testing, several technologies have been introduced in an automated software testing (AST) process. For example, automatic test generation approaches [59, 61, 65] help improve the process of generating the test cases for the project under testing. Automated test optimization techniques [12, 32] could help developers in optimizing the current test suites with respect to different criteria and strategies.

The success of artificial intelligence (AI) and machine learning (ML) has opened up several promising directions for advances in software testing. AI and ML have been contributing to making a transformative paradigm shift from Test Automation to Autonomous Software Testing (AUST). The idea of AUST is similar in spirit to Vehicle Autonomy set forth by the Society of Automotive Engineers [17]. While testing is a creative activity, parts of it are repetitive and boring by nature — just like driving is. The repetitive daily commute could be a recipe for mistakes, so repetitive activities in testing could be overlooked, and various serious defects could slip from us during our testing. Therefore, a natural question is whether AI/ML can help push software testing activities to become autonomous in a similar manner as autonomous car driving.

To transition to full autonomy in software testing requires a long process that involves the evolution of the hybrid and combination of humans and AI-powered systems that would enhance both human and machine capabilities. Eventually, autonomous software testing tools will be able to take control of test creation, maintenance, and execution processes. For example, AUST will be able to learn from the performance of the previous test suites and make decisions on how the new test cases should be generated and prioritized to achieve the best future performance in revealing software bugs even when conditions and testing environments change.

The area of AI/ML over the years has successfully developed a wide range of general models, methods, and algorithms that are highly efficient and effective for several application domains. Researchers in software testing have adapted and in many cases invented new techniques that leverage the state-of-the-art AI/ML models in producing the important, fundamental components toward a full software testing autonomy. The maturity of both AI/ML and software testing has given birth to the successes of several AUST approaches in different phases of software testing including Test Planning, Test Management, Test Maintenance, Test Generation, Test Selection and Optimization, Test Monitoring, Test Execution, and Test Evaluation and Report.

To advance the field of Autonomous Software Testing (AUST), we propose the Autonomous Software Testing Model, ASTM, that acts as a reference model for different AI-powered AUST approaches and techniques. We are inspired by the Autonomous Vehicle Model [17] that defines several levels of autonomy in automotive.

The Autonomous Software Testing Model ASTM has a basis on the Automation Theory [58] that classifies the interaction between human and machine ranging from a manual process to a fully autonomous one. Specifically, we have instantiated Ensley and Kaber's Automation Model [23] to propose our ASTM model for the domain of autonomous software testing. ASTM has six different levels in which the three lower levels are aimed for the *human-controlled* processes and the three higher levels are for the *machine-controlled* ones. The three human-controlled levels include 0) **Manual Testing** (no automation), 1) **Assisted Test Automation** (machines support human in various actions), and 2) **Partial Test Automation** (machines support human in making decisions in testing activities). The three machine-controlled levels include 3) **Integrated Automated Testing** (machines generate a list of decisions and human may approve the option or select another one), 4) **Intelligent Automated Testing** (machines generate options, select and carry out the option, and human mainly monitors and intervenes if needed), and 5) **Autonomous Testing** (machine has a full control on the testing process with intelligent decisions).

In ASTM, we also identify the Activity Areas pertaining to software testing, including Test Planning, Test Generation, Test Selection and Optimization, Test Execution, Test Maintenance, Test Evaluation and Report, Test Monitoring, and Test Management. For each of these activity areas, ASTM defines the criteria for each of the six aforementioned levels.

We then perform an extensive survey on the state-of-the-art AI-powered autonomous software testing approaches in

both academia and the software testing industry. We group them into different activity areas and use ASTM to assess the autonomy levels for each aspect of an approach. To facilitate the assessment, we introduce a visualization inspired by a spider web to illustrate the different autonomy levels of a software testing approach. From our survey, we also identify the potential activity areas in autonomous software testing that have not been explored in both academia and industry, pointing to the potential research of leveraging AI/ML in AUST. In this paper, we make the following contributions:

1. Autonomous Software Testing Model (ASTM): a reference model for the AI-powered autonomous software testing approaches.

2. Extensive Survey on AI-powered AUST approaches. This is the first extensive survey on the use of AI/ML in autonomous software testing in both academia and industry.

3. Potential Novel AUST research directions. We also identify the potential directions for leveraging AI/ML in software testing.

2 BACKGROUND ON AUTONOMOUS THEORY

In the field of Automation Theory [23, 42, 45, 58], several models have been proposed to classify the interaction between human and machine ranging from a manual process to a fully autonomous one. Various automation theory models have different levels of automation. In general, all the models have two sets of levels. In the first set of levels, the human plays the controlling role, while the machine does that in the second set. If we consider all the tasks in software testing ranging from Test Planning to Test Evaluation and Report, we could define the different levels of automation for each task. One can map the existing approaches into each cell at each level for the aforementioned tasks.

There exists several attempts from the software industry to define autonomous testing models. For example, Gartner proposes six levels of autonomy ranging from Manual Testing (Level 0), Assisted Testing (Level 1), to Intelligent Automated Testing (Level 4) and Autonomous Testing (Level 5) [28]. SmarBear describes six stages of test automation from manual to autonomous testing [48]. Such models are primarily useful for assessing at which levels test automation tools currently are and for providing directions for future tool offerings.

3 AUTONOMOUS SOFTWARE TESTING MODEL

3.1 Important Concepts

This section provides the details of our proposed **Autonomous Software Testing Model** (ASTM). ASTM is based on the following key concepts.

Definition 1. [**Action**] *Action is an atomic step that can be carried out by human or computer in the software testing process.*

Definition 2. [**Activity**] *Activity is a task that consists a number of actions carried out to achieve a specified objective.*

Definition 3. [**Decision**] *A decision is a resolution that is made on one or multiple actions and/or activities in the software testing process. The decision can be made by human or computer.*

Definition 4. [**Activity Area**] *Activity area is a phase in the software testing process. Each activity area consists of several related activities in the phase. For each activity, several decisions and actions will be made and performed by either human or computer.*

Taking *test generation* as an example, it is an important phase in the software testing process. According to our definitions above, *test generation* is an activity area that consists of the concrete *testing activities* including *test-case design*, *test data generation*, and *test script generation*. When designing test cases, the human or the computer performs *actions* that include, for example, *generating test cases for different types of test and environments*, or *creating test assertions and oracles*. In each action, they also make several *decisions* on the priority of software requirements for which test cases are generated, how many tests is created for each requirement, how many test cases are needed for each type of testing and environment, how much detailed a test case is, or whether a test case is made for automated or manual testing.

Definition 5. [**Level of Autonomy**] *The level of autonomy indicates the ability of the computer to make decisions and/or perform actions autonomously without human intervention during the software testing process.*

We define our autonomous software testing model via the levels of autonomy with respect to different activity areas. The levels are defined according to the following principles:

a. The higher level has a higher degree of autonomy.
b. In the three lower levels (Levels 0–2), the human plays the control role, while in the three upper levels (Levels 3–5), the machine plays the control role.
c. The lowest level is manual testing, which has no automated support, while the highest level is autonomous testing, which has no human intervention.
d. At the higher levels, the autonomy is with respect more to decisions than to actions.
e. At the higher levels, the autonomy involves more intelligent decision-making solutions.

Table 1 presents the six levels of the autonomous software testing model. From *Level 0 to Level 1*, the key difference is at the automated tool supports for certain testing actions, but not the decision support. The assistance for test automation can be for different actions in software testing, including providing the facilities in GUI to help a tester perform actions easily, such as changing the execution order of test cases, selecting test cases to run, and creating test cases according to record-and-playback mechanism, etc. These actions in the testing activities are disconnected but assist the human in making the decisions on his/her tasks for the system under testing (SUT).

Table 1: Autonomous Software Testing Model

Level	Title	Description
0	Manual testing	The human performs all activities and actions and makes all decisions concerning testing of the SUT.
1	Assisted Test Automation	The human performs most actions for testing the SUT with the support of the computer, e.g., automated software tools. The computer carries out certain testing actions automatically under the human's control. The human holds a main role and makes all testing decisions.
2	Partial Test Automation	Both the human and the computer perform testing actions and generate possible decision options, but the human still makes most testing decisions. The computer carries out testing actions based on the decisions made by the human.
3	Integrated Automated Testing	The computer generates a list of decision options, selects an option, and performs actions based on this option if the human approves. The human can also select another decision option for the computer to carry out.
4	Intelligent Automated Testing	The computer generates decision options, determines the best option, and performs testing actions according to this option. The human can still intervene if necessary.
5	Autonomous Testing	The computer has full control of the testing process for the SUT, including making decisions and carrying out all testing actions. The human cannot intervene.

Compared with Level 1, the computer at Level 2 starts to shift from the action aspect to the decision aspect. The computer can provide the analytic information about the test case, the code changes, the configurations, and the bugginess of the current code, to help human make the decision.

From *Level 2 to Level 3*, the key advance of Level 3 is the shared control in decision-making between two sides. The computer generates a list of decision options such as which environments to test, which types of test to perform, and how much to test, however, human may approve the option or select another one.

Compared with Level 3, the computer at Level 4 is more intelligent in making the decision with respect to the testing activities at hand. The computer is able to determine the best decision option. Despite its high intelligence, the control can still be overridden by the human if needed.

When reaching Level 5, the computer can perform testing completely automatically without human supervision.

At a high level of autonomy, AI approaches are applied to perform autonomous and intelligent actions and decisions, for example, about test evaluation and report, such as building test oracles, defect reports, and allocation.

We will present the level descriptions of the activity areas and the examples in the next section.

3.2 Activity Areas

The software testing process consists of a wide range of activities, from planning and designing tests to execution and evaluation [44] [27]. We define the concept of the activity area, which consists of several activities, as a means to access the level of autonomy for a given testing tool or solution. Each activity is then assessed based on its actions and decisions.

The testing process includes the activity areas shown in Table 2. Each activity can be an approach to solving the overall problem of the activity to which it belongs, or it can be a partial solution to the overall problem. Note that these activities are not always present during testing, depending on the software project under test, only some activities are executed. Furthermore, these activities are subject to change over time. Old activities can be adjusted or removed, new activities can be added to match the development trend of technology.

Let us provide the details of the autonomy level definitions of the activity areas in software testing. Due to space limitations, we present only three activity areas, including *test generation*, *test optimization*, and *test maintenance*. The level descriptions of those three activity areas are presented in Table 3.

In those three activity areas, we choose the Test Optimization area to further clarify its three activities, including *test case prioritization, test case selection, and test suite minimization*. In Table 4, we present the level descriptions of the three activities included in the Test Optimization area. The level descriptions of these activities form the basis for evaluating academic approaches and industry tools, which will be presented in Section 4.

4 SURVEYS

4.1 Academic approaches

An study has been done to show the usefulness of using ASTM in identifying the level of autonomy in the current research literature. We selected between 10 and 15 publications on a whole range of autonomy spectrum levels for each activity. The selected works are classified into different levels as the representative candidates based on the definition of ASTM in Section 3 to give the readers a sense of how different automation levels are.

Several vital criteria are considered when selecting publications for evaluations, including recent state-of-the-art methodologies, number of citations, previously mentioned times in other related papers, and publication date (in the descending order of prioritization).

The evaluation results are summarized in Table 5. We will further analyze and discuss them in detail in the following sections.

Table 2: Activity areas in software testing defined for the Autonomous Software Testing Model

Activity areas	Description
Test Management	Test management is concerned with planning, control, and completion of test activities [ISO/IEC/IEEE 29119]. Test planning includes defining the scope of testing, test environments, and strategies; estimating the effort required to perform testing and resources (people, hardware, software, and tools) needed; assigning the resources to the tasks and scheduling; and assessing and managing risks.
Test Generation	Test generation is the phase focusing on test-case design, test data generation, and test script generation. Test suites, which consist of test cases in a certain order to be executed, are also created in this activity area.
Test Optimization	Test optimization focuses on making testing activities more effective and efficient given certain situations, environments, and resources. This activity area includes activities such as test suite minimization (reducing execution time, resources, etc.), test case selection (choosing relevant test cases, data, and scripts based on certain criteria), and test case prioritization (ranking test cases based on certain criteria).
Test Maintenance	Test maintenance is the activity area concerned with repairing tests (test cases, data, and scripts) so that they can still be valid when the SUT is changed. Activities include detecting the fragile components, fixing broken tests, and making tests more resilient to changes.
Test Orchestration	This test area is concerned with determining which tests to be performed on which test environments using which resources, defining and carrying out sequences of automated test activities. At a high level of autonomy, test orchestration involves the synchronization of making decisions and performing actions across activity areas.
Test Monitoring	Test monitoring is concerned with collecting, measuring, and analyzing data, evaluating testing results against pre-specified objectives, and providing feedback on the current progress of software testing activities.
Test Execution	Test execution involves running the SUT using the test cases and recording the results. In manual testing, testers exercise the test cases on the SUT, observe, and record its results. In automated testing, test scripts along with test data are executed automatically.
Test Evaluation and Report	This activity area includes main activities such as evaluating, analyzing, and reporting test results. These activities involve evaluating test verdicts, analyzing the root cause of failures and areas of vulnerabilities, visualizing test results and analytics, and reporting test verdicts and defects.

Our observation also revealed that (1) there is a significant dependency on the number of publications of each activity area. For example, topics about test generation, self-healing, test prioritization, and test execution usually gain lots of attention in the research community, while other activity areas such as planning, orchestration, and monitoring are still in the early stage of the research. (2) the autonomous levels of proposed methods are mostly between Level 2 and Level 3 in our ATSM model, except these research works about planning, orchestration, and management usually achieve levels 0 to 1 due to high complexity and involve lots of human intervention, (3) a large number of scientific works are also proposed to make the interaction between humans and machines efficient and productive, ranging from test monitoring to test evaluation, and test reporting, which also shows promising directions for making essential building blocks toward building a fully functional autonomous testing system.

4.2 Industrial tools

To demonstrate the practical usability of the ASTM in evaluating industrial tools, we select representative automated testing tools based on their popularity and user benchmark. Our insight suggests that almost all testing tools nowadays support at least Level 0 - manual testing for main testing activity areas as explained in Section 3; therefore, in order to make a clear and concise comparison, tools that have all main activities belong to Level 1 or above are remained (excepts Test Planning, which the human usually takes main responsibility), and the rest of tools are filtered out. Finally, we come up with a list of ten representative testing tools covered by multiple aspects.

Since each main activity area may contain several activities and a tool usually targets a few key activities, we assess each activity's autonomy level based on the maximum performance level of all activities it has. It is worth mentioning that, within an activity, the autonomy level of each activity may be dissimilar; for example: in the context of test generation, a tool may have a good capacity for generating test scripts while performing poorly on generating test data, or for test maintenance, a tool may have an excellent ability on identifying UI changes, but at the same time lack of power for repairing test cases. Using the maximum level of activities for the assertion may lead to activities dominating an entire

Table 3: Level descriptions of three activity areas with respect to the Autonomous Software Testing Model

Level	Test Generation	Test Optimization	Test Maintenance
0 - Manually Testing	The human generates test fully manually.	The human performs test optimization fully manually.	The human maintains test cases fully manually.
1 - Assisted Test Automation	The computer can assist the human in performing Actionss by providing facilities, such as user interfaces and functionalities, for test generation.	The computer can provide GUIs to help users perform Actionss related to test optimization more easily.	The computer provides facilities for the human to maintain tests more easily.
2 - Partial Test Automation	The computer provides human information related to test cases and source code, information about the parts of code that are modified, added, or removed to help people make Decisions to generate tests appropriately. The human relies on that information to decide which area they need to generate tests. Implementation is done by the computer.	The computer assists the human in performing optimization by providing analytic test information such as coverage information, execution history, test case length, etc. The human relies on that information and their objectives to design the strategy to optimize tests. The computer performs test optimization based on a human-chosen strategy.	The computer finds the difference between an old and updated version and shows the analytic information for the human. The human relies on that information to determine which and how tests should be maintained.
3 - Integrated Automated Testing	The computer generates the list of generated strategies for the human to select. The human decides the strategy to generate tests. The computer will automatically generate tests according to the strategy chosen by the human. Manual tests could be added to the test suite.	The computer generates a list of optimization strategies for the human to select. The human decides which strategy should be used. After the computer performs the Actions belonging to the selected strategy, the human can make some adjustments to the test suite.	The computer will analyze and recommend the list of tests that need to be maintained. The human decide which tests should be maintained from the generated list or can choose others. The computer follows the human Decisions to perform the maintenance of the corresponding tests.
4 - Intelligent Automated Testing	The computer automatically generates tests based on the strategy that is the best for SUT. The computer needs human validation to improve recommendations. The computer has the capability to learn to improve performance through human validation.	The computer has the ability to analyze and choose the best strategy for optimizing test cases. Human validation is still needed. The computer has the capability to learn to improve performance through human validation.	The computer can determine which test should be maintained and automatically implement the maintenance. The human can add some adjustments to tests that The computer missed. The computer has the capability to learn to improve performance through human validation.
5 - Autonomous Testing	The computer generates and optimizes tests fully automatically, which is the highest benefit for the SUT.	The computer optimizes the tests fully automatically, the tests are optimized in the way that is most beneficial to the SUT.	The computer can identify all the tests that need to be maintained and fully automatically implement reparation.

particular testing activity, and evaluation relies solely on it, resulting in over-classification when evaluating a specific tool/application. However, for the purpose of illustrating the current landscape of AI-powered autonomous software testing tools, it has no significant effect.

A brief summarization and level classification for each selected tool are described below and in Table 6, respectively.

(1) **Functionize** [25] is a cloud-based automated testing tool. It achieves Level 2 for test generation (as it employs plain English and NLP to quickly generate test cases), Level 2 for test maintenance (it can detect UI changes and test failures seamlessly).

(2) **TestCraft** [11] is an AI-powered test automation platform built on top of Selenium for regression, continuous testing web application monitoring. It achieves Level 2 for test maintenance (ML is used to identify web elements correctly even when a web application change happens), Level 2 for test generation (its on-the-fly mode enables the creation of test models out of the test scenario, making it easier to reuse test steps).

(3) **Applitools** [1] is a visual UI testing and monitoring software. It achieves Level 1 for test maintenance (its comparison algorithm can recognize whether the changes

Table 4: Level descriptions of three activities in Test optimization

Level	Test Case Prioritization	Test Case Selection	Test Suite Minimization
0 - Manually Testing	The human fully decides the order test case.	The human selects test cases manually.	The human performs test optimization fully manually.
1 - Assisted Test Automation	**Decisions**: None **Actions**: The computer provides functionality that allows the user to change the execution order of the test cases contained in the test suite.	**Decisions:** None **Actions**: The computer provides facilities for the human to select tests. The human can select some test cases included in test suite, and ask the computer to execute only those test cases.	**Decisions:** None **Actions:** The computer provides a interface to remove or modify the test cases contained in the test suite easily.
2 - Partial Test Automation	**Decisions:** The computer supports people to make Decisions by providing the analytic information of test cases(test case coverage, execution history of test cases, the change of codes,...). This information will be the reference base for humans to decide the order of test cases. **Actions:** Similar to the Actions of Level 1	**Decisions:** The computer assists the human in making Decisions by supplying test case analytic information (test case coverage, execution history of test cases, the change of codes, etc.). The human bases on that information and their goal to decide which test cases should be selected. **Actions:** Similar to the Actions of Level 1	**Decisions:** The computer supports people in making Decisions by providing the analytic information of test cases (test case coverage, execution history of test cases, the change of codes, etc.). The human relies on that information and their objective to minimize the test suite. **Actions:** Similar to the Actions of Level 1
3 - Integrated Automated Testing	**Decisions:** The computer generates a list of prioritization strategies and recommends a strategy for implementation. Humans can agree to the machine's suggestions or choose a different strategy. **Actions:** The computer prioritizes test cases according to the strategy that the human has chosen. The human can make some adjustments to the order of test cases through the facility that the computer provides.	**Decisions:** The computer generates a list of selection strategies and recommends a strategy for implementation. Humans can agree to the machine's suggestions or choose a different strategy. **Actions:** The computer, based on the strategy, will choose suitable test cases and provide facilities for the human to add or remove some test cases to be suitable for their goals.	**Decisions:**The computer generates the list of minimization strategies and recommends a strategy for implementation. Humans can agree to the machine's suggestions or choose a different strategy. **Actions:** The computer is based on the chosen strategy to minimize the test suite. The human can modify the test suite through the facility that the computer provided.
4 - Intelligent Automated Testing	**Decisions:** The computer can generate a list of strategies and decide which strategy is used to prioritize test cases. **Actions:** The computer ranks test cases according to the strategy that it has chosen. Human validation is still needed. The computer has the capability to learn to improve performance through human validation.	**Decisions:** The computer can generate a list of strategies and decide which strategy is used to select test cases. **Actions:** The computer selects test cases according to the strategy that it has chosen. Human validation is still needed. The computer has the capability to learn to improve performance through human validation.	**Decisions:** The computer can generate a list of strategies and decide which strategy is used to minimize tests. **Actions:** The computer minimizes tests according to the strategy that it has chosen. Human validation is still needed. The computer has the capability to learn to improve performance through human validation.
5 - Autonomous Testing	The computer ranks the test cases fully automatically, the test cases are arranged in the order that is most beneficial to the SUT.	The computer selects the test cases fully automatically. The selected test suite contains test cases that are in the best interest of SUT.	The computer performs the minimization fully automatically and brings efficiency to SUT.

Table 5: A classification of autonomous levels for each activity in the current scientific approaches

Testing Activities	Autonomous Levels					
	Level 0	Level 1	Level 2	Level 3	Level 4	Level 5
Test Management		[15]		[64]		
Test Orchestration			[26]	[43],[21],[5]		
Test Maintenance			[19],[18]	[50],[16],[20],[30],[38]		
Test Generation			[46], [56],[13]	[62],[9],[2],[39],[63],[3],[31]		
Test Optimization				[49],[8],[22],[7],[41]		
Test Monitoring		[37],[14],[47],[29]				
Test Execution				[35][4][40][60][51][36]		
Test Evaluation & Report			[66][33][10]			

are meaningful or just bugs), and Level 2 for test monitoring (it can find visual bugs in apps and make sure that no visual elements are overlapping and invisible).

(4) **Testim** [55] is an end-to-end AI testing tool that focuses on functional testing and UI testing. It achieves Level 2 for test maintenance (it can detect the changes in the app to run automatic tests), Level 2 for test report (this tool integrates seamlessly with CI/CD tools, provides detailed bug reports and performs root-cause analysis of the failed tests for quick remedial action).

(5) **Mabl** [53] is an unified platform that creates and runs automated functional UI tests. It achieves Level 3 for test execution (it can fully manage testing infrastructures in the cloud, scale tests infinitely and run them all in parallel), Level 3 for test maintenance (it can automatically detect whether elements of applications have changed, and dynamically updates the tests to compensate for those changes), Level 4 for test report (it can continuously compare test results to test history to quickly detect changes and regressions, resulting in more stable releases), Level 2 for test monitoring (it can help identify and surface problems quickly, alerting you to possible impacts before they impact customers).

(6) **Tricentis Tosca** [57] powered by Vision AI and model-based test automation to deliver test automation. It achieves Level 3 for test generation (it can automatically recognize and identify user interfaces elements and controls across any form factor the same way humans do to aid in the automated generation of robust test cases), Level 3 for test optimization (its Risk-AI feature can automatically detect most at risk objects and select the right set of tests to minimize business and technical impact of code changes), Level 3 for test maintenance (its self-healing AI feature can automatically adapt test cases as applications evolve with each iteration).

(7) **UiPath Test Suite** [52] is a tightly integrated bundle of tools. It achieves Leve l 3 for test orchestration (it can deploy and manage an entire workforce, handling all the critical enterprise duties: release management, centralized logging, reporting, auditing and monitoring, remote control, scheduling, workload management, and asset management), Level 2 for test execution (it has capabilities on multiple machines, distributed across teams and through job scheduling makes the execution process autonomous and straightforward).

(8) **AutonomIQ** [6] is an AI-driven, autonomous platform. It achieves Level 3 for test generation (it can generate automation scripts automatically in plain English, and ensure compliance with all regulatory requirements and eliminate security risk using AI-generated synthetic data for all automation needs), Level 3 for test maintenance (it can maintain quality throughout the application lifecycle with autonomous discovery, autonomous healing capability, deliver flawless updates, and up-to-date tracking of changes), Level 4 for test execution (it runs multiple tests in parallel, determine test frequency, keep pace with browser updates and executions across operating systems and platforms).

(9) **Test.AI** [54] is an automation framework built on top of Selenium and Appium that puts humans in control of bots. It achieves Level 3 for test generation (it allows human to control a running AI-Bot as it navigates web application, and build tests in real-time), Level 2 for test execution (it can support a massive scale of thousands of VMs, and thousands of applications in a single run), Level 3 for test maintenance (it can adjust and identify UI changes), Level 3 for test report (as it can provide test flow results, full analytics and reports for each test case).

(10) **Katalon Studio** [34] is a testing platform for web, API, mobile, and desktop applications. It achieves Level 2 for test generation (with advanced record playback mechanism), Level 3 for test execution (it can execute tests on all OSs, browsers, and devices, operate on both cloud and on-premise infrastructures, support parallel and sequential executions), Level 3 for test maintenance (automatically locates the web or app elements when the AUT get updated), Level 3 for test orchestration (it can orchestrate tests smartly to improve test quality, maximize test coverage, and save maintenance costs), and Level 1 for reporting (it can generate comprehensive reports and analytics).

Table 6: A classification of autonomous levels for each activity in the current popular industry testing tools

Testing Activities	Autonomous Levels					
	Level 0	Level 1	Level 2	Level 3	Level 4	Level 5
Test Management	Functionize, TestCraft, Applitools, Testim, Mabl, Tricentis Tosca, UiPath Test Suite, AutonomIQ, Test.AI, Katalon Studio					
Test Orchestration		Functionize, TestCraft, Applitools, Testim, Tricentis Tosca	Mabl	Katalon Studio, UiPath Test Suite		
Test Maintainance		Applitools	Functionize, TestCraft, Testim	Tricentis Tosca, Mabl, UiPath Test Suite, AutonomIQ, Test.AI, Katalon Studio		
Test Generation			TestCraft, Applitools, Testim, Mabl, UiPath Test Suite, Functionize, Katalon Studio	Tricentis Tosca, AutonomIQ, Test.AI		
Test Optimization		Functionize, TestCraft, Applitools, Testim, Mabl, UiPath Test Suite, AutonomIQ, Test.AI, Katalon Studio		Tricentis Tosca		
Test Monitoring			Mabl, Applitools	Functionize, TestCraft, Testim, Tricentis Tosca, UiPath Test Suite, AutonomIQ, Katalon Studio, Test.AI		
Test Execution		TestCraft, Functionize, Applitools, Testim, Tricentis Tosca	UiPath Test Suite, Test.AI	Mabl, Katalon Studio, AutonomIQ		
Test Evaluation & Report		TestCraft, Applitools, Tricentis Tosca, Katalon Studio, AutonomIQ, UiPath Test Suite, Functionize	Testim	Mabl, Test.AI		

5 DISCUSSIONS

The overview results in Section 4.1 show many gaps in the current research landscape of software testing. Careful investigations can benefit researchers in finding out the subsequent directions to target concerning the automated software testing process. This section's goal is to outline further the activities that are most commonly studied and gained lots of attention in the research community. There are three key points that we have learned from our evaluation.

- We found that a large portion of research works have been proposed and developed mainly for test script generation, self-healing mechanism, test case prioritization, test execution with CI/CD tools, and AI-driven monitoring. These are well-defined problems, have a long development history, and can be integrated seamlessly into the existing system, with high demand from clients/customers. The reasons mentioned above have created more opportunities for research than the other activities in the same area.
- On the flip side, several activities remain untouched and not yet explored extensively, including test strategy planning, resource planning, automation reporting,

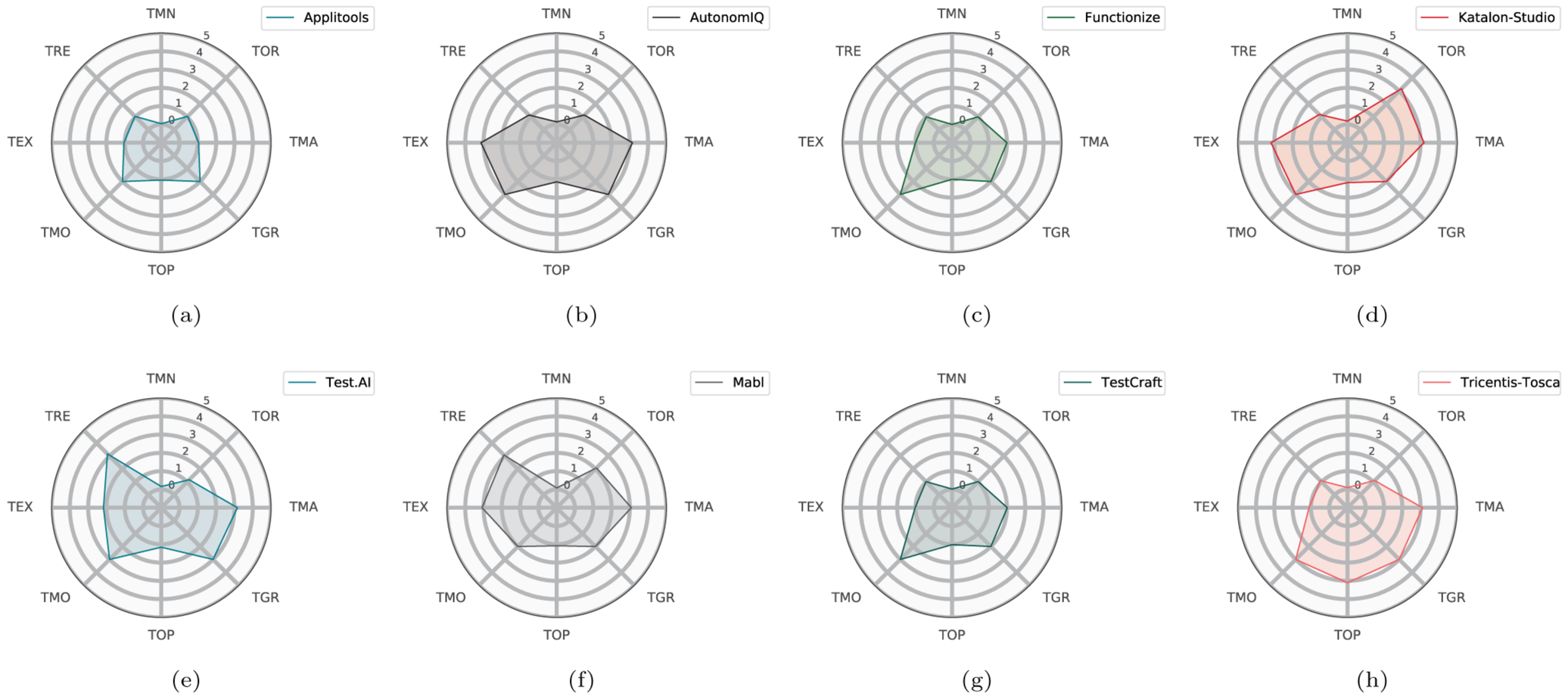

Figure 1: Industrial AI-powered software testing tools across activity areas: Testing Management (TMN), Test Orchestration (TOR), Test Maintenance (TMA), Test Generation (TGR), Test Optimization (TOP), Test Monitoring (TMO), Test Execution (TEX), and Test Evaluation and Report (TRE)

data generation, and orchestration. These activities are considered changing issues because they usually involve lots of human decisions and hardly be replaced by the computer at all. However, if computers can completely handle these activities, it would be a great stride toward building a fully autonomous software testing system.

- Current academic works mainly focus on automated testing rather than autonomous testing. The automated testing targets supporting primitive actions for humans, maximizing intra-activity performance. In contrast, autonomous testing aims to provide more intelligent decisions for humans to select, maximizing both intra- and inter-activities performance. These approaches in the first category mainly try to improve their capacities on a single-specific objective while paying little attention to other activity areas, making them less applicable, even infeasible, when integrating these solutions into the existing systems. Future research should pay more attention to autonomous testing, the overall software testing landscape, and the inter-activities relationship to benefit the testing process.

Section 4.2 also presents another interesting story, that the intelligence features in the current industrial tools are applied widely, although they are still quite primitive. These features usually lie in Levels 2 and 3 of the ASTM model. One can easily notice the similarity between academic approaches and industrial tools regarding which activities are most concerned about. The advancements in software engineering, natural language processing, computer vision, and the ability to extend its functionality by using third-party platforms can enhance the human-like decision-making capability in some primary activities. However, the autonomous levels of the features are still far from perfection (Level 5).

- To the best of our knowledge, a fully autonomous end-to-end testing system (or for particular testing activity) without human guidance and intervention is still not developed completely. Human-in-the-loop is still a safe, reliable option and is frequently adopted in the current testing tools.
- The current industrial tools still play the role of supporting humans in decision-making. Popular automated testing tools like Test.AI, Functionize, and Katalon Studio, usually achieve Level 1 in the ATSM model for primary activities such as test execution, optimization, and evaluation by providing humans with necessary user interfaces and utilities for making the final decision. Other activities may obtain higher autonomy as test generation (Level 2) or test maintenance (Level 3). However, no tools in our survey can completely automate these activities without human guidance.
- Our previous brief review reveals the considerable potential and the need for future research and development in the following testing activities: planning, evaluation, and orchestration. At the moment, tools only achieve Level 0-2 in the ASTM model, which means humans need to rely mainly on their expertise and other management platforms such as Jira, Klaros, or TestCollab to complete these activities, making the testing process disconnected, interrupted, and prone to

error. Bringing up to a higher level of autonomy would benefit the whole testing process in a significant way.

6 RELATED AUTONOMOUS TESTING MODELS

Being inspired by the Autonomous Vehicle Model [17], several autonomous testing models have been proposed for assessing the level of automation and autonomy of software testing tools. Gartner introduced an autonomous model consisting of six levels of autonomy, namely Manual Testing (Level 0), Assisted Testing (Level 1), Partially Automated Testing (Level 2), Integrated Automated Testing (Level 3), Intelligent Automated Testing (Level 4), and Autonomous Testing (Level 5) [28]. However, this model lacks a concrete framework for accessing the level of autonomy of given testing tools. Based on this model, Functionize also defined a five-level autonomous testing model along with an assessment framework [24]. Still, this assessment framework covers only three main activities of test automation, which are test creation, test execution and analysis, and test maintenance.

Similarly, Smartbear presented a model that consists of six stages from manual to autonomous testing and used this model as a reference to assess the level of automation and autonomy of their test automation tool [48].

Unlike these approaches, our model offers a concrete framework for accessing the level of autonomy for a given testing tool and approach. It is based on the core principles and concepts of action, decision, activity, and activity area.

7 CONCLUSION AND LESSONS LEARNED

Software testing has made a great stride in recent years, shifting from automated testing tools to a more autonomy testing framework. Autonomous Software Testing is characterized by full and intelligent control of the computer in operations and decision-making. This paradigm shift has been made possible due to the advances in the area of artificial intelligence (AI) and machine learning (ML).

In this paper, we present the Autonomous Software Testing Model (ASTM) that defines six levels of autonomy ranging from complete manual (Level 0) to full autonomous testing (Level 5). We have demonstrated our ASTM model via a variety of software testing activities. We have conducted an extensive survey on the levels of autonomy of notable AI-enabled test automation approaches proposed by the research community and tools introduced by the software industry.

We have reported the lessons learned from the state-of-the-art automated and autonomous testing approaches and suggested opportunities for research and practice towards higher levels of autonomy. Specifically, we reported

1. A large portion of research works are still focusing much on automated software testing including test orchestration, test maintenance, test generation, and test optimization.

2. Future research should pay more attention to advancing the field toward autonomous software testing, including test monitoring and test evaluation and report.

3. The state-of-the-art software testing tools in the industry are reaching Level 3. The focus of those industrial tools spans a wider range from testing orchestration, test maintenance, test generation, test monitoring, and test execution.

4. An autonomous end-to-end testing system (or for particular testing activity) without human guidance and intervention is still in future works. However, the approaches and tools have started moving toward supporting humans in decision-making instead of only supporting automated testing activities.

To sum up, not all testing activities have the same potential for automation, and we are still far from having a fully autonomous testing system. But understanding the current level of each testing activity, what components it has covered, what components are still missing, and how far from the current level to the perfection level are crucial steps for both practitioners and researchers. We can have a clear vision and detailed roadmap toward building such futuristic systems based on this information. In addition, given that humans may perform better than computers in some activities that require planning, intuition, and judgment, a promising direction for future research and development would be incorporate decision-making between humans and machines to take the best of both worlds.

REFERENCES

[1] Applitools. 2022. Automated Visual Testing. https://applitools.com/. Accessed: 2022-07-05.

[2] Italo L Araújo, Ismayle S Santos, João B Ferreira Filho, Rossana MC Andrade, and Pedro Santos Neto. 2017. Generating test cases and procedures from use cases in dynamic software product lines. In *Proceedings of the Symposium on Applied Computing*. 1296–1301.

[3] Andrea Arcuri, Gordon Fraser, and Juan Pablo Galeotti. 2014. Automated unit test generation for classes with environment dependencies. In *Proceedings of the 29th ACM/IEEE international conference on Automated software engineering*. 79–90.

[4] Cristian Augusto. 2020. Efficient test execution in End to End testing: Resource optimization in End to End testing through a smart resource characterization and orchestration. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Companion Proceedings*. 152–154.

[5] Cristian Augusto, Jesús Morán, Antonia Bertolino, Claudio de la Riva, and Javier Tuya. 2019. RETORCH: Resource-aware end-to-end test orchestration. In *International Conference on the Quality of Information and Communications Technology*. Springer, 297–310.

[6] Autonomiq.io. 2022. A Leading Provider Of Scriptless Test Automation Solutions. https://autonomiq.io/. Accessed: 2022-07-05.

[7] Mojtaba Bagherzadeh, Nafiseh Kahani, and Lionel Briand. 2021. Reinforcement learning for test case prioritization. *IEEE Transactions on Software Engineering* (2021).

[8] Antonia Bertolino, Antonio Guerriero, Breno Miranda, Roberto Pietrantuono, and Stefano Russo. 2020. Learning-to-rank vs ranking-to-learn: Strategies for regression testing in continuous integration. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. 1–12.

[9] Matteo Biagiola, Andrea Stocco, Filippo Ricca, and Paolo Tonella. 2019. Diversity-based web test generation. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 142–153.

[10] Tim Blazytko, Moritz Schlögel, Cornelius Aschermann, Ali Abbasi, Joel Frank, Simon Wörner, and Thorsten Holz. 2020. {AURORA}: Statistical Crash Analysis for Automated Root Cause Explanation. In *29th USENIX Security Symposium (USENIX Security 20)*. 235–252.

[11] Perfecto by Perforce. 2022. Introducing TestCraft for Codeless, Automated Testing | by Perforce. https://www.perfecto.io/blog/introducing-testcraft. Accessed: 2022-07-05.
[12] Gaocheng Cai, Qinghua Su, and Zhongbo Hu. 2021. Automated test case generation for path coverage by using grey prediction evolution algorithm with improved scatter search strategy. *Engineering Applications of Artificial Intelligence* 106 (2021), 104454. https://doi.org/10.1016/j.engappai.2021.104454
[13] José Campos, Andrea Arcuri, Gordon Fraser, and Rui Abreu. 2014. Continuous test generation: enhancing continuous integration with automated test generation. In *Proceedings of the 29th ACM/IEEE international conference on Automated software engineering*. 55–66.
[14] Jeanderson Candido, Maurício Aniche, and Arie van Deursen. 2019. Contemporary software monitoring: A systematic literature review. *arXiv e-prints* (2019), arXiv–1912.
[15] Thomas J Cheatham, Jungsoon P Yoo, and Nancy J Wahl. 1995. Software testing: a machine learning experiment. In *Proceedings of the 1995 ACM 23rd annual conference on Computer science*. 135–141.
[16] Shauvik Roy Choudhary, Dan Zhao, Husayn Versee, and Alessandro Orso. 2011. Water: Web application test repair. In *Proceedings of the First International Workshop on End-to-End Test Script Engineering*. 24–29.
[17] SAE On-Road Automated Vehicle Standards Committee et al. 2018. Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles. *SAE International: Warrendale, PA, USA*.
[18] Brett Daniel, Tihomir Gvero, and Darko Marinov. 2010. On test repair using symbolic execution. In *Proceedings of the 19th international symposium on Software testing and analysis*. 207–218.
[19] Brett Daniel, Vilas Jagannath, Danny Dig, and Darko Marinov. 2009. ReAssert: Suggesting repairs for broken unit tests. In *2009 IEEE/ACM International Conference on Automated Software Engineering*. IEEE, 433–444.
[20] Brett Daniel, Qingzhou Luo, Mehdi Mirzaaghaei, Danny Dig, Darko Marinov, and Mauro Pezzè. 2011. Automated GUI refactoring and test script repair. In *Proceedings of the First International Workshop on End-to-End Test Script Engineering*. 38–41.
[21] Hitham Haidar Deyab and Rodziah Binti Atan. 2015. Orchestration framework for automated Ajax-based web application testing. In *2015 9th Malaysian Software Engineering Conference (MySEC)*. IEEE, 1–6.
[22] Jackson Antonio do Prado Lima and Silvia Regina Vergilio. 2020. A multi-armed bandit approach for test case prioritization in continuous integration environments. *IEEE Transactions on Software Engineering* (2020).
[23] MICA R. ENDSLEY and DAVID B. KABER. 1999. Level of automation effects on performance, situation awareness and workload in a dynamic control task. *Ergonomics* 42, 3, 462–492. https://doi.org/10.1080/001401399185595
[24] Functionize. 2021. *5 Levels of Test Automation*. https://www.functionize.com/resources/5-levels-of-test-automation
[25] Functionize.com. 2022. Agile Automation Testing Framework with Machine Learning | Functionize. https://www.functionize.com/. Accessed: 2022-07-05.
[26] Sushant G Gaikwad and MA Shah. 2015. Pipeline Orchestration for Test Automation using Extended Buildbot Architecture. *International Journal of Computer Applications* 975 (2015), 8887.
[27] Vahid Garousi and Frank Elberzhager. 2017. Test automation: not just for test execution. *IEEE Software* 34, 2 (2017), 90–96.
[28] Gartner. 2020. *Innovation Insight for Autonomous Testing*. https://www.gartner.com/en/documents/3992325/innovation-insight-for-autonomous-testing
[29] Madhwaraj Kango Gopal, M Govindaraj, Paramita Chandra, Prathiksha Shetty, and Sunny Raj. 2022. Bugtrac–A New Improved Bug Tracking System. In *2022 IEEE Delhi Section Conference (DELCON)*. IEEE, 1–7.
[30] Mouna Hammoudi, Gregg Rothermel, and Andrea Stocco. 2016. Waterfall: An incremental approach for repairing record-replay tests of web applications. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. 751–762.
[31] Milad Hanna, Amal Elsayed Aboutabl, and Mostafa-Sami M Mostafa. 2018. Automated software testing framework for web applications. *International Journal of Applied Engineering Research* 13, 11 (2018), 9758–9767.
[32] Han Huang, Fangqing Liu, Zhongming Yang, and Zhifeng Hao. 2018. Automated Test Case Generation Based on Differential Evolution With Relationship Matrix for iFogSim Toolkit. *IEEE Transactions on Industrial Informatics* 14, 11 (2018), 5005–5016. https://doi.org/10.1109/TII.2018.2856881
[33] Brittany Johnson, Yuriy Brun, and Alexandra Meliou. 2020. Causal testing: understanding defects' root causes. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. 87–99.
[34] Katalon. 2022. Simplify Web, API, Mobile, Desktop Automated Tests. https://katalon.com/. Accessed: 2022-07-05.
[35] Jung-Hyun Kwon, In-Young Ko, and Gregg Rothermel. 2018. Prioritizing browser environments for web application test execution. In *Proceedings of the 40th International Conference on Software Engineering*. 468–479.
[36] Mariam Lahami, Moez Krichen, and Roobaea Alroobaea. 2019. TEPaaS: test execution platform as-a-service applied in the context of e-health. *International Journal of Autonomous and Adaptive Communications Systems* 12, 3 (2019), 264–283.
[37] Mykhailo Lasynskyi and Janusz Sosnowski. 2021. Extending the Space of Software Test Monitoring: Practical Experience. *IEEE Access* 9 (2021), 166166–166183.
[38] Stanislav Levin and Amiram Yehudai. 2017. The co-evolution of test maintenance and code maintenance through the lens of fine-grained semantic changes. In *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 35–46.
[39] Leonardo Mariani, Mauro Pezzè, and Daniele Zuddas. 2018. Augusto: Exploiting popular functionalities for the generation of semantic gui tests with oracles. In *Proceedings of the 40th International Conference on Software Engineering*. 280–290.
[40] Shouvick Mondal, Denini Silva, and Marcelo d'Amorim. 2021. Soundy Automated Parallelization of Test Execution. In *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 309–319.
[41] Vu Nguyen and Bach Le. 2021. Rltcp: A reinforcement learning approach to prioritizing automated user interface tests. *Information and Software Technology* 136 (2021), 106574.
[42] Ryan W Proud, Jeremy J Hart, and Richard B Mrozinski. 2003. *Methods for determining the level of autonomy to design into a human spaceflight vehicle: a function specific approach*. Technical Report. National Aeronautics and Space Administration Houston TX Lyndon B Johnson ....
[43] Nikhil Rathod and Anil Surve. 2015. Test orchestration a framework for continuous integration and continuous deployment. In *2015 international conference on pervasive computing (ICPC)*. IEEE, 1–5.
[44] Stuart Reid. 2013. ISO/IEC/IEEE 29119. (2013).
[45] Victor Riley. 1989. A general model of mixed-initiative human-machine systems. In *Proceedings of the Human Factors Society Annual Meeting*, Vol. 33. Sage Publications Sage CA: Los Angeles, CA, 124–128.
[46] Sina Shamshiri, José Miguel Rojas, Juan Pablo Galeotti, Neil Walkinshaw, and Gordon Fraser. 2018. How do automatically generated unit tests influence software maintenance?. In *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 250–261.
[47] Natalia Silvis-Cividjian, Marc Went, Robert Jansma, Viktor Bonev, and Emil Apostolov. 2021. Good Bug Hunting: Inspiring and Motivating Software Testing Novices. In *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1*. 171–177.
[48] Smartbear. 2021. *Six Stages from Manual to Autonomous Testing*. https://smartbear.com/resources/ebooks/six-stages-from-manual-to-autonomous-testing/
[49] Helge Spieker, Arnaud Gotlieb, Dusica Marijan, and Morten Mossige. 2017. Reinforcement learning for automatic test case prioritization and selection in continuous integration. In *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis*. 12–22.
[50] Andrea Stocco, Rahulkrishna Yandrapally, and Ali Mesbah. 2018. Visual web test repair. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 503–514.
[51] Panagiotis Stratis and Ajitha Rajan. 2018. Speeding up test execution with increased cache locality. *Software Testing, Verification*

and Reliability 28, 5 (2018), e1671.

[52] Uipath Test Suite. 2022. UiPath Test Suite, 2022. https://docs.uipath.com/test-suit/. Accessed: 2022-07-05.

[53] Intelligent Test Automation For Agile Teams. 2022. Mabl. https://www.mabl.com/. Accessed: 2022-07-05.

[54] Test.Ai. 2022. test.ai. https://test.ai/. Accessed: 2022-07-05.

[55] Automated Functional Testing Software Testing Tool Testim.io. 2022. AI-driven E2E automation with code-like flexibility for your most resilient tests. https://www.testim.io/. Accessed: 2022-07-05.

[56] Suresh Thummalapenta, K Vasanta Lakshmi, Saurabh Sinha, Nishant Sinha, and Satish Chandra. 2013. Guided test generation for web applications. In *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 162–171.

[57] 2022 Tricentis. 2022. Tricentis Tosca: Intelligent Test Automation. https://www.tricentis.com/products/automate-continuous-testing-tosca/. Accessed: 2022-07-05.

[58] Marialena Vagia, Aksel A Transeth, and Sigurd A Fjerdingen. 2016. A literature review on the levels of automation during the years. What are the different taxonomies that have been proposed? *Applied ergonomics* 53 (2016), 190–202.

[59] Tanja E. J. Vos, Pekka Aho, Fernando Pastor Ricos, Olivia Rodriguez-Valdes, and Ad Mulders. 2021. testar scriptless testing through graphical user interface. *Software Testing, Verification and Reliability* 31, 3 (April 2021). https://doi.org/10.1002/stvr.1771

[60] Benedikt Walter, Maximilian Schilling, Marco Piechotta, and Stephan Rudolph. 2018. Improving Test Execution Efficiency Through Clustering and Reordering of Independent Test Steps. In *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 363–373.

[61] Chunhui Wang, Fabrizio Pastore, Arda Goknil, and Lionel Briand. 2020. Automatic Generation of Acceptance Test Cases from Use Case Specifications: an NLP-based Approach. *IEEE Transactions on Software Engineering* (2020), 1–1. https://doi.org/10.1109/TSE.2020.2998503

[62] Chunhui Wang, Fabrizio Pastore, Arda Goknil, and Lionel Briand. 2020. Automatic generation of acceptance test cases from use case specifications: an nlp-based approach. *IEEE Transactions on Software Engineering* (2020).

[63] Chunhui Wang, Fabrizio Pastore, Arda Goknil, Lionel Briand, and Zohaib Iqbal. 2015. Automatic generation of system test cases from use case specifications. In *Proceedings of the 2015 international symposium on software testing and analysis*. 385–396.

[64] Tao Xie. 2006. Improving effectiveness of automated software testing in the absence of specifications. In *2006 22nd IEEE International Conference on Software Maintenance*. IEEE, 355–359.

[65] Yan Zheng, Yi Liu, Xiaofei Xie, Yepang Liu, Lei Ma, Jianye Hao, and Yang Liu. 2021. *Automatic Web Testing Using Curiosity-Driven Reinforcement Learning*. IEEE Press, 423–435. https://doi.org/10.1109/ICSE43902.2021.00048

[66] Celal Ziftci and Diego Cavalcanti. 2020. De-flake your tests: Automatically locating root causes of flaky tests in code at google. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 736–745.