Tampere University of Applied Sciences

Software Testing Tools

Ozan Topcu

March 2024

Bachelor's Degree Programme in Software Engineering

CONTENTS

1 INTRODUCTION

Software testing is an indispensable aspect of the software development lifecycle, ensuring the quality, reliability, and performance of software products. In today's rapidly evolving technological landscape, the use of testing tools has become increasingly prevalent, offering organizations efficient and effective means to validate their software systems. This report aims to provide a comprehensive overview of software testing tools, exploring their significance, types, automation methods, and benefits. By delving into these topics, we seek to equip readers with valuable insights into the role of testing tools in modern software development practices.

2 Why Should We Use Testing Tools?

Why are testing tools not just important to remember but a crucial part of testing for a test analyst and a software developer? Testing tools help the organization streamline the process, rather than focusing on a single part of the process. This might feel unnecessary and tedious initially; however, in the long run, a project that utilizes its testing tools efficiently and at the right time saves much more time.

Think of it like this: an employee might feel that entering a defect in a tracking system might cause them to lose the time that they could have used to focus on future products; instead, they might just prefer to let the developer know immediately through unrecorded channels like sending an email, text message, or even just mentioning to them face-to-face. Indeed, this might feel like it actually, saves the time of that specific employee but not recording a defect would cause the whole system to miss out on gathering metrics from it. Later on, this would decrease the accuracy and accountability of the report that gets done.

In addition, not using the tool to record that issue might cause that defect to even be overlooked; the developer might forget the conversation, the message might get lost, or the single email might get drowned under maybe hundreds of other issues. That’s why; even though utilizing the tools might seem like a great hassle, it saves a considerable amount of effort, time, and resources in the future.

Throughout the project, one of the main factors a project manager or even software testing engineer needs to keep in mind that the amount to which they can utilize a certain tool.

> “Tools often require effort up front to realize gains later in the process. In some cases, as in test execution automation tools, the development costs for the scripts are much higher than the benefit that is realized the first time the scripts are run. It may take several executions before the return on the investment justifies the cost.” (Bath & McKay, 2014, 13.2)

> “As a rule, not all available test tools are applied in a project. However, the test manager should know available tool types to be able to decide if and when to use a tool efficiently in a project.” (Spillner, Rossner, Winter & Linz, 2006, 17)

The return on the investment mentioned above might never justify the cost as well, in those cases test manager decides to shift to a different testing tool if plausible or let go of that testing practice altogether.

3 Types of Tools Used in Testing

Testing tools as a concept bears a fundamental categorization for each section of effective software testing. These tools are indispensable parts that in the end provide a streamline of software testing, they encompass manual aids, automation suites (which will be talked about in another chapter later on), design utilities, data manipulation tools, and execution platforms.

3.1 Test Design Tools

The whole idea of test design tools at its core is to provide test cases and scenarios based on various criteria such as requirements, use cases, and business processes. To achieve this, they work with particular requirements tools and receive these project requirements in a specific format, such as Unified Modelling Language (UML). In the case of the requirements management tool not having its own test design tool, there are still plenty of more options. For example, using

a classification tree tool to generate the lists of combinations we need to test has a great impact when building these test cases. In today's world few of the most common design tools include Tricentis Tosca, Cucumber, and SpecFlow

3.2 Data Tools

Data tools facilitate the management and manipulation of test data used in testing processes. Test data management is crucial for ensuring that test cases cover a wide range of scenarios and conditions. Data tools enable testers to generate synthetic data, anonymize sensitive information, and maintain data consistency across test environments. Additionally, these tools often integrate with test automation frameworks to streamline data-driven testing.

The problem with using data generation and mining tools is they are usually large sets of test data created with no identification of the expected outcomes. On top of that you might still need to write the test cases that will utilize this data. As aforementioned anonymization of data is useful in cases where data preparation tools that take part in this process remove information such as credit card numbers, addresses, and private information like that so that a human testing team can go through this large set of data with more confidence that there is no risk of compromising personal information. Another useful data tool might be the data creation tools that produce brand new data from a few given parameters, it can list a whole set of data from just developer-set parameters. Popular data tools include Informatica, Talend, and Delphix.

As mentioned by Andreas Spillner et al. (2006) Data generation tools can be grouped into 4 different categories:

- Database-based test data generators that create test data on the basis of database schemas or database content.
- Code-based test data generators that analyze the source code to derive the test data. Target or expected values, however, cannot be derived.
- Interface-based test data generators that derive test data through the identification of interface parameter domains.
- Specification-based test data generators that derive test data and associated target values from a formal specification.

3.3 Test Execution Tools

Test execution tools, as it states in its name, are responsible for executing test cases, recording these executions, and reporting these executions back to developers. Although mainly they are automated there are cases where manual tests are performed as well. Moreover, these automated scripts are quite often based on manual test cases that have been developed for functional testing.

The overall structure of test execution tools often integrates test management systems to retrieve test cases and store test results centrally. They also offer features for scheduling test runs, parallel execution, and distributed testing across multiple environments. Examples of test execution tools include HP Unified Functional Testing (UFT), Ranorex Studio, and Katalon Studio.

3.4 Static Test Tools

Static Test tools play a crucial role in analyzing software artifacts, such as code, documentation, and design specifications, without executing the code. These tools help identify defects, inconsistencies, and potential issues early in the development lifecycle, contributing to improved software quality and reduced maintenance efforts. Common types of Static Test tools include:

- Static Code Analysis Tools: These tools analyze source code to identify potential defects, security vulnerabilities, coding standards violations, and performance bottlenecks. By detecting issues such as code smells, unused variables, and improper coding practices, static code analysis tools aid in improving code quality and maintainability. Examples include SonarQube, Checkstyle, and PMD (Programming Mistake Detector).

- Code Review Tools: Code review tools facilitate collaborative code reviews among development teams, allowing team members to provide

feedback, suggestions, and comments on code changes. These tools promote code quality, knowledge sharing, and adherence to coding standards. Examples include GitHub, GitLab, and Bitbucket.

- Static Security Analysis Tools: Static security analysis tools focus on identifying security vulnerabilities and weaknesses in software systems, such as injection attacks, cross-site scripting (XSS), and authentication flaws. These tools help organizations fortify their applications against potential cyber threats and comply with security standards and regulations. Examples include Veracode, Fortify, and Coverity.

3.5 Dynamic Test Tools

Dynamic Test tools, unlike static testing tools, involve the execution of software components or systems to evaluate their behavior and performance. These tools simulate real-world usage scenarios, interactions, and inputs to validate software functionality and detect defects. Common types of Dynamic Test tools include:

- Functional Testing Tools: Functional testing tools verify that software functions correctly according to specified requirements. They automate the execution of test cases and validate software behavior against expected outcomes. Examples include Selenium, QTP (QuickTest Professional), and TestComplete.

- Performance Testing Tools: Performance testing tools assess the performance characteristics of software, including speed, scalability, and stability under varying load conditions. They simulate user traffic, transactions, and system resources to identify performance bottlenecks and optimize software performance. Examples include JMeter, LoadRunner, and Apache Benchmark.

- Security Testing Tools: Security testing tools identify vulnerabilities and weaknesses in software systems, ensuring that they remain secure against potential threats. They detect security vulnerabilities such as SQL

injection, cross-site scripting (XSS), and insecure authentication mechanisms. Examples include OWASP ZAP, Burp Suite, and Nessus.

4 Automation in Testing

Automation is one of the most crucial parts of software testing, in a way it connects the whole DevOps with its culture and practices it brings to the topic of testing. As CI/CD pipelines are expected to be automated processes not having testing automated would be a large setback.

4.1 Why Automation

When tools tailored to automation are efficiently used they can provide the following benefits:

- Reduce the cost of repeated executions of the same tests, in situations like regression tests that would be required to run multiple times for a single release. When that testing or group of tests is not just a part of a niche development but part of a whole mainstream workflow there might be cases where hundreds of developers might be updating the code base, in that case, imagine the loss of overall time just being spent on single testing.
- In addition to the aforementioned, it introduces repeatability into sets of tests and test executions since automation software will always run the same tests in the same way whereas we humans tend to vary what we do.
- The ability to test every tiny detail of software with large sets of data that would be otherwise impossible to test with using only manual testing.

The benefits of automation usually outweigh its drawbacks by a mile, however as a testing manager you must know what the right scenarios are to prefer manual testing over automation. At the end of the day, automation can produce a large bill and will bring maintenance requirements with itself.

4.2 When to Automate

During integration of automation into testing it is important to keep in mind the significant benefits it brings in terms of efficiency and reliability, but it is also important to keep in mind the strategical allocation of resources to areas where automation yields the highest return on investment. The most significant and initial consideration to implement automation is the presence of tests that require repeated execution with minimal changes. Tests like regression suites, are great examples in these cases where it offers stable ground for automation implementation. With the automation of regression tests, which typically involve validating existing functionalities, teams can achieve significant time savings and ensure consistent quality across software releases.

Similarly, smoke tests, also known as build verification tests, have great potential to be considered for automation. However, it is known to require more care and maintenance due to the evolving features and changes, automating smoke tests provides an efficient means to verify build integrity, especially in environments where numerous and regular builds are the norm. Automating build verification tests in continuous integration environments not only saves time for testers but also enhances collaboration between testers and developers by swiftly identifying build issues early in the development cycle.

Beyond regression and smoke testing, automation finds utility in integration testing, system testing, and system integration testing phases. Automating tests at the API level, followed by component and integration tests, proves beneficial in ensuring seamless interaction between software components. However, it's essential to consider the stability of the software when deciding the timing of automation implementation. While automation is most effective when applied to relatively stable software, starting early in the development process allows for prolonged utilization and maximizes benefits over time.

Despite the advantages of automation, it's vital to recognize scenarios where automation may not be the most cost-effective solution. Investing in automation for unstable or rapidly evolving software can lead to increased maintenance costs

and diminished returns. Therefore, a balanced approach, focusing initially on regression and smoke tests before gradually expanding automation coverage, is advisable. By prioritizing automation efforts based on the cost-benefit trade-off, organizations can optimize testing processes, enhance software quality, and ultimately deliver value to stakeholders.

4.3 How to Automate

In this section we will be talking about one of the most common and simple automated test implementation (Capture/Playback), data-driven automation implementation, keyword-driven automation implementation.

4.3.1 Capture/Playback

Capture/Playback is a low-code solution that helps teams automate their testing. It is done by using a record feature of a testing tool to generate testing scripts, the developer or anybody in reality can initially perform manual actions on an non-production application these testing tools will capture the actions and automatically turn them into test scripts.

The benefit of Capture/Playback testing is one might record many testing scripts and can run these tests every time a new feature is introduced to the codebase. That’s where the “Playback” side of this test comes into action, due to this it can be used efficiently in regression testing specifically. One other benefit of this method is that it is, as mentioned before, a low-code method that is particularly useful for teams where programming skills are more in the background, it allows many to start and use with just a few simple tutorials.

Figure 4-1 The Marathon login dialog (Bath & McKay, 2014, 13.3.6)

```
Login.User_Name.Enter("HJones")
Login.Password.Enter("wdft56&st")
Login.OK.Press
```

Here an example of capture/playback example can be seen, this way we can understand what's actually not so helpful in testing. First of all, the data in here is hard coded whatever the person recording enters will be the data to be used every time this test is being played. Additionally, this testing method is prone to break every time a GUI update is done the test might be unusable. For example, the change of the button "OK" to "Login" would be enough for the mentioned issue.

4.3.2 Data-Driven Automation

These automation techniques allow the developer to reduce testing costs by allowing them to create more realistic and more possibility-rich test scenarios using test scripts developed by test analysts. As expected, it is done by a person who has experience in this kind of testing and testing in general with some level of programming skills. In this method, automation responsible, can and will use the data-generation tools to create testing scenarios.

The initial part of data-driven automation is the storage of large sets of data that will be later on used for testing, these storage units are mostly databases that hold the information in tables or files, then the following part is to implement an automation that cycles through these databases to run the tests and yield results in a much faster manner. With data-driven automation, each result of a test can be validated with either another single test result or by being compared to a larger group of results to detect anomalies or outright wrong results.

4.3.3 Keyword-Driven Automation

Keyword-driven automation is a testing technique that focuses on the use of keywords or action words to define test steps and interactions with the application under test (AUT). Instead of scripting tests using traditional programming languages, test analysts create test cases using a set of predefined keywords, each representing a specific action or assertion. The great advantage that comes with this method again is that it enables testers and developers to focus on the test design and execution rather than focusing on their programming expertise and the efficiency of their code.

The implementation of keyword-driven automation is achieved through linking keywords to scripts that will be executed by the automation, each keyword may be a common action or business process that a user might use. These can be gathered from surveys, observations of users, business models, and so on.

4.4 Benefits of Automation Methods

Talking about specific keyword- and data-driven automation techniques they provide benefits such as:

- immensely large test coverage proportional to the data size that is being used,
- the comfort of horizontal scalability through the addition of new keywords and data rather than having to write scripts from scratch,
- less weight and requirement for the codebase that is under the application,
- and highly reusable data that is not embedded in the test itself.

5 Summary

In this report, we have elucidated the importance of software testing tools in ensuring the quality and reliability of software products. We began by discussing the rationale behind using testing tools, highlighting their role in streamlining testing processes and enhancing productivity. Subsequently, we delved into the various types of testing tools, including test design tools, data tools, and test execution tools, each serving distinct functions within the testing ecosystem.

Furthermore, we explored the concept of automation in testing, outlining the reasons for automation, the appropriate scenarios for automation, and different automation methods such as capture/playback, data-driven automation, and keyword-driven automation. Finally, we summarized the benefits of automation methods in improving testing efficiency, scalability, and maintainability.

Through this exploration, we have underscored the critical role of testing tools in modern software development practices, emphasizing their significance in achieving high-quality software products. As organizations continue to navigate the complexities of software development, the adoption of testing tools remains essential for ensuring the delivery of robust and reliable software solutions.

REFERENCES

Bath, G. B., & McKay, J. M. (2014). The Software Engineer's Test Handbook (2nd ed.) [O'Reilley]. Rocky Nook.

Spillner, A., Linz, T., Rossner, T., & Winter, M. (2007). Software Testing Practice: Test Management: A study guide for the Certified Tester Exam ISTQB Advanced level. https://openlibrary.org/books/OL8815726M/Software_Testing_Practice_Test_Management