Received October 7, 2021, accepted October 25, 2021, date of publication November 12, 2021, date of current version December 7, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3128071

Toward a Multi-Criteria Framework for Selecting Software Testing Tools

ASMA J. ABDULWARETH[1] AND ASMA A. AL-SHARGABI[1,2], (Member, IEEE)
[1]Department of Computer Science, College of Computing and IT, University of Science and Technology, Sana'a, Yemen
[2]Department of Information Technology, College of Computer, Qassim University, Buraydah 51452, Saudi Arabia

Corresponding author: Asma J. Abdulwareth (aj.alsharjabi@gmail.com)

ABSTRACT Software testing is a vital part of software engineering process. Automated testing makes this process more accurate and more efficient. For automated testing, many different testing tools were introduced. Due to the large number and the variety of testing tools, selecting the appropriate tools became a difficult confusable task. This research aims at developing a comprehensive taxonomy for testing tools that cover a broad range of testing tools criteria. This comprehensive view would help software developers and software vendors to specify the testing tool/s they need/develop accurately. In details, the framework includes two main parts: (1) comprehensive taxonomy of testing tools; (2) multi-criteria selection method. The first part covers different criteria of testing tools. Because these criteria are large in numbers, wide and variant, a taxonomy of these criteria is needed. This taxonomy will help developers distinguish among testing tools based on a wide spectrum of different criteria. The second part of the framework is a multi-criteria selection method; that enables software developers to choose the appropriate testing tool using a systematic and adequate automated manner. The selection method employs scientific two well-known methods of multi-criteria decision-making techniques; Analytic Hierarchy Process (AHP) and Technique for Order Preference by Similarity to Ideal Solution (TOPSIS). The testing tools taxonomy is well validated by academic professionals in software engineering and achieved good scores in terms of significance, usefulness and comprehension. Academics reported that the taxonomy is slightly complex and needs to be simplified. The selection method was validated using different scenarios to prove the quality of selection even in complex cases with many criteria and many alternatives.

INDEX TERMS Software testing, automated testing, testing tools, taxonomy of testing tools, testing tools classifications, testing tools selection.

I. INTRODUCTION

Software is one of the most important aspects of technological advances in the world. Software affects all aspects of our daily life and control most other fields and disciplines [1], [5], [20], [24], [72]. Within the heart of software development life cycle (SDLC), testing represents a milestone process to check the validity and quality of software [9]. In software industry, testing occupies around 40% of the SDLC, regardless the software development approach that is used [10], [15], [22]. Therefore, testing should be managed and achieved using the best practices to develop high quality software with cost-effective approaches [32], [33], [72].

The associate editor coordinating the review of this manuscript and approving it for publication was Giuseppe Destefanis.

IEEE defines software testing as the process to evaluate a system or its components manually or by automated means to determine whether it fulfills the user requirements or to find the differences between actual results and expected results [56]. Manual testing is performed by interacting with the software based on some predefined test cases. Test cases provide explanations of the features examined and the expected results. The process of manual testing is time-consuming [37] and the problem becomes worse when we repeat the tests after each correction and with each related feature. Automated testing makes this process more effective [37]. We can define test cases just once and we can exercise the software many times automatically. There are many varied tools that provide automated testing. These tools exercise the software against some predefined test cases; and compare the actual outcomes with expected ones [5], [17], [18]. If there are deviations in the results, this means there are bugs that need to be spotted and fixed. The developer have to look through the code, spot and correct the bugs, and continue running tests until the actual and expected results be identical [16]. Test cases can be arranged in suites to facilitate testing [3], [5], [13], [17]. Utilizing testing CASE tools helps in reducing the amount of time and money spent on projects, improving the quality of the system developed, enhancing the developers' productivity and satisfaction [7], [13], [15], [39], [69].

SW industry sector introduces a huge number of testing CASE tools. SW developers face difficulties when choosing the testing tools that are suitable for their systems. This is normal given the wide variety of software features and the difficulty of having a testing tool that includes all this diversity [2], [11], [31], [42]. There are many factors that affect your choices such as time, resources, the size of the SW, and the required quality level. With large-scale software, it usually combines between the manual and the automated testing.

There are many testing tools classifications and different approaches/views to build these classifications. There is a need for a broad multi-view testing tools classification, and also there is a need for a method to use and utilize this classification.

To help SW developers choosing the appropriate tool/s for particular software, this research aims at introducing a comprehensive taxonomy of testing tools. This taxonomy introduces a wide knowledge base about testing tools types, this would help SW developers select testing tools accordingly. It also could help tools developers introduce better products based on this comprehensive taxonomy.

The paper structure is as follows: Section II is devoted for related work and research gaps, Section III explains the novelty of work and introduces the proposed framework constructs, Section IV includes the experiments and evaluation, and Section V is devoted for conclusions & future woks.

II. RELATED WORK

Software testing is a vital phase in software life cycle. Software industry pay a strong attention for this phase due to the important effects on software quality. Whatever the software development approach, testing occupies not less than 40% of the total software project time [1], [8], [32], [42]. Testing automation is not a new idea. Actually, many tools are utilized in different software development phases. In the testing phase, there are many testing tools which can be used effectively to speed up, control, and improve the testing process [8], [11], [37], [42], [65]. To facilitate using the testing tools, the research community introduced different approaches of testing-tools classifications [13], [37], [39], [42], [65]. Beside taxonomies and classifications, other researchers have addressed the key factors of testing-tools selection and suggested some guidelines to choose testing-tools easily and systematically [8], [13]. Basically, testing tools is selected and used from software developers/testers based on testing views, software features, or/and testing tools features [23], [42], [52]. There are different views of testing, we can address software testing based on testing levels [25], [31], [35], testing models [4], [19], [21], [23] testing types [4], [21], [33], and testing techniques [6], [21], [25], [31]. Similarly, software that has being tested, also has different features such as programming language, data model, and platform [14], [30], [73].

Before addressing what has been done regarding the testing-tools taxonomies, some testing terminologies should be reviewed. The coming two sections are devoted to introduce the testing views and terminologies. Terms that are introduced are unit testing, integration testing, system testing, acceptance testing, static testing, dynamic testing, testing management, testing preparation, testing implementation/execution, testing evaluation, functional testing, usability testing, reliability testing, security testing, performance/stress/load/volume testing, and maintainability testing.

Testing levels is decomposed into: unit testing, integration testing, system testing, and acceptance testing. Unit testing is concerned with testing the smallest component in software individually. Components could be an object, a method if the programming language is object oriented, a function, or a procedure if the programming language is structured language. Unit testing focuses on the function of the tested unit/component. It is usually done by the developer himself. Unit testing is usually white-box, which means that the test-cases and test-data are selected based on the structure of the code. Integration testing deals with testing the integration between two or more software components. It is usually achieved after testing the different component individually, therefore, it focuses on testing the interfaces between components and the whole functionality of the all-integrated components collectively. System testing -as its name indicates- is concerned with testing the overall system as a whole. It is basically achieved to test all the functions and services regardless the components that it contains. It is usually black-box testing, where black-box means testing the actual output using a general selected input and expected/targeted output. Beside functionality, system testing aims at testing other system behaviors/features such as security, performance, and usability. These prior features are usually tested at system level because of testing at unit level does not make sense. Unit, integration, and system testing are usually performed for software verification which means that all tests are derived from the requirements' specifications. Acceptance testing is the final test level that is achieved by the user and targets the system validation, which means that the test-cases and test-data are derived and managed directly by user. System verification aims at ensuring that the software satisfies user needs and customers' expectations. There are different forms of acceptance testing, the common two types are alpha-test, and beta-test. Alpha-test is achieved in the developer organization and environment whereas beta-test is achieved within user organization and real environment [6], [25], [61], [65].

Testing basically aims at testing the code. Testing the code could be performed using two general techniques; static techniques and dynamic techniques. Static testing tests the code without running, and this is why it is called static. It is performed for errors detection or code standards convention. It is performed by going through the code and tracing the error causes or the standards' violations. Dynamic testing tests the code in running mode. It is performed according to a pre-prepared input and against an expected output [4], [18], [21], [25], [31]. Within dynamic Test criteria, input and the expected output are prepared based on different techniques. The common techniques for inputs/outputs preparing are equivalence partitioning, boundary value analysis and model-based. The model-based technique is also used for deriving/extracting the test-criteria. Model-based examples are decision-table based, state-transition testing and any other specification-based/model-based technique [9], [14], [21], [58], [65].

According to the testing process/lifecycle view, many testing stages are involved which are test-analysis/test-planning, test-design, test-implementation/execution, and test-evaluation. Test-analysis phase is concerned with studying all testing artifacts (testware). Test-design phase builds on test-analysis phase with creating the test-cases/criteria and test-data. Test execution/implementation deals with performing the different prepared testes and recording the results. Test-evaluation phase aims at comparing the test results against the expected results that were specified in test-analysis and test-design phases. Finally, test management is an umbrella activity that controls all testing processes. Test management aims at monitoring, and controlling the whole test process [23], [65].

Based on software quality models, there are different characteristics [52]. The common characteristics are: functionality, reliability, security, maintainability, usability, portability, and efficiency [59], [60]. Quality characteristics include all non-functional requirements for software. Excluding functionality, most test types and test tools target these characteristics in some way, for example: security testing, performance testing, usability testing,…etc. This is the reason why we consider quality characteristics as an important view of software testing and software testing tools.

SW development life-cycle and SW process models are other factors that affect SW testing practices. V-model is mostly used with waterfall model whereas test-driven/test-first approach is used with agile methods, besides there are testing processes for object-oriented systems [16], [19], [26], [31], [33], [35], [65], [68].

Based on Technical and Economic Factors, there are several criteria. According to relevant literature, criteria such as budget, vendor, software technology, and platform can be addressed [14], [16], [27], [33], [30].

After this overview of SW testing, these different views are employed for testing tools classification. In addition, these views inspired other researchers in their classification approaches. The following section illustrates the previous work on testing-tools classifications. Literature addressed testing-tools categories based on different views [42]. However, we can cluster these approaches in two general categories; the first one is top-down and the second one is bottom-up. The top-down approach utilizes the testing notion, types, techniques and all views that were mentioned in the previous section to classify the testing tools. The bottom-up approach starts from the tools themselves and go up by clustering the tools into similar groups.

A worthy milestone is what is introduced by Dorothy Graham in 1991. The introduced scheme proposed six major categories: Test Management, Test Design, Non-Execution Evaluation, Test Execution, Test Analysis, and Test Quality Evaluation [64]. The scheme was simple, deep, scalable, and comprehensive at the same time. Most classifications that came latter use this scheme as a reference model. Dorothy 's Scheme include many sub-categories within each main category. The scheme was -and still- perfect. Actually, this taxonomy works well till now. However, with the current big leap in software testing and in automation in general, many subcategories can be added/removed. Whatever the tool you want to classify, it will be involved easily to one or more category within this taxonomy. The boundaries between the categories are not sharp, however a clear discrimination between them was introduced. Therefor a testing tool can belong to more than one category where it can support many testing functionalities. A brief description for major categories is as follows [64]:

Test management tools: this category covers all activities of risk assessment, planning, resource allocation monitoring, and control of the testing process. **Test design tools**: this category is concerned with different aspects of test design including deriving, managing, and documenting of test criteria, test inputs, test data, data structures, and test environment.

Non-execution evaluation tools: these tools provide support for testing activities which evaluate software quality without executing the code.

Test execution tools: this category conveys tools that support the process of exercising software with pre-defined test inputs. The target of test could be a system function or a system behavior. Consequently, confirmation testing, regression testing, and stress testing are sub-categories of this major category. In confirmation testing, test input must be checked against the expected output to ensure that the software is preformed correctly. Confirmation testing can be made for new code or after change. Regression testing is performed after a change has been made to ensure that there are no adverse side-effects. Stress testing is performed to assess the software's resilience to excessive size or rate of input.

Test analysis tools: these tools are concerned with assessing and dealing with the results of dynamic test runs, that is, analyzing the cause of failure in order to identify faults or defects. Test analysis tools encompass all debugging tools, including those for special type of systems such as real-time systems and distributed systems.

Test quality evaluation tools: this category encompasses tools that are used for assessing the quality of testing. All coverage tools belong to this category.

Between 1988 and 1992, ISO 9126 emerged [59], then IEEE standards followed; 1061(IEEE, 1992), 1209 (IEEE, 1992), and 1348 (IEEE, 1995) [48], [58], [60], [65]–[67]. These criteria were issued for software CASE tools selection and adoption, and for software quality criteria. These criteria affected some other taxonomies that appeared later. The software quality aspects that could be used as a basis for software testing tools taxonomy are as follows:

ISO 9126 classification:

ISO 9126 introduces six quality attributes of software. These attributes could be considered as a basis of software testing aspects and also testing tools classification. These quality attributes are as follows [59]:

Functionality - "A set of attributes that bear on the existence of a set of functions and their specified properties." The functions are those that satisfy stated or implied needs." These attributes are:

- Suitability
- Accuracy
- Interoperability
- Security
- Functionality compliance

Reliability - "A set of attributes that bear on the capability of software to maintain its level of performance under some stated conditions for a stated period of time." which are:

- Maturity
- Fault tolerance
- Recoverability
- Reliability compliance

Usability - "A set of attributes that bear on the effort needed for use, and on the individual assessment of such use, by a stated or implied set of users." These attributes are:

- Understandability
- Learnability
- Operability
- Attractiveness
- Usability compliance

Efficiency - "A set of attributes that bear on the relationship between the level of performance of the software and the amount of resources used, under stated conditions"; which are:

- Time behavior
- Resource utilization
- Efficiency compliance

Maintainability - "A set of attributes that bear on the effort needed to make specified modifications." These are:

- Analyzability
- Changeability
- Stability
- Testability
- Maintainability compliance

Portability - "A set of attributes that are born on the ability of software to be transferred from one environment to another."

- Adaptability
- Installability
- Co-existence
- Replaceability
- Portability compliance

IEEE 1061 classification: This classification benefited IEEE Std 1209-1992, IEEE Std 1348-1995, and IEEE Std 1061-1992 (ISO 9126) that illustrates a possible set of software quality factors and tools criteria. IEEE quality factors are the same of ISO 9126: **Efficiency, Maintainability, Portability, Reliability, Usability, and functionality** [48], [60], [66], [67]. It is worthy to say that, "security" -which became a basic requirement today for any type of software- is not included in IEEE 1061, and included as a sub-criterion within ISO 9126. This neglect was reasonable at that time; however, today we are in need to include security as a basic quality requirement for software.

In 1999, Mark Fewster with Dorothy Graham built on what was introduced in [61] and [64]; they classified testing tools based on V-model or testing life-cycle as illustrated by Figure 1. The main classes are as follows:

Test design tools: these tools are used in requirements specification, architectural design, and detailed design. Test design tools help derive test inputs or test data.

Static analysis tools: these tools are used in coding/implementation phase. Static analysis tools analyze codes without executing them. This type of tool detects certain types of defects much more effectively and cheaply than can be achieved by any other means. Such tools also calculate various metrics of code such as McCabe's cyclomatic complexity, Halstead metrics, and many more.

Coverage tools: they are used in unit testing phase. Coverage tools assess how much of the software under testing has been exercised by a set of tests.

Dynamic analysis tools: they are used in integration testing phase. Dynamic analysis tools assess the system while the software is running. For example, tools that can detect memory leaks are dynamic analysis tools.

Performance testing: used in system and acceptance testing. This type measures the time taken for various events. For example, they can measure response times under typical or load conditions.

Test execution and comparison tools: used in unit testing, integration testing, and system testing as well. Test execution and comparison tools enable tests to be executed automatically and test outcomes to be compared to expected outcomes.

Test management tools: used at all phases. This type is used to assist in test planning, keeping track of what tests have been run, and so on. This category also includes tools to aid traceability of tests to requirements, designs, and code, as well as defect tracking tools.

In [61] the classes of testing tools were slightly more fine-grain. Some sub-categories in [64] appear as a major

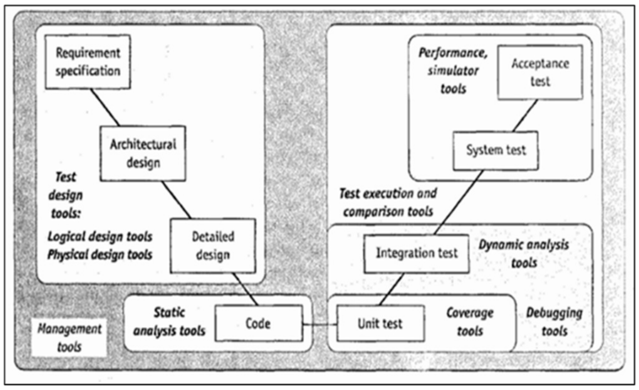

FIGURE 1. Testing tools spans the software development life cycle [61].

category in this taxonomy. Non-execution tools appear as Static analysis tools category and separated from other analysis tools (Dynamic analysis), and test evaluation tools entitled as Coverage tools. Stress testing which is included within Test execution class is also isolated in broader category class that is entitled as Performance testing. Performance testing includes different aspects such as speed and stress/load. Security testing completely disappeared [61], [64].

Author in [25] experiences 600 testing tools and then classifies testing tools into 7 main groups. Each main group has some sub-groups. The main divisions encompass [25]:

Test Design Tools: Tools that help you decide what tests need to be executed. Test data, test database and test case generators are subgroups in this division.

GUI Test Drivers: Tools that automate execution of tests for products with graphical user interfaces. Client/server test automation tools, including load testers, also go here.

Load and Performance Tools: Tools that specialize in putting a heavy load on systems (especially client-server systems). These tools are often also GUI test drivers.

Test Management Tools: Tools that automate execution of tests for products without graphical user interfaces. Also, tools that help you work with large test suites.

Test Implementation Tools: Miscellaneous tools that help you implement tests. For example, tools that automatically generate stub routines go here, as do tools that attempt to make failures more obvious.

Test Evaluation Tools: Tools that help you evaluate the quality of your tests. Code coverage tools go here.

Static Analysis Tools: Tools that analyze programs without running them. Metrics tools fall in this category.

Authors in [25] adopt a practical top-down strategy by studying more than 600 CAST before introducing their classification. Many categories still the same as those in [61], [64], and this is a positive indicator for stability of categories in [61], [64]. The classification in [25] distinguishes between "GUI test drivers' tools" and "test implementation tools," whereas they both belong to "test execution tools" in [61], [64]. Load testing tools appear obviously in the title of "load and performance tools." The category of "GUI test drivers" indicates a new trend to classify CAST based on the tool technical features. The category of "test implementation" is used for what is called in previous works by "test execution" category.

In 2004, SWEBOK standards for software engineering body of knowledge was issued by IEEE and ISO [47], [48]. SWEBOK classified testing tools based on functionality into the following categories [47], [48]:

Test harnesses (drivers, stubs): these tools provide a controlled environment in which tests can be launched and the test outputs can be logged. In order to execute parts of a program, drivers and stubs are provided to simulate calling and called modules, respectively.

Test generators: provide assistance in the generation test cases. The generation can be random, path-based, model based or a mix thereof.

Capture/replay tool: this category automatically re-executes or replays previously executed tests which have recorded inputs and outputs (e.g., screens).

Oracle/file comparators/assertion checking tools: these tools assist in deciding whether a test outcome is successful or not.

Coverage analyzers and instrumenters: These work together. Coverage analyzers assess which and how many entities of the program flow graph have been exercised amongst all those required by the selected test coverage criterion. The analysis can be done because of program instrumenters that insert recording probes into the code.

Tracers: record the history of a program's execution paths.

Regression testing tools: support the re-execution of a test suite after a section of software has been modified. They can also help to select a test subset according to the change made.

Reliability evaluation tools: Support test result analysis and graphical visualization in order to assess reliability-related measures according to selected models.

The former category obviously adopted the technical nature of the tools themselves as a basis of classification. "Test harness (drivers, stubs)" which is a common terminology at that time occupies a main category. This manner was the common technical method to implement the unit testing. Similarly, "capture/reply tools" are separated with a main category. Capture/reply title indicates the technical feature of these tools. "Tracers" and "coverage" tools are distinguished to indicate that the former is performed in run-time where the latter is not, where both perhaps trace the code paths. "Execution tools" are spans many categories within this classification, they are: test harness, capture/reply, comparators, tracers, and regression categories. Reliability tools is dedicated with the main category, with a clear technical feature which is graphical visualization. Reliability criteria could be spired with software quality criteria in standards [48], [66], [67].

In 2005, author in [21] categorizes testing tools based on objectives and features into the following categories:

Function/Regression Tools — These tools help you test software through a native graphical user interface. Some also help with other interface types. Another example is Web test tools that test through a browser, for example, capture/replay tools.

Test Design/Data Tools — These tools help create test cases and generate test data.

Load/Performance Tools — These tools are often also GUI test drivers.

Test Process/Management Tools — These tools help organize and execute suites of test cases at the command line, API, or protocol level. Some tools have graphical user interfaces, but they don't have any special support for testing a product that has a native GUI. Web test tools that work at the protocol level are included here.

Unit Testing Tools — These tools, frameworks and libraries support unit testing, which is usually performed by the developer, usually using interfaces below the public interfaces of the software under test.

Test Implementation Tools — These tools assist in testing at runtime.

Test Evaluation Tools — Tools that help you evaluate the quality of your tests. Examples include code coverage tools.

Static Test Analyzers — Tools that analyze programs without running them. Metrics tools fall in this category.

Defect Management Tools — Tools that track software product defects and manage product enhancement requests. They manage defect states from defect discovery to closure.

Three titles should be noted in this classification, they are "Function/regression testing" tools, "Unit testing tools," and "Defect management tools" [21]. Regression testing which was included as a sub-set category from "execution tools" in [64] is separated from "function testing" in dedicated category. This manner in classification could be affected by ISO and IEEE standards [47], [48], [59], [60], [66], [67]. The category of "Unit testing" appears clearly as a main category, without including the rest test levels. This, perhaps, indicates the focus of tools vendors and SW developers on unit testing at that time. "Defect management" is new category devoted for tools that support defect tracing, perhaps because of its importance; whereas in prior work it is included within "test management" category.

Within ISTQB, Dorothy Graham classified testing tools based on testing activities into the following classes [58], [65]:

Tool support for management of testing and testware:
- Test management tools and application lifecycle management tools (ALM)
- Requirements management tools (e.g., traceability to test objects)
- Defect management
- Configuration management tools
- Continuous integration tools

Tool support for static testing:
- Static analysis tools

Tool support for test design and implementation:
- Model-Based testing tools
- Test data preparation

Tool support for test execution and logging:
- Test execution tools
- Coverage tools
- Test harnesses

Tool support for performance measurement and dynamic analysis:
- Performance testing tools
- Dynamic analysis tools

A. TOOL SUPPORT FOR SPECIALIZED TESTING NEEDS

This category involves all other specific testing types for non-functional aspects.

ISTQB Classification has more discrimination between the different categories with simple hierarchy of main categories and subset categories. "Test implementation tools" has clear boundary of "Test execution tools." Model-based testing tools is introduced to specify the tools that adopt a particular SWE model such as UML or state-transition model, this title left the door open for any new model supportive tool. "Execution tools" encompass "coverage tools" and also "test harnesses" and this is better, logical and simpler than the prior classifications. "Performance tools" is a broad terminology which can include different testes such as load testing and stress testing. The last category could include all non-functional aspects tests such as security, usability, reliability… etc. Security which is one important testing type currently is included in the open category in this classification.

In 2010, author in [35] classifies testing tools basically based on test levels. In some subset of classification ISO 9126 and SWEBOK classifications were referenced. Reference [35] also benefited from what was introduced in [62]- as both researches were conducted at the same university. This classification is broader than all prior classifications. To make it short, the new categories are described where the old ones are already described in previous classifications. This approach classifies testing tools into the following categories:

Unit Testing Tools: This class includes the following tools classes [35]:

- **Manual Program Execution**: The whole program is being run. Proper parameter values are derived by manual calculation in order to invoke the required unit. The main disadvantage of this approach is that it is very time-consuming, considering that a unit is tested several times with different test data, and it requires writing client cod.
- **Automated Test Drivers**: It is test harness (stub, drivers) tools.
- **Direct Test Access:** The tools can provide the same functionality as automated test drivers but without the need of constructing stubs. It allows the direct control of the unit under test, without taking the unit out of its operational environment.

B. INTEGRATION TESTING TOOLS

- **Top-down integration Testing Tools:** In this approach the stubs tools are used (the same as in unit testing). This fact explains why software testing tools, initially designed for unit testing, are also used in integration testing.
- **Bottom-UP integration Testing Tools**: Bottom-up integration starts from construction and testing components at the lowest level of the program. In this approach no stub tools are used, because all the required processing information of a component is already available from the previous steps. Tools that are used for integration testing again correspond to those for unit testing activity (test drivers). This could be almost considered as an extension of unit testing.
- **Regression Testing Tools:** This activity can be carried out using automated capture/playback tools, which allow testers to record test cases and repeat them for following result comparison. Regression testing often starts when there is anything to integrate and test at all. Test cases for regression should be conducted as often as possible.

C. FUNCTIONAL TESTING TOOLS

Functional testing tools ensures that the tested function produces the expected outcome, as it is described in (ISO 9126, 1988) for functionality quality characteristic.

D. SYSTEM TESTING TOOLS

System testing tools support a different test whose primary purpose is to fully exercise the whole computer-based system. In addition, this category includes recovery, security, stress and performance testing. Among the existing tools, there are subsets that focus on specific security areas: database security, network security and web application security. This category includes the following sub-categories.

- **Security Testing Tools:** Security testing relies on human expertise much more than an ordinary testing, therefor automation of the security test is less achievable than other testing types. However, there are different black box test tools designed for testing application security issues.
- **Performance and Stress Testing Tools:** Performance tests are often coupled with stress testing. Stress testing is conducted to evaluate a system at the maximum design load, while performance testing aims at verifying that the software meets the specified performance requirements. The main principle of operation of performance and stress testing tools is simulation of a real user with "virtual" users. The tool then gathers the statistics on virtual users' experience. In general, performance testing tools can be divided into load generators, monitors and frameworks which are used for finding performance bottlenecks, memory leaks and excessive memory consumption.

E. ACCEPTANCE TESTING TOOLS

This testing activity differs from others in aspect that it may or may not involve the developers of the system, and can be performed by the customer. If some errors are identified during acceptance testing, after developers correct them or after any change, the customer should go through acceptance tests again.

The latter classification is natural accumulated growth of the different prior classifications. It contains two levels of categories which allow to encompass easily the different testing tools. Most categories in prior classifications are involved. Security which became a basic recommended software feature has a sub-category within system testing main category. Unit testing tools include three types of tools based on the technical implementation of the testing tool. According to ISO 9126, functional testing tools has a separate category. However, the overlapping between functional category and other categories was not clearly described. Some main categories were disappeared, such as coverage tools, and testing evaluation tools. The idea of classification based on testing levels is useful for both developers and testers, as it arranges tools usage process in a systematic manner.

In 2013, author in [33] introduced the following classifications based on the nature of the testing process:

Management tools: This class includes different management classes as follows:

- **Test management tools**: this category includes tools that are used to manage the different assets of testing process. Assets could be test cases, defects, or test tasks.
- **Requirement management tools**: These tools are usually used in software analysis phase; however, it is also used in the testing phase to link tested units and/or defects with the relative requirements. These links are useful to estimate the severity of bugs and locating the infected areas.
- **Incident management tools**. This category includes the tools that are used to document and track system failures and anomalies. It is also known as defects management or bugs-tracking tools.
- **Configuration management tools**. These tools are used to maintain changes during the testing.

Execution tools: This category encompasses the tools that are used with code running mode. It includes:

- **Test execution tools:** This is the most commonly known category. It is also known as test running, or capture and replay tools. These tools are basically used for automation of regression testing. These tools reduce test execution time significantly when tests are repeated and/or allow more tests to be executed.
- **Test harness/unit test framework:** This type of tools is usually used by developer to test a unit of code. It provides stubs or drivers to interact with the unit under test.
- **Test comparator:** These tools are used to compare the actual results of programs with predefined ones.

- **Coverage measurements:** tools in this category are used to examine the coverage of code being tested. It provides a clear image about how thoroughly software is tested.
- **Security tools:** This category includes tools that check if there is any data leakage.

Static testing tools: This class includes tools that are used in different types of static testing. It supports formal reviews, inspections, and walk-throughs. This category includes:

- **Review process support tools:** this type is used to support managing static reviews assets.
- **Static analysis tools:** Tools that fall in this category are used to analyze the syntax and also the semantic/logic of code without running it.
- **Modelling tools:** This category includes tools that support a particular design model. It helps generating data for that model. For example, generating test data for such state transition diagram.

Performance testing tools. This type is an umbrella category for all tools that are used to test effectiveness and speed of software; for example, stress testing, load testing, spike testing tools fall in this category. Testing tools for web applications usually support performance testing that is supposed to be used by a lot of different customers. This category includes:

- **Load testing tools:** This type of tools is used to examine the behavior of the software when there are large numbers of users at the same time. These tools are used to detect the system blockage and how to respond when the load is gradually enlarged.
- **Stress testing tools:** This category includes tools that examine the ability of the system to retain a reasonable level of efficiency within harsh conditions.
- **Dynamic analysis tools:** This type is dynamic because it is performed in running mode; however, it is different of dynamic testing. Tools in this category are used to explore what happens with code behind the scene; Whether the code is executed by particular test case or used by operation correctly.
- **Monitoring tools:** Tools in this category provides insight of different system aspects such as memory, CPU while system running.

Test specs tools: Tools in this category are usually used to help design test cases or generate test data. They usually support a particular requirements specification format/model. They provide information that help choose the test type for a particular target test level or type. For example, they could specify all true combinations of such classification tree. This information will help create the true data that cover all paths in the tree.

Taxonomy that has been introduced in [33] is slightly similar to ISTQB classification. It introduces a fine-grain classification with two-levels of categories. Many details and sub-categories are clearly involved and described. Management tools include all possible tools with three sub-categories. Security tools are included within execution tools, where execution tools category is a broad category that includes test execution tools, coverage tools, test harnesses, and security tools. Similarly, performance tools category is broad and includes load testing tools, stress testing tools, dynamic analysis tools and monitoring tools with clear discrimination between them. Monitoring tools appear as a separate category as they became popular. Modeling tools category is included within the static testing tools, although there is a test specs tools category that conveys the tools that are used for generating test data based on a specific design model. Nevertheless, this model is comprehensive, simple and clear.

On the other hand, there are variant testing tools that were developed for specific type of software, technology, or specific software criteria. Software industry sector introduces different testing solutions based on application technology such as web applications testing tools [14], [30], mobile applications testing tools [28], [52], [53], [55], [56], and embedded software testing tools [12], [30]. These solutions serve in different ways, there are stand-alone systems, web services, or even cloud-based services. Besides, there are tools that are built for specific type of software such as object-oriented software [25], [26], aspect-oriented software [63], and web-services systems [38]. For many software criteria/quality-aspects, there are many tools in software market. For example, there are tools for security, reliability, usability, and maintainability [19], [21], [57], [63].

Finally, there is a lack of a systematic comprehensive way to classify testing tools [19], [21], [57]. There are many views that could be used for classification. A comprehensive taxonomy will help both developers/testers and vendors and will help using automation tools in software industry. On the other hand, there is no a method to use testing tools taxonomies; there are a need for such methods to facilitate selection between the wide range of testing tools.

III. PROPOSED MULTI-CRITERIA FRAMEWORK FOR SELECTING TESTING TOOLS

This section introduces a framework of testing tools selection. This framework aims to help software developers to select appropriate testing tool/s accurately from large number of testing tools based on broad criteria classification. The novelty of this paper is represented by: 1) introducing a comprehensive taxonomy of SW testing tools as the previous taxonomies are usually admit just one view, so they usually not comprehensive and cannot cover all testing tools criteria; 2) introducing a scientific method to utilize the introduced taxonomy to select the appropriate tool/s even in case of complex decisions with many criteria and many alternatives. A multi-criteria decision making is used. This method can then be built as an engine and produced as a web service to facilitate choosing the appropriate testing tools or a particular software.

The proposed framework consists of two main parts; (1) the taxonomy categories, (2) a multi-criteria selection method for choosing the appropriate tool/s for a particular software.

The first part which is the taxonomy tries to involve many criteria of testing tools as possible; the aim is a comprehensive taxonomy that could help both users and vendors as well, with automatic accurate selection method. The introduced selection method uses two well-known methods of multi-criteria decision-making techniques, AHP (Analytic Hierarchy Process) and TOPSIS (Technique for Order Preference by Similarity to Ideal Solution). AHP and TOPSIS are used in many fields to support decision making, especially in business. This is why we think to utilize in the context of SWE decisions. AHP and TOPSIS performed an accurate rank of decision alternatives even where the decisions are complex with many criteria and many alternatives. Accuracy here means producing an accurate rank for alternatives [between 1 - 0] [43]–[46]. Figure 2 illustrates the components of the proposed framework.

For industrial sector, this framework would contribute to make automated testing easier and more efficient. It could be developed as an agent within website and then fed by a large knowledgebase of all available testing tools and their classification based on the proposed taxonomy. This may contribute to reducing software developers' reluctance to use testing tools as described in Section I. On the other hand, software developers can utilize the proposed taxonomy to choose the type of their products based on broad view. Consequently, test automation practices will be improved.

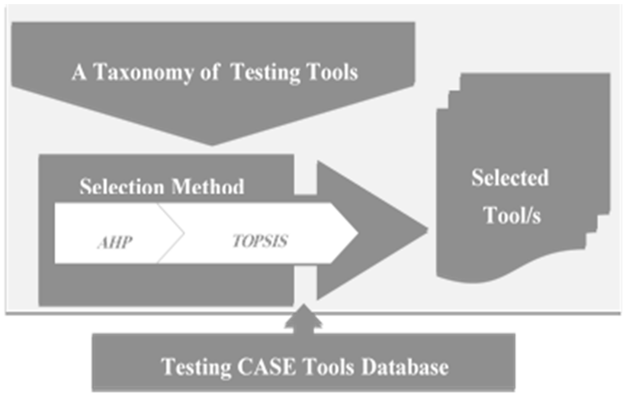

FIGURE 2. Multi-criteria framework for selecting testing tools.

A. A COMPREHENSIVE TAXONOMY OF TESTING TOOLS

This section introduces the first part of the framework, the different categories of the introduced taxonomy. The proposed taxonomy encompasses three basic classes depending on the similarities on the different criteria in the literatures as shown by Figure 3 and TABLE 1. These three classes are: (1) technical and economic view, (2) testing process view, and (3) Software quality standards. The taxonomy includes three abstraction levels as shown in TABLE 1. This is due to the different abstraction/refinement level of the different criteria. To make the taxonomy comprehensive, all testing tools criteria -which are addressed in the literature- is included in the proposed taxonomy within the appropriate class/sub- class. The third/final level includes the criteria that the user can directly deal with. According to the nature of some sub-classes, the scalability by adding new criteria is taken into consideration. For example, the "SW operating system/platform," and "Vendors" criteria includes open slot.

Each class in the proposed taxonomy is explained below.

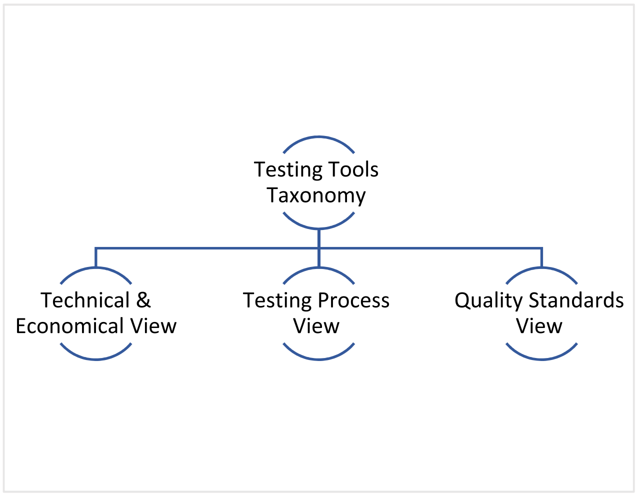

FIGURE 3. The proposed taxonomy for testing tools (the main categories).

1) TECHNICAL AND ECONOMIC VIEW

This class includes criteria that are usually first considered by tools customers (developers/testers). Most literature classifies testing tools based on these criteria. Is the tool technically appropriate for the software being tested? and how easy we can get the tool? These questions represent the basic considerations for customers. We can say that, this class represents the first layer/filter of criteria; Where after that, other criteria within other classes can be investigated. This class include variant technical, and economic criteria such as platform, software technology, and programming languages. As mentioned earlier, some sub-criteria slots are open to include any new criteria.

2) TESTING PROCESS VIEW

Testing process view is the second class in the proposed taxonomy. This class was very complicated at first as there were many different dimensions within it. However, it was subjected to several refinement cycles to make it simpler while maintaining the same level of richness. The complexity came not just from the variant views, but also from the large amount of possible interlapping links between these views. For example: we can include the test technique under test levels and vice versa. For simplification, we avoid making the classification in more than three-levels. Therefore, the criteria are not mutually exclusive where a particular test could be classified as: dynamic testing, unit testing, functional testing, and black-box testing at the same time. Besides, testing tools is already classified in the literature based on all these criteria as illustrated by TABLE 1. The sub-class "Function of testing tools'' is inspired basically from ISTQB classification [7], [58], [65], IEEE SWEBOK classification [38], [47], [48], and many other researches as anticipated in TABLE 1 and as described in related work section. ''Function of testing tools'' category includes different functions such as test management, test design, and performance testing. Functions that go together and hold different titles, or tools that has the same function are included in the same sub-criteria such as ''Test Design/Model-based Testing'' and ''Test Execution/ Comparators/Test Harness.'' Security testing represented a separate category. It is not included clearly within both SEBOK and ISTQB; however, it is included in many other researches as clarified in related work. In IEEE 106, it is included within ''functionality'' criteria as one aspect of functionality. Security became a basic feature for software today where IOT platform became the future platform, so it is better to have it as a separate clear category. The last category within this class is ''Software testing Model/Approach.'' This category includes the different SWE models that was supported by testing tools such as ''Agile'' and Service-Oriented-Architecture SOA. We found many testing tools are built and worked based-on.

3) SOFTWARE QUALITY STANDARDS

SW quality standards can be employed as one perspective of testing-tools classification that specifies the general quality aspects for software. Some criteria within these standards are already included within the previous classes, such as ''Efficiency'' in standards and ''Performance Testing'' in the prior class. Similarly, ''Maintainability'' in standards and ''Static Analysis'' in the second class, where maintainability is measured using different metrics using static-analysis testing tools. On the other hand, ''Security'' is not included in this class clearly, although it is a sub-quality-criteria within ''Functionality'' criteria in standards. This is because it is already included in the second class as described earlier. Finally, for all quality criteria in this class, several tools can be found in SW market.

B. A MULTI-CRITERIA METHOD FOR SELECTING TESTING TOOLS

This section introduces the second part of the proposed framework, which is a multi-criteria selection method for testing tools. This method is proposed to simplify the selection process among the huge number of testing tools. Besides, the criteria of testing tools are variant; which makes it difficult for software developer to choose the best appropriate tool for the software in hand. In the introduced framework, a multi-criteria decision-making technique are employed; specifically, AHP and TOPSIS [43], [44]. These two method were selected because their success in decision making within many different fields, especially in business and administration. They are well-known accurate methods, in terms of the

TABLE 1. The criteria of the proposed taxonomy.

Criteria level 1	Criteria level 2	Criteria level 3		References
Technical and Economic View	SW Operating System/Platform	OS	Microsoft	[1, 16, 14, 27, 29, 33,30, 73]
			Unix/ Linux	
			iOS	
			Android	
			………	
		IOT/big Data Platform	Edge Testing	
			Fog Testing	
			Cloud Testing	
		…………		
	Tool Vendor/s	IBM		[6, 12, 28, 30]
		Katalon		
		Neotys		
		Microsoft		
		Micro Focus		
		Parasoft		
		……		
	Tools Acquiring Method/Price	Open Source		[28, 29, 12, 6, 16, 30]
		Commercial		
	Software Technology	Object-Oriented		[21,12, 6, 16, 26, 28, 30, 52, 53, 55, 70]
		Web-Service		
		Network Protocol (TCP)		
		Embedded System		
		Database		
		Mobile Application		
		Web Application		
		Open Source		
		GUI		
		………..		
	Programming Language	Java		[28,12,16, 29, 33, 70]
		C ,C++,C#		
		Php		
		Python		
		SQL		
		HTML		
		……………		
		Test Management		
Testing Process View	Function of Software Testing Tools	Test Design/ Model-based Testing		[7,72, 6,12,16, 33,40, 47,49,58, 65]
		Test Data Generator		
		Test Execution/ Comparators / Test Harness		
		Static Analysis		
		Defect tracking/ Debugging		
		Configuration Management/ Changes Management		
		Performance Testing/Dynamic Analysis/ Monitoring		
		Security Testing		
		Test Evaluation/ Coverage tools		
	Testing Type	Static Testing		[21,4, 33,71]
		Dynamic Testing		
	Software Testing Model/Approach	Agile		[38,36, 4, 19, 35, 16, 31, 21,33, 68]
		Aspect-Oriented		
		V-Model		
		Test-Driven Approach		
		Service-Oriented-Architecture SOA		
		Object-Oriented Approach		
		……….		

TABLE 1. *(Continued.)* **The criteria of the proposed taxonomy.**

	Testing Technique	Black-Box	[6, 36, 25, 35, 4, 21, 18, 31, 33,40 ,71]
		White-Box	
		Gray-Box	
	Testing level	Unit Testing	[61, 6, 35,25,9, 33,38, 26, 21, 18, 31,70]
		Integration Testing	
		System Testing	
		Acceptance Testing	
	Change-Based Testing Type	Conformance Testing	[58,19,21, 35,47,48]
		Regression Testing	
	Software Aspect	Functional Testing	[58, 59, 38,21,33, 35, 65, 66, 67, 60, 19, 41]
		Non-Functional Testing	
Quality Standards View	ISO 9126	Functionality	[60, 66, 67, 19,21,41,47 , 48, 49, 58, 59, 38, 33]
		Portability	
		Maintainability	
	IEEE 1061	Efficiency	
		Usability	
		Reliability	

best decisions they help to make between many alternatives and with many complex criteria [43]–[46].

The following section introduces an overview about AHP and TOPSIS. AHP is used to quantify the weighted importance of different criteria, then TOPSIS is used to rank all appropriate tools and determining the best.

1) AN OVERVIEW OF ANALYTIC HIERARCHY PROCESS (AHP)

Analytic Hierarchy Process (AHP) is based on priority theory; it is used because of its consistency with the multi-layers of taxonomy as it works well with the decisions that have layered criteria. In addition, it provides a way to compute the weights of different criteria accurately using pairwise comparison of the different criteria [43]–[46].

Step 1: Decomposition

In this step, the hierarchy of criteria is done; in this case, the proposed taxonomy will be employed.

Step 2: Weighting the Criteria

In this step, a numerical weight is given for each criterion in the hierarchy by making a pair-wise comparison. This is done as follows:

- Building a single pairwise comparison matrix for the criteria.

- Calculating the Eigenvector of the matrix.

- Repeating step 2 until the difference between successive normalized rows sum is less than a very small value (pre specified). This step can be shortened by summing values in each column of the pairwise comparison matrix and then dividing each element by its column total and then computing the average of elements in each row.

- Calculating and checking the Consistency Ratio (CR).

This is a main step in AHP that reflects the consistency of the pairwise results. For example, if the result of comparison between A and B is 4 and between B with C is 3 so, A compared with C must be $4 \times 3 = 12$. If the result was not 12, some inconsistency might exist and the pairwise comparisons must be revised. Steps of the examination of consistency are as follows:

1) Multiplying pairwise comparison matrix via relative priorities to acquire a weighted sum vector.
2) Dividing weighted sum vector elements via associated priority values.
3) Calculating the average (denoted λmax) of the values from Step 2.
4) Calculating Consistency Index (CI) as shown by Formula 1.

$$\mathrm{CI} = (\lambda\max - \mathrm{n})/(\mathrm{n} - 1) \tag{1}$$

where n is the number of the compared items.

5) Computing Consistency Ratio (CR) as clarified Formula 2:

$$\mathrm{CR} = \mathrm{CI}/\mathrm{RI} \tag{2}$$

where RI is a Random Index.

2) AN OVERVIEW OF TECHNIQUE FOR ORDER PREFERENCE BY SIMILARITY TO IDEAL SOLUTION (TOPSIS)

TOPSIS method is used for ranking all alternatives (tools in this case) by measuring the distance between each alternative in addition to the best and worst solution. The best solution will be the alternative with the shortest distance from the ideal solution, and the farthest from the worst one. TOPSIS produces a high accurate indication/rank in comparison with the other solutions, even with multi-criteria decisions [44]. The output of AHP is the input of TOPSIS, criteria with weight matrix will be the input in this case. The following steps then are achieved [43], [44].

Input to TOPSIS:

A decision matrix $(x_{ij})_{m\times n}$ of m alternatives and n criteria; and a vector $(w_i)_n$ of criteria weights. TOPSIS supposes that there are m options and n criteria and it has the score of each alternative with respect to each attribute e.

Step 1: Constructing normalized decision matrix

This step converts various attribute dimensions to non-dimensional criteria, that allows comparisons across attributes. Matrix $(xij)\ m \times n$ is then normalized to form the matrix $(rij)\ m \times n$ as illustrated by Formula 3:

$$\mathrm{r_{ij}} = \mathrm{x_{ij}}/\surd(\Sigma \mathrm{x_{ij}^2})_\mathrm{i} \quad \text{for } \mathrm{i} = 1, \ldots, \mathrm{m};\ \mathrm{j} = 1, \ldots, \mathrm{n} \tag{3}$$

Step 2: Constructing the weighted normalized decision matrix

- Assume we have a set of weights for each criteria $\mathrm{w_j}$ for $\mathrm{j} = 1, \ldots \mathrm{n}$.
- Multiply each column of the normalized decision matrix by its associated weight. Elements of the new matrix are calculated using Formula 4:

$$\mathrm{v_{ij}} = \mathrm{w_j r_{ij}} \tag{4}$$

Step 3: Determining the ideal and negative ideal solutions

For calculating the best and worst ideal solutions, Formula 5, and Formula 6 are used:

- Ideal solution A*.

$$A^* = \{v_1^*, \ldots, v_n^*\}, \quad \text{where}$$
$$v_j^* = \{\max(v_{ij})_i \text{ if } j \in J; \min(v_{ij}) \text{ if } j \in J'\} \tag{5}$$

- Worst ideal solution A′.

$$A' = \{v_1', \ldots, v_n'\}, \quad \text{where}$$
$$v' = \{\min(v_{ij})_i \text{ if } j \in J; \max(v_{ij}) \text{ if } j \in J'\} \tag{6}$$

where:

$J = \{1, 2, 3, \ldots., n \mid j\}$ is associated with the criteria that have a positive impact, and

$J' = \{1, 2, 3, \ldots., n \mid j\}$ is associated with the criteria that have a negative impact.

Step 4: Calculating the separation measures for each alternative.

This step is calculated using Formula 7 and Formula 8. The separation from the ideal alternative S_i^* is:

$$S_i^* = [\Sigma(v_j^* - v_{ij})^2]^{1/2} \quad i = 1, \ldots, m \tag{7}$$

Similarly, the separation from the negative ideal alternative S_i' is:

$$S_i' = [\Sigma(v_j' - v_{ij})^2]^{1/2} \quad i = 1, \ldots, m \tag{8}$$

Step 5: Calculate the relative closeness to the ideal solution Ci*

This step is achieved using Formula 9:

$$C_i^* = S_i'/(S_i^* + S_i'), \quad 0 < Ci^* < 1 \tag{9}$$

The alternatives/solutions are ranked based on distance from value 1. The best alternative will be the alternative with C_i^* closest to 1.

3) ILLUSTRATIVE EXAMPLE FOR USING AHP AND TOPSIS IN THE FRAMEWORK

Based on the framework, the first step will apply AHP to get the relative weight for each criterion, where the user will be required to determine the criteria according to the proposed taxonomy. Then TOPSIS will follow to rank the different alternatives/solutions starting with the best solution.

The first stage of AHP is to decompose the goal into the criteria that affect the alternative assessment. Users can choose any number of criteria from the proposed taxonomy. All criteria will be chosen from the third level. In our example, the selected criteria are clarified by TABLE 2.

Then pairwise comparison matrix is produced using criteria in testing tools as shown in TABLE 3.

Then, eigen vector is calculated for each row in the matrix and finally the weights of criteria will be as shown by TABLE 4.

Although the number of criteria is large, AHP could calculate the relative weight accurately. The weights are used by TOPSIS to evaluate each alternative in the database. Alternatives will be the different testing tools within the database.

TABLE 2. The chosen criteria in the illustrative example based on the proposed taxonomy.

No.	Criteria
1	Usability
2	Mobile Application
3	Java
4	Dynamic Testing
5	Open Source
6	Waterfall
7	Black-Box Testing
8	Windows

TABLE 3. Pairwise comparison matrix for the different criteria.

	Usability	Mobile Application	Java	Dynamic Testing	Open Source	Waterfall	Black-Box Testing	Windows
Usability	1	1	1	1	1	2	9	1
Mobile Application	1	1	3	2	1/9	2	4	1
Java	1	1/3	1	5	3	4	5	1
Dynamic Testing	1	1/2	1/5	1	2	4	4	1
Open Source	1	9	1/3	1/2	1	3	2	7
Waterfall	1/2	1/2	1/4	1/4	1/3	1	2	1
Black-Box Testing	1/9	1/4	1/5	1/4	1/2	1/2	1	9
Windows	1	1	1	1	1/7	1	1/9	1

TABLE 4. Summary of criteria with weights.

Criteria	Weights
Usability	0.12
Mobile Application	0.14
Java	0.19
Dynamic Testing	0.12
Open Source	0.20
Waterfall	0.05
Black-Box Testing	0.08
Windows	0.09

Now, TOPSIS will use the output of AHP to select the best choice between different alternatives in testing tools database. A testing tools database is created based on the criteria in the proposed taxonomy. When testing tools database is created, each test tool is inserted with its criteria; each tool will have a value for each criterion in third level within the proposed taxonomy.

Therefore, two basic inputs will be used in TOPSIS; the weight matrix which is produced by AHP, and the alternative decision matrix that will be retrieved from the alternatives database. TABLE 5 shows values of different criteria for each testing tools; which are retrieved from the database.

TABLE 5. Criteria for each tool in database (decision matrix for alternatives).

	Usability	Mobile Application	Java	Dynamic Testing	Open Source	Waterfall	Black-Box Testing	Windows
Selenium	0.00	0.00	0.45	0.35	0.58	0.35	0.35	0.35
Test Complete	0.45	0.45	0.00	0.35	0.00	0.35	0.35	0.35
Asml	0.00	0.45	0.45	0.35	0.00	0.35	0.35	0.35
Janova	0.00	0.45	0.45	0.35	0.00	0.35	0.35	0.35
Ranorex	0.45	0.00	0.00	0.35	0.00	0.35	0.35	0.35
Appium	0.45	0.45	0.45	0.35	0.58	0.35	0.35	0.35

TOPSIS method will calculate the positive ideal alternative **A*** and the negative ideal alternative **A**⁻ as described in TABLE 6.

TABLE 6. Positive ideal and negative ideal solutions.

A*	0.06	0.07	0.09	0.04	0.11	0.02	0.03	0.03
A⁻	0.00	0.00	0.00	0.04	0.00	0.02	0.03	0.03

Finally, all alternatives are ranked based on the distance to the positive ideal solution and the negative ideal solution as shown by TABLE 7.

IV. EXPERIMENTS AND EVALUATION

This section articulates experiments and evaluation for the proposed framework. First of all, the taxonomy is validated by academic experts in software engineering, then the selection method is implemented and tested based on different criteria and scenarios.

A. VALIDATION OF THE TAXONOMY OF SOFTWARE TESTING TOOLS

To validate the proposed taxonomy model, the model is validated by a group of SWE academics. These academics have been selected from different universities in Yemen and Saudi Arabia based on their long expertise; the most experienced people in different universities were selected. The validation tool was designed by the researcher and revised by specialists. "Likert" scale is used. The validation tool included sixteen questions, twelve of them were closed questions and four of them were opened questions. The main dimensions- which are covered by the validation tool- are: usefulness, credibility, comprehension, accuracy, logical correlation between the content in the different levels within classes, and finally, usability of the taxonomy model. TABLE 8 Show the basic questions in validation. Figure 4 illustrate a snapshot of the results after answering the specific questions by academic experts.

TABLE 7. The relative closeness to the positive ideal solution.

Tools	Results	Priority
Selenium	0.92	
Test Complete	0.86	
Asml	0.87	
Janova	0.87	
Ranorex	0.83	Worst
Appiunm	1.00	Best

TABLE 8. The basic questions in taxonomy validation tool.

No.	Statement
1	The taxonomy is useful
2	The idea is useful for software industry sector
3	The idea is useful for software engineering research field
4	The proposed taxonomy includes the most important criteria of software testing tools
5	The taxonomy contains logical/reasonable classes/criteria
6	The criteria at the lower level belong to the criteria at the top level correctly
7	The taxonomy is easy to understand
8	The taxonomy is easy to use
9	The taxonomy is scalable
10	The taxonomy contributes to achieve the research objectives
11	The taxonomy can solve some of the problems regarding the lack of using testing tools

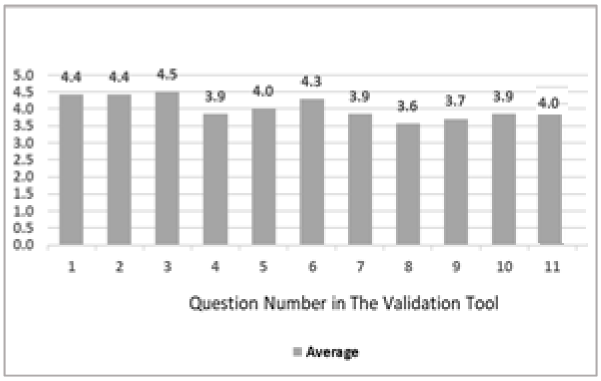

FIGURE 4. The results of validation of the proposed taxonomy (response of academic experts).

As shown by Figure 4, the result was positive. Almost all experts indicated that it was better to simplify the taxonomy more.

B. IMPLEMENTATION AND EVALUATION OF THE PROPOSED SELECTION METHOD

After validating the proposed taxonomy, the selection method is implemented and evaluated. The method is implemented as a website; the idea is to make the framework available for SW industry market. The proposed selection method is evaluated according to predefined criteria using the proposed taxonomy and some common testing tools. Real working systems are selected in the first author university; University of Science and Technology, Yemen. Experiments has been carried out to examine the different test cases. Scenarios are designed based on different parameters. The actual result is compared against predefined expected result. Description of the data which is used in evaluation and the test cases parameters is clarified in the following sections.

1) DESCRIPTION OF DATA

At first, it is important to clarify that there is no a well-known dataset for this type of research. However, a real data is used for the framework evaluation. Data include two categories: (1) Used testing tools; (2) Some real working software. The following section introduces these two categories:

a: USED TESTING CASE TOOLS

Among hundreds of available testing tools, twelve well-known of them were selected to be involved in the framework experiments. The selected tools are the most well-known tools within industrial community. Furthermore, the tools are selected to cover different criteria. TABLE 9 shows the used testing tools with their testing criteria. The empty cells mean that the tool does not support these criteria.

b: USED SYSTEMS

A number of working systems are used to test the proposed framework. These systems are used within the University of Science and Technology (UST). Some of them are part of UST ERP and other dedicated systems. TABLE 10 shows these systems and their characteristics based on the proposed taxonomy. The criteria are carefully prepared by system admins and then reviewed by the managers in the UST IT center.

2) EXPERIMENTS

This section describes the experiments that were conducted to evaluate the framework. At first, the different parameters that were adopted to evaluate the framework were clarified. Then the different scenarios/test cases are designed in the light of the parameters introduced in advance. The evaluation process then was taking place using the data clarified in the last section. Finally, the result is introduced and carefully criticized based on the expected results in the test cases/scenarios.

To examine the performance of the framework, some parameters were taken into consideration when designing the scenarios/test cases. These parameters were chosen in the light of multi-criteria decision-making methods [44]; because the framework is a multi-criteria selection framework. The first factor is the number of tools that are available to test each system; if the number of the available tools is large then the selection becomes more difficult. For the same reason, the second parameter is the number of criteria for each tool. In addition, the weights of the criteria were also one of the adopted parameters. The different weights are important parameters as where the weights are contrasted, the decision become more difficult. Consequently, the different scenarios were designed with different combinations of these. TABLE 11 summarizes the different parameters that form the different experiments scenarios/test cases.

In the light of the parameters in TABLE 11, many scenarios were designed. This section introduces some main different Scenarios/test cases with the expected results to clarify how the evaluating process is conducted.

All tables below show the expected results that is manually estimated by developers, and the results that is produced by selection method. For all scenarios, method could success in choosing the best tool/s with accurate rank [between 0-1] for all tools can be used in each scenario. The best choice will be the tool/s that has/have the higher value as described in Section III and Table 7. value 1 means that this alternative is most close to the ideal solution with 100% satisfaction of the required criteria, in other words, it is the best choice based on the selected criteria.

a: THE FIRST SCENARIO

This scenario aims at testing the framework against the first, second and third parameters with few tools and few software criteria. The results are shown in TABLE 12.

Because of that the scenario parameters are simple, the expected results that estimated manually is similar to extent with actual result that produced by the selection method.

b: THE SECOND SCENARIO

This scenario aims at testing the framework against the first, second and third parameters with many tools and few software criteria. The results are shown by TABLE 13.

TABLE 9. Used testing tools with criteria.

Testing CASE Tools	Testing Criteria Based on the Proposed Taxonomy								
	Function of Software Testing Tools	ISO-9126 Classification	Testing Type	Software testing Model/ Approach	Test Level	Operating System/ Platform	Acquiring Method	SW Tech.	Language
Selenium [16,31])	Test Manag.	Functional	Static	RAD, Waterfall, XP		Windows PC/MAC /UNIX	Open Source	Web App.	Java .Net Perl PHP Python Ruby
QUCIK TEST PROFESS IONAL [28,2,16]						Windows XP	Commercial	Database	VB script Java script
TEST COMPLE TE [28,2,16]						Windows 7, Windows Vista, Windows Server 2008	Commercial	Any App.	VB script, Jscript, Delphi script, C++Script, C# Script
Sauce Labs [28,2]	Test Manag., Test Harness				Unit Testing. Integration Testing	IOS, Windows, Android		Mobile App., GUI	C, C#, C++, Java Script. VB Script Python PHP, OBDC MS SQL Server Java
Ranorex [6,30,50, 51]	Test Manag.				Unit Testing, Integration Testing	IOS Unix Android Linux Mac Windows	Open Source	Mobile App., Web App. Object Oriented Database Service-Oriented	MATLAB, C, C#, C++, .Net., Java Script, VB, PHP, Python OBDC MS SQL Server
Testrail [28,2]	Test Manag.	Usability, Maintainability, Portability, Functionality, Reliability			Unit Testing, Integration Testing				
Zephyr [28,2]	Test Manag.	Usability Maintainability, Portability, Functionality, Reliability,		Agile					
Testpad [28,2])	Test Manag.	Efficiency		Agile, Spiral, Waterfall					
Testtrack [28,2])		Usability, Maintainability, Portability, Functionality, Efficiency, Reliability							
Coverity [28,2])	Test Manag.								
Asml [2]	Test Harness	Reliability	Dynamic	RAD, Waterfall				GUI	
Appuinm [28, 51]	Test Harness. Test Manag.		Static			IOS Android		Mobile App., GUI	

TABLE 10. Used software with criteria.

	Software	Testing Criteria Based on the Proposed Taxonomy							
		Programing Language	Software Technique	Operating System/ Platform	Testing Level	Quality Assurance Requirements	Testing Technique	Tool Acquiring Method	Software testing Model /Approach
1	Accounting System UST	Oracle, SQL, PHP	Database, Web App.,	Win, Unix	Unit, Integration Testing	Security, Reliability, Usability, Efficiency, Functionality	Black-Box White-Box	Free, Rent	Waterfall
2	Human Resources System UST	Oracle, SQL, Java, PHP	Database, Web App., Mobile App.	Win, Unix, Android, iOS	Unit, Integration Testing	Functionality Security, Reliability, Usability	Black-Box White-Box	Free, Rent, Bough	Waterfall, Agile, Incremental
3	Open Education System UST	Oracle, SQL, Java, PHP	Database, Web App., Mobile App.	Win, Android, iOS	Unit, Integration Testing	Functionality , Reliability, Usability	Black-Box White-Box	Free, Rent	Agile, XP
4	Registration System UST	Oracle, SQL, PHP	Databases , Web App.	Win, Unix	Unit, Integration Testing	Functionality , Security, Reliability, Usability	Black-Box White-Box	Free, Rent	Waterfall
5	Balanced Scorecard System UST	PHP, SQL, Java Script	Database, Web App.	Win	Unit, Integration Testing	Functionality , Usability	Black-Box White-Box	Free, Rent	Agile
6	Service and Maintenance System UST	Oracle, SQL, Java Php	Database, Web App., Mobile App.	Win, Android, iOS	Unit, Integration, System Testing	Functionality , Usability	Black-Box White-Box	Free, Rent	Agile, Incremental
7	Journals Editing System UST	PHP, SQL, Java Script	Database, Web App.	Win	Unit, Integration Testing	Functionality , Reliability, Usability	Black-Box White-Box	Free, Rent	Agile, Incremental
8	University of Science and Technology Website	PHP, SQL, Java Script	Database, Web App.	Win	Unit, Integration Testing	Functionality , Usability	Black-Box White-Box	Free, Rent	Waterfall, Agile
9	Learning Management System (CLMS) UST	PHP, SQL, Java Script	Database, Web App.	Win	Unit, Integration, System Testing	Functionality , Usability	Black-Box	Free	Agile, Incremental

TABLE 11. A summary of the different parameters.

No.	Parameter
1	Number of Available Tools for each System/Software
2	Number of Criteria for each System/Software
3	Weights of the Different Criteria for each System/Software

As shown by TABLE 13, even where tools are many (more than 10), the selection method produced distinguishable results. At the first and third case, the manual estimation was difficult due to the high number of potential tools.

TABLE 12. The first scenario.

System	Testing Criteria	Scenario /test Case Parameters	Result	
			Expected Results	**Actual Results**
Journals Editing System	Open Source, Test Manag.	Few tools, Few criteria, Variant weights	Ranorex	**Ranorex=1** Sauce Labs=0

c: THE THIRD SCENARIO

This scenario aims at testing many tools, few software criteria, and close weights for the same criteria. The results are shown by TABLE 14.

TABLE 13. The second scenario.

System	Testing Criteria	Scenario /test Case Parameter	Result	
			Expected Results	Actual Results
Learning Manag. System (LMS)	Usability ,Web App.	Many tools, few criteria, variant weights	Zephyr TestPAD	**Testrail=1 Zephyr=1** Seleniu=0 QT=0 Sauce labs=0 Coverity=0 TestPAD=0 Ranorex=0 TestTrack=0 Appuinm=0 Test Complete=0
University of Science and Technology Website	Open Source, PHP, Windows		Ranorex	**Ranorex =1** Asml=0 Test Complete=0 QT=0 Appuinm=0, Sauce labs=0 Coverity=0 Testrail=0 Zephyr=0 TestPAD=0 Ranorex=0 TestTrack=0 Selenium=0
Balanced Scorecard System	Agile, Java, Usability		Testrail TestPAD Test-Track	**TestPAD=1 TestTrack=1** Selenium=0 asml=0 Test Complete=0 QT=0 Appuinm=0 Sauce labs=0 Coverity=0 Testrail=0 Zephyr=0 Ranorex=0

TABLE 14. The third scenario.

System	Testing Criteria	Scenario /test Case Parameter	Result	
			Expected Results	Actual Results
Open Education System UST	Agile, Oracle, Usability	Many tools, Few criteria, **Close weight values for the criteria**	TestTrack Zephyr Testrail TestPAD	**TestTrack=1** Zephyr=0.87 Testrail= 0.49 TestPAD= 0.49 Selenium=0 Asml=0 Test Complete=0 QT=0 Appuinm=0 Sauce labs=0 Coverity=0 Ranorex=0

TABLE 15. The fourth scenario.

System	Testing Criteria	Scenario /test Case Parameter	Result	
			Expected Results	Actual Results
Learning manag. system (LMS)	Usability, Dynamic Testing, Unit testing, integration testing, Database, Web App. Agile, Oracle, Windows	Many tools, many criteria, convergence weights for criteria	Can not be estimated	**TestPAD=1** Zephyr=0.83 TestTrac= 0.83 Sauce labs=0.55 Ranorex= 0.55 selenium=0 Asml=0 QT=0 Coverity=0 TestPAD=0

As shown by Table 14, even with this difficult scenario, where criteria are close in weights, the results are distinguishable. The first and the second tools (TestTrack, Zephyr) are clearly more appropriate for the investigated system, while the former ones (Testrail, TestPAD) are not good choice. The manual estimation make all four selected tools are equal in rank for the system in hand.

d: THE FOURTH SCENARIO

This scenario assumes a confusable case with many tools and many criteria. In this case, a manual decision is difficult and even impossible. TABLE 15 shows how easily the framework can make an accurate decision.

A summary of results is in the following:

1. The selection method has achieved a good success in selecting the appropriate testing tools listed from the best to the worst. As shown by table 16, our method success to select the best choice at all different scenarios.

TABLE 16. Summary of results of all scenarios.

Scenario	**Expected Results** (manual estimation by developers)	**Actual Result** (results of selection method)
1	Ranorex	**Ranorex=1 (the best)** Sauce Labs=0

2	Zephyr,
	TestPAD,
		Testrail=1 (the best)
		Zephyr=1 (the best)
		Selenium=0
		QT=0
		Sauce labs=0
		Coverity=0
		TestPAD=0
		Ranorex=0
		TestTrack=0
		Appuinm=0
		Test Complete=0

	Ranorex	**Ranorex =1 (the best)**
		Asml=0
		Test Complete=0 QT=0
		Appuinm=0
		Sauce labs=0 Coverity=0
		Testrail=0
		Zephyr=0
		TestPAD=0
		Ranorex=0 TestTrack=0
		Selenium=0

	Testrail,
	TestPAD,
	TestTrack
		TestPAD=1 (the best)
		TestTrack=1 (the best)
		Selenium=0
		asml=0
		Test Complete=0
		QT=0
		Appuinm=0
		Sauce labs=0 Coverity=0
		Testrail=0
		Zephyr=0
		Ranorex=0

3	TestTrack,,
	Zephyr,
	Testrail,
	TestPAD
		TestTrack=1 (the best)
		Zephyr=0. 87
		Testrail=0.49
		TestPAD=0.49
		Selenium=0
		Asml=0
		Test Complete=0 QT=0
		Appuinm=0
		Sauce labs=0 Coverity=0
		Ranorex=0

4	(difficult to estimate manually, many criteria with close weights, many tools)	**TestPAD =1 (the best)**
		Zephyr=0.83
		TestTrac=0.83
		Sauce labs=0.55
		Ranorex=0.55
		selenium=0
		Asml=0
		QT=0
		Coverity=0
		TestPAD=0

TABLE 16. *(Continued.)* Summary of results of all scenarios.

2. The results and calculations still accurate even with complicated cases. Accuracy here means that the best choice still gets the high rank value even there are many criteria and many alternative tools.
3. As a consequence of 2, the framework is flexible; any number of tools and criteria can be added and used.

Finally, as shown by results, the different parts of the proposed framework are well evaluated. The proposed taxonomy is well validated by experts in terms of usefulness, effectiveness, and importance. The selection method performs good results in all different scenarios by determining the appropriate testing tools with accurate rank.

V. CONCLUSION & FUTURE WORKS

This research introduces a framework for selecting testing tools. The proposed framework includes a comprehensive taxonomy of testing tools, and a selection method for developers to use the taxonomy for selecting appropriate testing tools.

Based on findings, the proposed framework was well evaluated, the taxonomy of the testing tools and also the selection method. For the first part of the framework- testing tools taxonomy- the experts indicated that the taxonomy is/will: (1) important and useful; (2) comprehensive; (3) help developers choosing the best tools for software testing among a large number of testing tools. However, most experts indicated that the taxonomy is complicated a little bit and this could make it difficult to use.

The second part of the proposed framework was successful based on the experiments that have been conducted. The evaluation is designed and implemented carefully using different parameters and scenarios. The proposed selection method achieved good results even with complicated cases. For all scenarios, method can determine the appropriate testing tools with accurate rank. Thanks go to using AHP with TOPSIS as a hybrid multi-criteria selection method. The results were accurate and the method always chooses the best tool with high rank and ranks all tools accurately from the best tool to the worst.

Using the proposed framework would help developers to choose the best testing tools for their software based on the criteria of both tools and the software. It reduces the cost of selection process and helps the beginners in testing to choose the testing tools based on scientific and comprehensive criteria.

For future work, the taxonomy can be simplified. A wide validation for the taxonomy by developers and academics is recommended. An experimental study is recommended as it can provide an accurate image about the proposed taxonomy.

REFERENCES

[1] B. B. Agarwal, S. P. Tayal, and M. Gupta, *Software Engineering and Testing*. Boston, MA, USA: Jones & Bartlett, 2010.

[2] M. Barnett, W. Grieskamp, W. Schulte, N. Tillmann, and M. Veanes, "Validating use-cases with the AsmL test tool," in *Proc. 3rd Int. Conf. Quality Softw. (QSIC)*, Dallas, TX, USA, Nov. 2003, pp. 238–246.

[3] S. Bhuvana and M. V. Srinath, "A survey on automated combinatorial testing for software tool (ACTS) with experimental revise based on T-way test generation," *Int. J. Comput. Appl.*, vol. 6, no. 3, pp. 13–20, 2016.

[4] R. Bryce and R. Kuhn, "Software testing [guest editors' introduction]," *Computer*, vol. 47, no. 2, pp. 21–22, Feb. 2014.

[5] I. Burnstein, *Practical Software Testing: A Process-Oriented Approach*. Springer, 2014.

[6] R. K. Chauhan and I. Singh, "Latest research and development on software testing techniques and tools," *Int. J. Current Eng. Technol.*, vol. 4, no. 4, pp. 2368–2372, 2014.

[7] G. Coleman and M. Walsh, *Agile Testing: An ISTQB-BCS Foundation Guide*. Swindon, U.K.: BCS Learning & Development Limited, 2017.

[8] E. Dustin, J. Rashka, and J. Paul, *Automated Software Testing: Introduction, Management, and Performance*. Boston, MA, USA: Addison-Wesley, 1999.

[9] M. M. Eslamimehr, "The survey of model based testing and industrial tools," M.S. thesis, Linköping Univ., Linköping, Sweden, 2008.

[10] T. Garrepalli, "Knowledge Management in software testing," M.S. thesis, Blekinge Inst. Technol., Karlskrona, Sweden, 2015.

[11] J. Iivari, "Why are CASE tools not used?" *Commun. ACM*, vol. 39, no. 10, pp. 94–103, Oct. 1996.

[12] S. Jain, V. Srivastava, and P. Katiyar, "Integration of metric tools for software testing," *Int. J. Enhanced Res. Sci. Technol. Eng.*, vol. 2319, no. 7463, pp. 445–447, 2014.

[13] S. Jarzabek and R. Huang, "The case for user-centered CASE tools," *Commun. ACM*, vol. 41, no. 8, pp. 93–99, Aug. 1998.

[14] H. Javed, N. M. Minhas, A. Abbas, and F. M. Riaz, "Model based testing for web applications: A literature survey presented," *J. Softw.*, vol. 11, no. 5, pp. 347–361, 2016.

[15] S. H. Kan, *Metrics and Models in Software Quality Engineering*, 2nd ed. Boston, MA, USA: Addison-Wesley, 2002.

[16] H. Kaur and G. Gupta, "Comparative study of automated testing tools: Selenium, quick test professional and test complete," *J. Eng. Res. Appl.*, vol. 3, no. 5, pp. 1739–1743, 2013.

[17] C. Klammer and R. Ramler, "A journey from manual testing to automated test generation in an industry project," in *Proc. IEEE Int. Conf. Softw. Quality, Rel. Secur. Companion (QRS-C)*, Prague, Czech Republic, Jul. 2017, pp. 591–592.

[18] B. B. Konka, "A case study on software testing methods and tools," M.S. thesis, Gothenburg Univ., Gothenburg, Sweden, 2012.

[19] E. Koppel, "Software test management tool evaluation framework," M.S. thesis, Univ. Tartu, Tartu, Estonia, 2012.

[20] C. Lassenius, T. Soininen, and J. Vanhanen, "Constructive research," in *Proc. Methodol. Workshop*, Espoo, Finland: Helsinki Univ. of Technology, 2001.

[21] W. Lewis, *Software Testing and Continuous Quality Improvement*, 2nd ed. Boca Raton, FL, USA: CRC Press, 2005.

[22] G. Lopez and A. Martinez, "Use of Microsoft testing tools to teach software testing: An experience report," in *Proc. 1st Annu. Conf. Expo.* Indianapolis, IN, USA: American Society for Engineering Education, Jun. 2014, pp. 1310–1324.

[23] V. Maheshwari and M. Prasanna, "Generation of test case using automation in software systems—A review," *Indian J. Sci. Technol.*, vol. 8, no. 35, pp. 1–9, Dec. 2015.

[24] S. Masuda, "Software testing design techniques used in automated vehicle simulations," in *Proc. IEEE Int. Conf. Softw. Test., Verification Validation Workshops (ICSTW)*, Tokyo, Japan, Mar. 2017, pp. 300–303.

[25] P. Pohjolainen, "Software testing tools," M.S. thesis, Univ. Kuopio, Kuopio, Finland, 2002.

[26] L. Rajamanickam, "Testing tool for object oriented software," *Int. J. Sci. Res. Manage.*, vol. 2, no. 8, pp. 1205–1208, 2014.

[27] R. Rattan, "Comparative study of automation testing tools: Quick test pro and selenium," *VSRD Int. J. Comput. Sci. Inf. Technol.*, vol. 3, no. 6, 2013.

[28] G. Shah, P. Shah, and R. Muchhala, "Software testing automation using Appium," *Int. J. Current Eng. Technol.*, vol. 4, no. 5, pp. 3528–3531, 2014.

[29] K. Shaukat, U. Shaukat, F. Feroz, S. Kayani, and A. Akbar, "Taxonomy of automated software testing tools," *Int. J. Comput. Sci. Innov.*, vol. 2015, no. 1, pp. 7–18, 2015.

[30] J. Singh and M. Sharma, "Comprehensive review of web-based automation testing tools," *Int. J. Innov. Res. Comput. Commun. Eng.*, vol. 3, no. 10, pp. 9255–9259, 2015.

[31] K. Sneha and G. M. Malle, "Research on software testing techniques and software automation testing tools," in *Proc. Int. Conf. Energy, Commun., Data Anal. Soft Comput. (ICECDS)*, Chennai, India, Aug. 2017, pp. 77–81.

[32] Software Testing Fundamentals. (Oct. 11, 2016). *Definition of Test*. [Online]. Available: http://softwaretestingfundamentals.com/definition-of-test/

[33] M. Tiitinen, "Key factors for selecting software testing tools," M.S. thesis, Metropolia Univ. Appl. Sci., Helsinki, Finland, 2013.

[34] T. Toroi, "Testing component-based systems: Towards conformance testing and better interoperability," Ph.D. dissertation, Univ. Kuopio, Kuopio, Finland, 2009.

[35] S. Uspenskiy, "A survey and classification of software testing tools," M.S. thesis, Lappeenranta Univ. Technol., Lappeenranta, Finland, 2010.

[36] V. Vaishnavi, W. Kuechler, and S. Petter. (Mar. 21, 2017). *Design Science Research in Information Systems*. [Online]. Available: http://www.desrist.org/design-research-in-information-systems/

[37] K. Valliammai and P. Sujatha, "Analysis of efficiency of automated software testing," *Methods, Direction Res., Int. J. Sci. Res.*, vol. 5, no. 12, pp. 34–38, 2015.

[38] T. Wala and A. K. Sharma, "Improvised software testing tool," *Int. J. Comput. Sci. Mobile Comput.*, vol. 3, no. 9, pp. 573–581, 2014.

[39] H. V. Gamido and M. V. Gamido, "Comparative review of the features of automated software testing tools," *Int. J. Electr. Comput. Eng.*, vol. 9, no. 5, pp. 4473–4478, Oct. 2019.

[40] R. Isha, G. Pooja, and M. Himani, "A review of tools and techniques used in software testing," *J. Emerg. Technol. Innov. Res.*, vol. 6, no. 4, pp. 262–266, 2019.

[41] D. Mohammad and I. Mohammad, "How software testing impacts the quality of software systems?" *Int. J. Eng. Comput. Sci.*, vol. 1, no. 2, pp. 5–9, 2019.

[42] A. Shamsu, A. Zakari, H. Abdu, A. Nura, M. A. Zayyad, S. Suleiman, A. Adamu, and A. S. Mashasha, "Software testing: Review on tools, techniques and challenges," *Int. J. Adv. Res. Technol. Innov.*, vol. 2, no. 2, pp. 11–18, 2020.

[43] R. V. Rao, *Decision Making in the Manufacturing Environment: Using Graph Theory and Fuzzy Multiple Attribute Decision Making Methods*. Berlin, Germany: Springer, 2007.

[44] E. Triantaphyllou, *Multi-Criteria Decision Making Methods: A Comparative Study*. Boston, MA, USA: Springer, 2000.

[45] M. Büyükyazıcı and M. Sucu, "The analytic hierarchy and analytic network processes," *Hacettepe J. Math. Statist.*, vol. 32, no. 1, pp. 65–73, 2003.

[46] T. L. Saaty, "How to make a decision: The analytic hierarchy process," *Eur. J. Oper. Res.*, vol. 48, no. 1, pp. 9–26, 1990.

[47] *IEEE Guide to the Software Engineering Body of Knowledge (SWEBOK), Version 3.0*, IEEE, Piscataway, NJ, USA, 2014.

[48] *Guide to the Software Engineering Body of Knowledge (SWEBOK)*, Standard ISO/IEC TR 19759:2004, 2004.

[49] G. Saini and K. Rai, "Software testing techniques for test cases generation," *Int. J. Adv. Res. Comput. Sci. Softw. Eng.*, vol. 3, no. 9, pp. 261–265 2013.

[50] F. Okezie, I. Odun-Ayo, and S. Bogle, "A critical analysis of software testing tools," *J. Phys., Conf. Ser.*, vol. 1378, Dec. 2019, Art. no. 042030.

[51] S. Singh. *10 Top Testing Automation Tools for Software Testing Tools*. https://www.netsolutions.com/insights/top-10-automation-testing-tools/
[52] D. R. Mohammad, S. Al-Momani, Y. M. Tashtoush, and M. Alsmirat, "A comparative analysis of quality assurance automated testing tools for Windows mobile applications," in *Proc. IEEE 9th Annu. Comput. Commun. Workshop Conf. (CCWC)*, Jan. 2019, pp. 414–419.
[53] H. Anjum, M. Imran, M. Jehanzeb, M. Khan, S. Chaudhry, S. Sultana, Z. Shahid, F. Zeshan, and S. Nazir, "A comparative analysis of quality assurance of mobile applications using automated testing tools," *Int. J. Adv. Comput. Sci. Appl.*, vol. 8, no. 7, pp. 249–255, 2017.
[54] P. Srinivas. *What is OpenScript, Testing Tools*. Accessed: May 24, 2019. [Online]. Available: http://www.testingtools.co/oats/what-is-openscript
[55] K. Holl and F. Elberzhager, "Mobile application quality assurance," in *Advances in Computers*. Amsterdam, The Netherlands: Elsevier, 2018.
[56] S. Zein, N. Salleh, and J. Grundy, "A systematic mapping study of mobile application testing techniques," *J. Syst. Softw.*, vol. 117, pp. 334–356, Jul. 2016.
[57] M. S. Iqbal, M. Sadiq, A. Rehman, and T. Khan, "Framework for the development of automated inspection tools," *Int. J. Comput. Sci. Netw. Secur.*, vol. 16, no. 1, pp. 16–26, 2016.
[58] *Foundation Level Syllabus, Version 3.1*, ISTQB, Brussels, Belgium, 2018.
[59] *Software Engineering—Product Quality*, Standard ISO/IEC 9126, ISO/IEC 25022:2016, Oct. 1988.
[60] *Software Quality Metrics Methodology*, IEEE Standard 1061-1992, doi: 10.1109/IEEESTD.1993.115124.
[61] M. Fewster and D. Graham, *Software Test Automation: Effective Use of Test Execution Tools*. Great Britain, U.K.: Pearson, 1999.
[62] O. Tipale, "Observations on software testing practice," M.S. thesis, Lappeenranta Univ. Technol., Lappeenranta, Finland, 2007.
[63] R. M. Parizi, A. A. A. Ghani, R. Abdullah, and R. Atan, "Empirical evaluation of the fault detection effectiveness and test effort efficiency of the automated AOP testing approaches," *Inf. Softw. Technol.*, vol. 53, no. 10, pp. 1062–1083, Oct. 2011.
[64] D. R. Graham, "Software testing tools: A new classification scheme," *Softw. Test., Verification Rel.*, vol. 1, no. 3, pp. 17–34, Oct. 1991.
[65] D. Graham *et al.*, *Foundations of Software Testing: ISTQB Certification*, 4th ed. 2019.
[66] *Recommended Practice for the Evaluation and Selection of CASE Tools*, IEEE Standard 1209-1992.
[67] *Recommended Practice for the Adoption of Computer-Aided Software Engineering (CASE) Tools*, IEEE Standard 1348-1995.
[68] M. A. Jamil, M. Arif, N. S. A. Abubakar, and A. Ahmad, "Software testing techniques: A literature review," in *Proc. Int. Conf. Inf. Commun. Technol. Muslim World*, Nov. 2016, pp. 177–182.
[69] J. Kasurinen, O. Taipale, and K. Smolander, "Software test automation in practice: Empirical observations," *Adv. Softw. Eng.*, vol. 2010, pp. 1–18, Feb. 2010, doi: 10.1155/2010/620836.
[70] K. M. Mustafa, R. E. Al-Qutaish, and M. I. Muhairat, "Classification of software testing tools based on the software testing methods," in *Proc. 2nd Int. Conf. Comput. Electr. Eng.*, 2009, pp. 229–233.
[71] S. Nidhra and J. Dondeti, "Black box and white box testing techniques—A literature review," *Int. J. Embedded Syst. Appl.*, vol. 2, no. 2, pp. 29–50, Jun. 2012.
[72] I. Sommerville, "Software engineering," in *Software Tool Classification*, 10th ed. 2015.
[73] J. P. Dias, F. Couto, A. C. R. Paiva, and H. S. Ferreira, "A brief overview of existing tools for testing the Internet-of-Things," in *Proc. IEEE Int. Conf. Softw. Test., Verification Validation Workshops*, Apr. 2018, pp. 104–109.