OULUN AMMATTIKORKEAKOULU

Shehryar Ali Malik

User interface, Accessibility, and Automation Testing Principles for E-commerce Platforms

User interface, Accessibility, and Automation Testing Principles for E-commerce Platforms

Shehryar Ali Malik
Spring 2024
Modern Software and Computing Solutions
Oulu University of Applied Sciences

ABSTRACT

Oulu University of Applied Sciences
Modern Software and Computing Solutions

Author(s): Shehryar Ali Malik
Title of the thesis: User Interface, Accessibility, and Automation Testing Principles for E-commerce Platforms
Thesis examiner(s): Teemu Leppänen
Term and year of thesis completion: Spring 2024

Ensuring an ideal user experience, accessibility, and effective testing procedures are essential components for success in the quickly changing world of e-commerce platforms. This thesis explores the fields of user interface design, software testing, and the importance of accessibility in the context of e-commerce. The investigation covers different software testing methodologies, testing phases, and the critical function of user experience. The guidelines for designing user interfaces for e-commerce platforms are highlighted, focusing on accessibility as a key component. One of the main goals is to include test automation into the testing framework, considering factors like coverage requirements, time and money constraints, and other limitations. The thesis offers an overview of well-known automation tools, such as Appium for mobile automation, Selenium for online automation, and Robot Framework for web automation. This also explores test automation's drawbacks as well as its benefits, covering anything from upfront costs to difficulties adjusting to quick changes. The features of test suites and scripts, successful and unsuccessful tests, and automated log analysis are all covered in detail in this examination of the automation testing process. The investigation is enhanced with a case study analysis that compares the testing procedures of two different e-commerce platforms. The comparative study discusses advantages and disadvantages and provides insightful information for improving testing methods. This thesis essentially adds to the growing body of knowledge on automation testing, accessibility, and user interface principles for e-commerce platforms.

Keywords: Software Testing, Usability, Testing Approaches and Levels, Test Automation, Robot Framework, E-commerce

Table of Contents

1 INTRODUCTION

In the modern fast-moving world of technology, it is an extremely essential factor for companies to stay up to date with the latest advancements in the market. New software versions are being developed in weeks now and companies are investing a lot of effort and resources to be the front-runners in their niche. This leads to a fast-moving and continuous process of software development and continuous integration of small bits so that the systems keep running while being upgraded. Software development teams and testing teams must continuously adopt new and faster methods of development and testing to keep up with the pace of the market and ensure that the consumer always has the best possible solutions.

Software Testing is a core process in software development. Not only do new features and developments need to be tested before and after deployment, but the existing functionalities also need to be tested to check if they are working normally after the changes or not. Even after the deployment of software and applications, there are other testing procedures for usability testing, accessibility testing, and performance testing among many others to ensure that the product is working at its full potential and the consumers are satisfied.

The goal of this thesis is to shed more light on the need for Software Testing and the importance of Usability Testing and Accessibility Testing. The methods used for evaluation will both be manual evaluations and automated evaluations using Robot Framework as a tool for developing and running Test Automation scripts. Moreover, the limitations and challenges faced by the organizations while implementing these methods will also be discussed and recommendations will be made for future reference.

2 SOFTWARE TESTING: THEORY AND BACKGROUND

2.1 Software Testing Approaches

The approach or method used for software testing depends on various factors. There are different methods to test the functionality of the software than to check if it meets legal requirements or the user-friendliness of the software. All testing methods can be used at any testing level starting from the basic code to the finished software and it is crucial to get customer satisfaction and their trust in the organization through these practices.

In the powerful domain of software development, the honesty and unwavering quality of a product application stand as points of support deciding its prosperity and client fulfillment. The systematic use of a variety of testing methods is necessary to guarantee a software system's functionality, performance, and overall quality. These methodologies, overall known as software testing approaches, are crucial parts of the product advancement life cycle. They want to find flaws, check the software's functionality, and make sure it meets all the requirements. Each approach, which ranges from manual testing to automated testing, unit testing to acceptance testing, contributes to the development of software that is both robust and user-friendly. This thesis dives into the complexities of these product testing draws near, disentangling their importance, individual attributes, and aggregate effect on the quest for conveying top-notch programming arrangements. As we explore the different scenes of testing procedures, the objective is to understand how each approach assumes an exceptional part in the bigger setting of software development, eventually guaranteeing that product applications fulfill the developing needs and assumptions of end-clients (App & Browser Testing Made Easy, n.d.).

Black Box Testing

In this approach, the internal structure of the code or the system is unknown to the tester. This approach can be used to check the functionality of a webpage or any other system and evaluate it against predefined standards. The name of this approach suggests that the architecture and networking inside the system are unknown to the tester, so they must report the changes or bugs to the team who knows the internal architecture. The functionality of the webpage or a system may

include user interface, starting and termination of the system, communication with other systems or services, performance issues, or compliance issues with the standards (Software Testing Fundamentals, n.d.).

Black box testing is a strong testing procedure since it practices a framework from start to finish. The emphasis is on trying the usefulness of the product with practically no information on its inside code design, logic, or execution details as shown in Figure 1. The tester regards the product as a "black box," collaborating with it through its outside connection points and sources of info and noticing the results (Black Box Testing, n.d.).

Figure 1: (What is BLACK Box Testing? Techniques, Types & Example, n.d.)

Black box testing has both pros and cons. One of the major benefits of black box testing is that the tester doesn't need to have a programming background or needs to be a technical person to perform this testing. This testing is performed just to check if the designed application is performing as desired. The major con of black box testing is that if the testing fails, it can be very difficult to find the source of error especially if the tester is of a non-technical background (Application Security - Black Box Testing, n.d.).

White Box Testing

The white box testing approach is focused on the internal structure of the code, component, or system. This approach can also be used on any test level in the hierarchy of the system. It is easier to locate the errors as the component under observation is open to the tester, but the tester should be familiar with the required knowledge to assess the faults and the procedures to remove them. Unlike black box testing, an error can be traced to the source at the time when the tests are being executed by the same person rather than referring the problem to a different person or a team (Ehmer, 2012).

The essential objective of white box testing is to guarantee that the code is legitimately solid and liberated from mistakes. This kind of testing is regularly finished at the unit, coordination, and framework levels of software development. The figure below shows the general structure of how test case is input into the application code which returns the test case output.

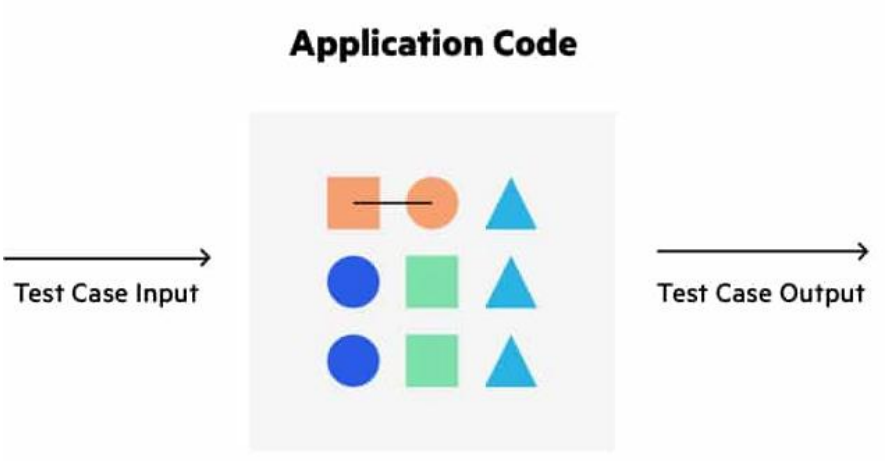

Figure 2: Test case execution (White Box Testing, n.d.)

The key factors of white box testing can include:

1. Testing Internal Logic
2. Code Coverage
3. Testing Cases Based on Code Structure
4. Knowledge of Implementation Details

(White Box Testing, n.d.)

Grey Box Testing

This is a combination of the previous techniques, and the tester has a hybrid knowledge of the internal structure of the system or component as well. This approach can be helpful if the resources are limited but it cannot give full code coverage or requirements coverage because a very high level of skills is required to achieve all of this by one person. This approach can also be implemented on any test level in the system hierarchy (Ehmer, 2012).

Grey Box testing ordinarily recognizes setting explicit mistakes that exist in web applications. For instance, if a defect is discovered during testing, the tester fixes it by making changes to the code and then tests it again in real-time. It provides the capacity to test both the front end as well as inside coding structure. Integration and penetration testing are the most common applications for it (Grey Box Testing, n.d.). The figure below shows the general structure of how the combination of black box and white box testing results into grey box testing.

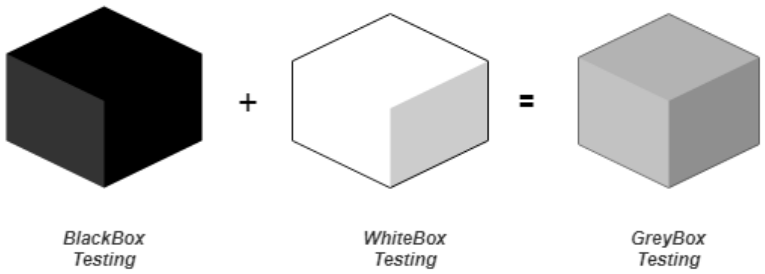

Figure 3: Comprehensive Testing (GreyBox Testing, n.d.)

2.2 Levels of Software Testing

Unit Testing

This is the first step as seen in the testing procedure as shown in Figure 4. The developer carries out tests here to check an individual module of code or webpage to see if that is working correctly. It is also sometimes referred to as component testing. The functionality can be tested with sample input values or by visual observation before the component is integrated with other components. Unit testing is also performed on all units individually sometimes after the changes are made in the system as a whole and these tests can be automated as well to save time and effort (What is Software Testing | Everything you should know, n.d.).

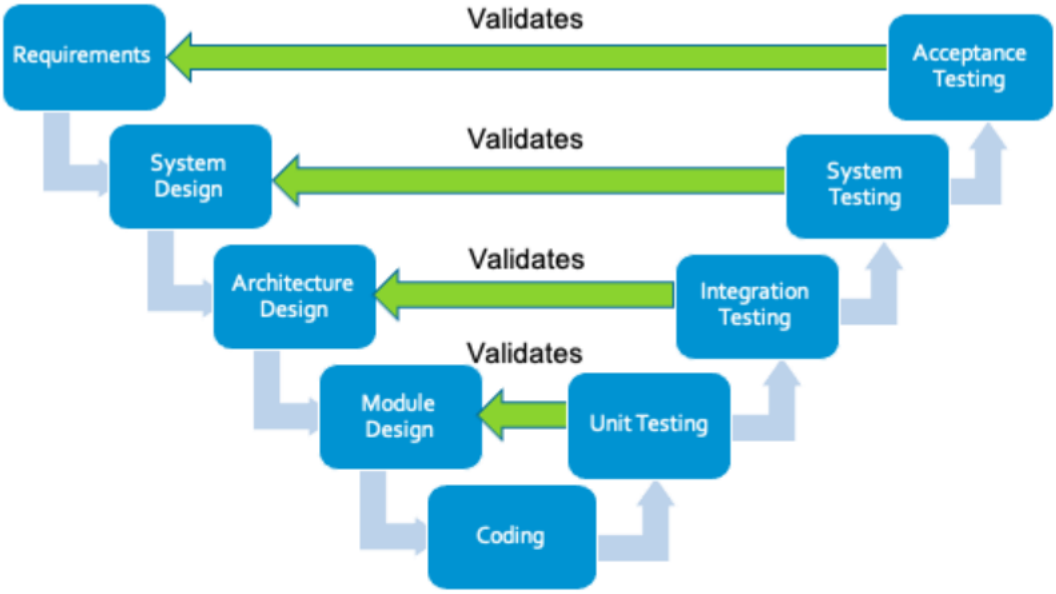

Figure 4: Levels of Software Testing (Camacho, 2022)

It can also be considered as each building block within the system. The purpose of this is to validate each building block separately so that it is working accordingly. A "unit" in this context is the smallest testable part of an application, typically a function, method, or procedure within the code. As unit testing is the first phase of testing, sometimes testers ignore this which can lead to a high cost for fixing the defect that is discovered during the next phases of testing. So, it is better to perform the unit testing so that there is minimal chance of defects in the next phases of testing. Unit testing can also serve the purpose of documentation, as each unit can be documented and ultimately it makes the complete documentation of the application (Unit Testing Tutorial – What is, Types & Test Example, n.d.).

Integration Testing

If the individual modules or units are working correctly then they are integrated into an existing system or combined to begin building a new system. The testing of interactions between these components or their interfaces is called integration testing. In this procedure, the focus is based on networking and communications between units rather than their functionality. If communication patterns are not yet developed between components, then stubs are used to make a close-to-reality working system. Integration testing is also used for systems when they are integrated into larger systems and the focus will also be communications between interfaces and smooth information

transfers between the systems (Bentley, 2018). The figure below shows the common region between module A and module B is called the Integration Testing.

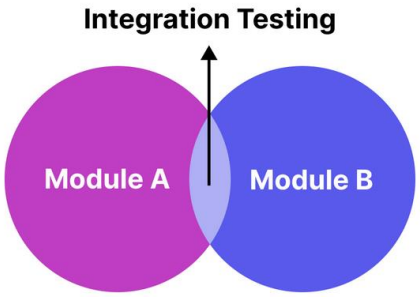

Figure 5: (What is Integration Testing? Definition, How-to, Examples, n.d.)

There are two types of Integration Testing. The Top-Down type has the following attributes.

- Starting at the top of the software hierarchy, this method begins testing and gradually moves down to lower-level modules.
- As the top part of the software hierarchy includes the critical functionalities, this type helps to test those first.

The Bottom-up approach has the following attributes.

- This approach starts testing from the lower-level modules and bit by bit climbs to more high-level modules.
- It ensures that the system's foundation is solid and assists in the early detection of issues in the lower-level modules.

System Testing

System testing is performed in comparison with the requirements after all the components are integrated successfully as shown in Figure 6. It is closely related to black box testing as the input and output of the system are usually under observation rather than the internal communication which is tested in previous steps. The system can be a product as a whole or it can be a part of an even

bigger system and another integration test will follow to see if it must be integrated into a larger system.

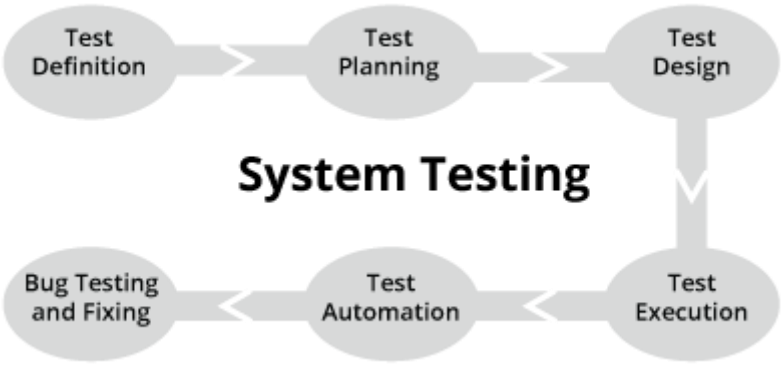

Figure 6: (What Is System Testing – A Ultimate Beginner's Guide, n.d.)

It is performed by an expert and dedicated team of testers as its purpose is to test the complete system. All the modules of the application are integrated into one and then tested to check if the application works as expected or not.

The key difference between Integration and System testing is that integration testing involves integrating a combination of modules and testing them one by one whereas, in system testing, all the modules are integrated and tested as a whole. For example, if an application has three modules M1, M2, and M3, then integration testing will first test the integration of M1 and M2, then M2 and M3, and at last M1 and M3. In system testing, all three modules M1, M2, and M3 will be integrated and tested as a whole (What Is System Testing – A Ultimate Beginner's Guide, n.d.).

Acceptance Testing

After the testing of the system inside the organization, the next step is the evaluation by the customers or end-users. In this step, the requirements from the customers are checked and the coverage of these requirements is evaluated by the customer. There are also other aspects of legal obligations and ease of use which are kept in mind at this step as if this step is passed then the product is ready to be delivered to the customer.

Regression Testing

The software undergoes many upgrades and fixes in its lifetime and these events often disturb the existing functionality of the software. Whenever there is such an event, it is necessary to ensure that after the upgrade or a fix, there is nothing broken in the software that was previously working. This type of testing is called regression testing as these disturbances are called regressions. This process is an ideal candidate for automation as the same tasks must be executed repeatedly during the life cycle of the software or application. Various frameworks are used for test automation which will be discussed later.

2.3 Introduction to Test Automation

The importance of Test Automation is significant in the software development lifecycle, and it cannot be overlooked in any progressive organization. All kinds of test types and test levels can be automated if needed using automation tools and scripts. The general rule is that if a testing process is required more than once, then it is a candidate for automation. However, the budget and time limitations can make the organizations choose from all suitable candidates and automate the ones who are most suitable or repeated most often. Test Automation is very useful in the present software development practices in which new features are being implemented in weeks and even days so they must be tested at every step along with the regressions that will happen once these features are implemented.

There are levels of software testing starting from unit testing and ending at acceptance testing as already discussed in this chapter. If a feature must be tested at the unit level and all the other levels, then it will be a time-consuming process to do every step manually every time, and after the testing is completed, there will be bugs to fix and then repeat the whole process. Continuous delivery practices in software development enable organizations to make workflows which include software development, testing, implementation, and bug fixes all running simultaneously and no step can make a bottleneck and delay other steps. In this environment, test automation plays a vital role in speeding up software delivery and helping organizations keep up with the pace of the market. The most suitable candidate for test automation is always regression testing as it has to be repeated every time a change is made so that possible disruptions can be fixed. But testing of user interfaces,

databases, networks, and all the other components are also suitable candidates depending upon the product and architecture of the organization (What is Automation Testing?, n.d.).
Automation Testing helps in improving accuracy, increases testing speed, and is helpful in the consistent testing of the application. Table 1 shows the comparison of the Automated Testing and the manual testing in terms of e-commerce websites:

Automated Testing	Manual Testing
These tests are based on the scripts, so if the scripts are accurate, these tests would result in 100 percent accuracy.	These tests can result in errors as they are performed by humans manually.
This type of testing gives the results immediately.	Results are generated manually by the testers, so it is time-consuming.
Once a script is created, it can be reused every time testing needs to be performed.	This type of testing needs to be done from scratch every time.
This type of testing is performed very quickly.	This type of testing is time-consuming.

Table 1: (Automatic tests in eCommerce: what is their purpose?, n.d.)

2.4 User Experience in the Software Industry

It has been discussed how the technological advancements in the modern era have made software production immensely competitive and fast-paced. There are testing procedures that are related to the product which have been discussed in the previous sections but there is a whole other aspect of User Experience that needs to be addressed as well. Many factors are included in user experience which include the value of the product, ease of use, desire to use, and the ease of adaptability to the product especially for the first-time user.

There are many competitors in every niche and a user is comparing among them when they first start searching for something. The above-mentioned principles are very important at that time to make the user engaged and interested and make them a permanent customer by providing the ultimate user experience. The chances of a user returning after they have chosen a specific option among the competitors are very thin and most likely they will find another option which they will

stick to forever. Consequently, no matter how superior the internal quality of a product is, it is judged first time based on the external user experience and that proves the importance of this aspect in the software development life cycle (More Than Usability: The Four Elements of User Experience, n.d.).

Figure 7: Importance of Usability (Why Usability Is Important?, n.d.)

Usability is very important in terms of how much the user enjoys using the product as evident in Figure 7. The needs and requirements of the targeted users must always be considered when designing the product so that the users can easily understand the product. When designing the product, it is important to understand that the whole product represents the company. It is the image of the company that is being represented through the designed product. So, the product must be according to the user's needs, a user must understand its usage easily and the user must enjoy using the product.

The product should be simple, there should be clarity about what the product is about, and there should be consistency in the interface of the designed product. Feedback from the users should be taken regularly so that more improvements can be made with time. One way to test if the product is designed as per the user requirements is to check if there are minimal calls to the support or technical support departments of the company regarding the designed product.

2.5 User Interface Principles in E-Commerce

The principles which are used to define the user interface in e-commerce platforms are described below:

- Ensure consistency among the e-commerce platform's various pages' design components, layouts, and interactions. When interacting with a website or application, consistency promotes familiarity and comfort in the user experience.
- Highlight the significance of visually appealing and captivating design components. This involves utilizing typefaces, color schemes, photography, and other visual components that complement the brand identity and improve the user experience.
- Ensure that the e-commerce platform is aesthetically pleasing and usable on a range of screens, including those of smartphones, tablets, and PCs. Regardless of the device being used, a seamless user experience depends on responsive design.
- Regular feedback should be taken from the users and that feedback should be incorporated in the design of the system. This will help users gain trust in the application and then they become your permanent clients.

2.6 Importance of Accessibility

There is another aspect of a good design which is called being accessible. The design of a website or application should be usable by as many people as possible to increase the number of customers. There are visually impaired people and people with other disabilities who cannot use the conventional methods of accessing webpages and the information must be deliverable by making it more accessible.

There are multiple ways shown in Figure 8 which make a product more accessible although not many products are focused on this aspect. Usually, the websites of government institutions or media generally consider this aspect of accessibility, but it should be considered in every product. There is a feature called Text-to-speech which can be used by visually impaired people to hear the text on the webpages. It is one of the most common features used for accessibility. Also, there are options to make the interfaces magnified with fewer options so that the most important information can be shortlisted. One feature that can be very useful is using artificial intelligence to take voice

commands and do tasks on behalf of the user. This is a feature that is seen in cell phones, and it is very efficient even now but implementing this feature in webpages is a thing of the future.

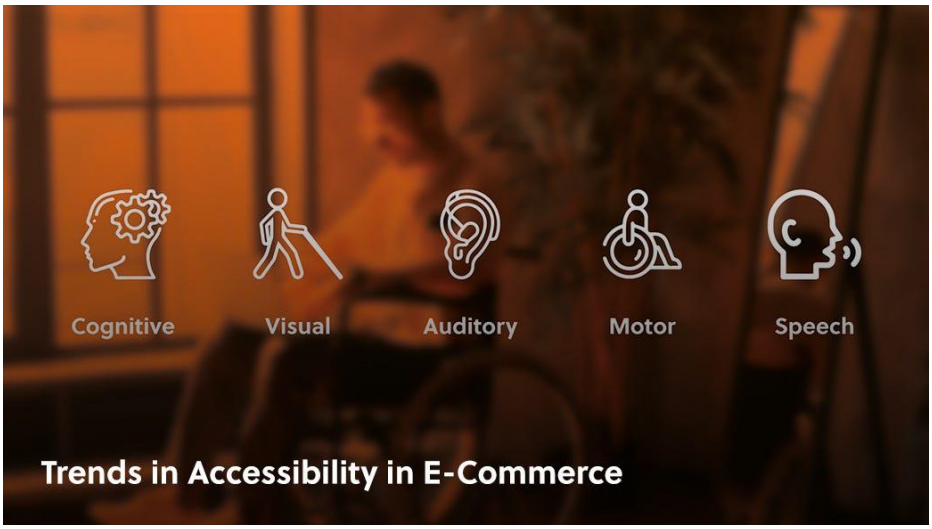

Figure 8: Accessibility in E-commerce (Trends in Accessibility in E-Commerce, n.d.)

All over the world, many disabled people shop online to get their required products. An e-commerce store is said to be accessible when it is convenient to use for every type of audience whether they possess any disability or not. There can be many kinds of disabilities like visual impairments, color blindness, and hearing disabilities. There should be a way on the online store for these kinds of people to easily buy their products.

2.7 Evaluation of Usability

As discussed previously, user experience holds a significant role in making a customer who comes for the first time a permanent customer. This is true in any field of life, matter digital or physical. The value for money and good experiences always makes people recommend the product to others and keep using it themselves. While a product may fulfill all the technical requirements, it may not be a good product from the perspective of usability. The layouts and steps are not well defined, and the status is not always visible to the user. There are various methods to evaluate digital prod-

ucts from the usability and accessibility perspective which can provide valuable feedback and suggestions for improvement. The process of usability testing is as iterative as other testing procedures. There are multiple prototypes involved with the constant evolution of the design until the end user gives the highest possible scores for its usability. The design could be favorable to a certain group of target audience which are most users, and this can be a benchmark for a successful design as satisfying everyone is not likely to happen (Usability 101: Introduction to Usability, n.d.).

2.8 Methods Used for Evaluation of Usability

There are many ways to test usability and user experience, but the work done by Jakob Nielsen is known worldwide when it comes to usability. The rules and definitions defined in his work are used on a very wide scale in evaluations and surveys to make the user interfaces better. The rules are summarized below,

1. The current state of the system and the decisions must always be quickly visible to the user.
2. The language and interfaces should be as close to the real world as possible so that more different types of users can use them.
3. The options to correct a mistake or cancel the whole process should always be available.
4. The phrases and common terms of the real world should mean the same thing inside the systems so that the learnability of the system is easy.
5. There should be checks to avoid errors so that difficult situations can be avoided beforehand rather than at a later stage.
6. The information flow in the system should lean towards recognition of previously viewed elements rather than recalling them due to lack of visibility.
7. Shortcuts and process accelerators should be provided both for new users and experienced users.
8. The design should be minimalistic so that the user does not have to find the key information from clusters of things.
9. Recovery from errors and suggestions for solutions should be as simple as possible.
10. There should be proper help and documentation sections to cover the most important parts of the system.

After applying these general rules to any system, the system will at least be easy to use and less prone to errors. The visuals are also covered here so that the design is not complex to learn, and the user does not need to remember a lot of things. These rules can also be applied to physical products and systems to make them better in usability (10 Usability Heuristics for User Interface Design, n.d.).

Another way to evaluate digital products is by getting actual feedback through surveys from users. Some common standards are used by most organizations to get feedback from users and evaluate the products. Moreover, specific surveys and questionnaires can also be developed if the product is one of a kind or it needs to fulfill some specific requirements than general ones. An example template for evaluation is below.

Ratings: 5 Excellent 4 Good 3 Acceptable 2 Poor 1 Bad

Heuristics	Rating	Description	Notes
1. Visibility of system status	5	The system should always keep users informed about what is going on, through appropriate feedback within a reasonable time.	
2. Match between the system and the real world	5	The system should speak the users' language, with words, phrases, and concepts familiar to the user, rather than system-oriented terms. Follow real-world conventions, making information appear in a natural and logical order.	
3. User control and freedom	4	Users often choose system functions by mistake and will need a marked "emergency exit" to leave the unwanted state without having to go through an extended dialogue. Support undo and redo.	
4. Consistency and standards	4	Users should not have to wonder whether different words, situations, or actions mean the same thing.	
5. Error prevention	3	Even better than good error messages are a careful design that prevents a problem from occurring in the first place. Either eliminate error-prone	

Heuristics	Rating	Description	Notes
		conditions or check for them and present users with a confirmation option before they commit to the action.	
6. Recognition rather than recall	4	Minimize the user's memory load by making objects, actions, and options visible. The user should not have to remember information from one part of the dialogue to another. Instructions for the use of the system should be visible or easily retrievable whenever appropriate.	
7. Flexibility and efficiency of use	4	Accelerators—unseen by the novice user—may often speed up the interaction for the expert user such that the system can cater to both inexperienced and experienced users. Allow users to tailor frequent actions.	
8. Aesthetic and minimalist design	4	Dialogues should not contain information that is irrelevant or rarely needed. Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility.	
9. Help users recognize, diagnose, and recover from errors	3	Error messages should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution.	
10. Help and documentation	4	Even though it is better if the system can be used without documentation, it may be necessary to provide help and documentation. Any such information should be easy to search, focused on the user's task, list concrete steps to be carried out, and not be too large.	
Overall Average	4		

Table 2: Accebility test template (Accessibility USDA, n.d)

This survey is just an example of how this task can be achieved. There are other ways to test and evaluate usability by asking the user to do a specific task and think aloud during the process. Altogether, these methods are designed to make the designs better and increase the ease for the user.

3 SOLUTIONS FOR TEST AUTOMATION

Test automation is defined as doing the manual tasks automatically using tools and code or scripts so that the testing process is speedy, and less manpower is required to do the repetitive tasks. Test automation can be implemented both on the front end and the back end of the products. Networking can also be tested so test automation has a wide variety of applications, and the popularity is increasing.

In common practices, Smoke tests are the ones which are automated most of the time because they are related to the existing functionality of the system. Similarly, Regression testing is more closely related to the existing codebase functionality. Performance testing and Security testing can also be automated if there are procedures and standards which are must to be achieved.

3.1 Requirements for Test Automation

The automation testing process involves multiple steps which are explained below:

1. Automation testing means using different tools to perform tests automatically. These tasks when performed by humans are time taking. These tests are performed by writing separate lines of code. These lines of code test the actual code to check if the actual code meets the requirements or not.
2. An essential first step in the automated testing process is setting up a test environment. Reliable and repeatable test results are ensured by a well-configured test environment. An overview of how to set up a test environment for automated testing can be found here:
3. Clearly state what your test environment's needs are. This covers the specs of the hardware, the software, the network setups, and any other requirements. This is important so that everyone can understand the requirements beforehand.
4. Identify the kinds of test environments that are required, including those for development, testing, staging, and production. This is important so that there is proper procedure in line before the development, testing, and production phase.
5. Identify the hardware and software specifications required for the test environment. This may include servers, databases, operating systems, browsers, and other tools. All these

selected should be of the best available specifications as per the application being developed.

6. To organize and version your test data, scripts, and configuration files, use version control systems. This guarantees that, should the need arise, you may quickly revert to a certain version. This can help if there is a system failure or is a need to revert to a previous version.
7. Create scripts or configuration files that automate the setup of your test environment. This could include scripts to install software, configure settings, and deploy applications.
8. To prevent any unintentional influence on active systems, keep the test environment isolated from the production environment. Make use of distinct server, database, and network configurations. This can serve as a backup as well as manage the load on the main environment.
9. Prepare the test data needed for your test cases. For thorough testing, this data should include a range of scenarios and edge cases. Test data should be set up to cater to the worst-case scenario.
10. Use configuration management tools to manage and track changes to the test environment. This helps in maintaining consistency across different environments. This can also help in keeping the logs for all the environments to maintain the history.
11. To establish and maintain isolated environments, take into consideration utilizing virtualization or containerization technologies. This makes it simple to distribute and replicate the test environment. This also helps in saving the hardware cost.
12. Set up the network configuration to resemble the production environment as much as feasible. This covers load balancers, firewalls, and other network hardware. The network should be made as safe as possible to prevent any possible attacks.
13. Make sure the testing grounds are safe. Protect sensitive data by putting security measures like firewalls, access limits, and encryption in place. All the environments should be made as safe as possible.
14. To keep the test environment up to date with changes in the production environment, update and maintain it regularly. This covers patching, software updates, and test data refreshes. Monthly hardware check-ups can be done to ensure all the hardware is working properly.
15. Make sure to record every configuration and dependency during the setup process. In the future, this documentation will be essential for maintaining and debugging the test environment. This will help every person to understand the setup easily. This documentation can be improved with time.

(Test Environment: A Beginner’s Guide, n.d.)

3.2 Regressions and Solutions

Unintended effects or flaws that arise when new modifications are introduced to a software program are referred to as regressions. These adjustments could take the shape of added features, bug patches, or adjustments to already-existing functionality. Regression testing is the process of executing test cases that have already been completed to make sure that the most recent modifications have no detrimental effects on the application's current features.

Introduction of New Bugs

There's always a chance that new bugs or flaws may be introduced when developers add new code or modify already existing code. A comprehensive set of automated tests must be run to fix this. To make sure that recent modifications don't unintentionally destroy already-existing functionality, a variety of situations are tested.

These modifications can be tested bitwise so that when one modification is made, it is tested. This will ensure that all the bits are separately tested and then the system can be tested to see if it is working as per the requirements. Bugs can always arise during development, so it is always better to test the modifications bitwise, as finding bugs in the whole system can sometimes be very difficult.

Changes in Application Behavior

Insignificant changes to the program might change how it behaves and have unanticipated results. Automated tests should be conducted regularly to ensure that the program continues to function as intended following changes. This guarantees that the functionality of current features won't be negatively impacted by future ones or bug patches. Regression testing is essential for identifying unanticipated modifications or defects that were inadvertently introduced during the application's development or modification. Tests that were previously passed can suddenly fail, which suggests

a regression. Integration problems may arise if the application interacts with other services or systems and their behavior or APIs change. Regression testing aids in verifying that the integration points continue to operate as intended.

Integration Issues

Modifications made to one area of the application might cause problems integrating it with other parts. The automated suite's integration tests verify how various modules or services interact with one another. This aids in the early detection and correction of integration problems throughout the development process. To ensure that data interchange and communication between various system components continue to function as intended, automated tests are conducted. Automated tests verify that the data transferred during integration adheres to the anticipated patterns and maintains consistency even in the event of application modifications.

Performance Degradation

The application's performance may suffer from new features or modifications, resulting in resource consumption or slower response times. A slower user experience may arise from modifications made to the coding that introduce inefficiencies, bottlenecks, or unexpected effects. The automation suite's performance testing feature helps track and fix any deterioration in an application's performance. This guarantees that even after updates, the application will continue to function well and be responsive. Regression testing is used to identify the precise modifications that led to the drop in performance, should it be identified. This entails figuring out which code commits or updates are causing the performance problems (What is Regression Testing?, n.d.).

Solutions for Effective Regression Testing in Test Automation

1. It is necessary to create and manage an extensive collection of automated test cases. Unit tests, integration tests, and end-to-end tests covering essential features are all included in this. As the application changes, the test suite's relevance and efficacy are guaranteed by regular updates and expansions.
2. Code change integration, test execution, and application deployment are all automated via CI/CD processes. Automated regression tests are started with every code commit, giving

engineers instant feedback. By doing this, problems are certain to be found early in the development cycle.

3. Git and other version control systems are crucial for monitoring codebase changes. They make it possible for team members to work together, offer the capacity to undo changes and make it easier to pinpoint the exact moment that problems first surfaced.
4. The secret is to use data-driven testing and parameterization to make tests reusable and adaptable. Data-driven tests employ a variety of inputs to evaluate different conditions, increasing test coverage, while parameterized tests enable testing of several scenarios using the same test logic.
5. Based on their potential to impact the company and their propensity to be impacted by changes, identify, and rank the most important test cases. Regression testing guarantees that essential capabilities are evaluated properly by concentrating on high-impact regions. (Automated Regression Testing: A Detailed Guide, n.d.)

3.3 Robot Framework

Robot Framework is a popular acceptance test-driven development (ATDD) and keyword-driven test automation framework that is available as an open-source tool. It offers a straightforward yet effective syntax and facilitates behavior-driven development (BDD) as well as keyword-driven testing. The Robot Framework offers the following benefits to test automation solutions:

Keyword Driven Testing

Robot Framework uses a keyword-driven methodology in which test cases are specified using keywords in a natural language style. This facilitates the writing and comprehension of test cases for those with non-technical backgrounds. Writing test cases more understandably and legibly can help the technical and non-technical team members work together more effectively. Operations or actions are represented by keywords. These could be bespoke keywords made by testers or built-in keywords supplied by Robot Framework. The use of keywords encourages reuse. It is simple for testers to reuse common activities across different test cases by building a library of custom keywords.

Behavior Driven Testing

Robot Framework allows Gherkin-style syntax, which helps with BDD. This facilitates the creation of feature files that stakeholders, both technical and non-technical, can read. BDD syntax encourages cooperation and a common understanding of requirements among developers, testers, and business stakeholders. Test cases are arranged into feature files, each of which stands for a feature or component of the system that is being tested. BDT enables the creation of expressive and intelligible test cases that non-technical stakeholders can read and comprehend with ease (Robot Framework User Guide, n.d.).

Cross-Platform and Cross-Browser Testing

Cross-platform testing is supported by Robot Framework, enabling automation across many operating systems. It may also be applied to cross-browser testing. The same test scripts may be used to test applications that are operating on many platforms and browsers, increasing test coverage, and guaranteeing application compatibility. Cross-platform testing confirms that a program functions properly across many operating systems, including Linux, macOS, and Windows.

Cross-browser testing ensures that a program works properly across a range of web browsers, including Edge, Firefox, Chrome, and Safari. These techniques can be used with Robot Framework to write tests that are adaptable enough to run on several browsers and operating systems, offering coverage, and guaranteeing a consistent user experience in a variety of settings.
Because Robot Framework can be expanded upon and integrated with other tools and libraries, teams may take advantage of pre-existing resources and technological advancements. To improve the entire automation pipeline, teams may combine Robot Framework with other technologies for reporting, continuous integration, and version control (Cross-Platform & Cross-Browser Testing: The Right Way Towards Quality Software, n.d.).

3.4 Selenium for Web Automation

A popular open-source framework for automating web applications is called Selenium. It offers a range of online browser automation capabilities and is compatible with Java, C#, Python, Ruby, and JavaScript, among other computer languages. Selenium is a key component of web automation and is frequently used for functional testing of online applications.

Cross-Browser Testing

The process of making sure an online application functions properly in a variety of web browsers is known as cross-browser testing. As seen in Figure 9, selenium facilitates automation for a variety of web browsers, including Edge, Firefox, Chrome, and Safari. Web applications may be tested across a variety of browsers to guarantee cross-platform compatibility and consistent behavior. It's essential to test your online application across a range of browsers to guarantee a consistent user experience, as different browsers may interpret and render web elements differently.

Reports are generated to track test results for individual browsers, and special considerations can be required for browser-specific behaviors. Integrating with cloud-based testing services and continuous integration pipelines improves speed and coverage while guaranteeing a consistent user experience across various browser contexts.

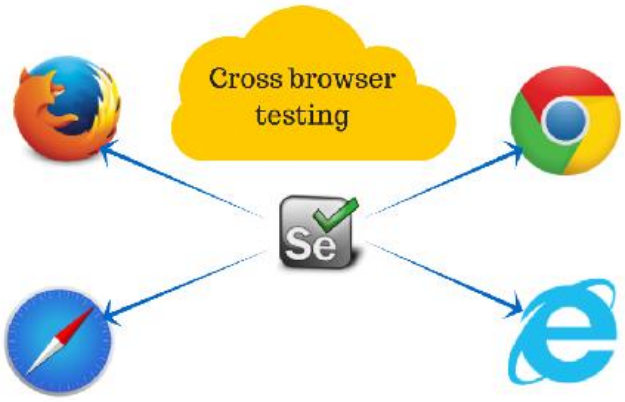

Figure 9: Implementation of Selenium (Cross Browser Testing using Selenium, n.d.)

Dynamic Web Element Interactions

JavaScript-generated dynamic web components may be interacted with using Selenium's methods. Modern, dynamic web applications can be accurately tested because of automation scripts' ability to handle dynamic information and interactions. Using techniques to deal with items whose characteristics change in runtime is part of handling dynamic web elements in Selenium for web automation. Both implicit and explicit waits are frequently used; implicit waits provide a general wait period for the WebDriver as a whole, while explicit waits let you wait for circumstances to occur

before moving on. Furthermore, it is possible to build dynamic XPaths and CSS selectors that may adapt to attribute changes, giving items greater flexibility in their location. Partial matches or alternative attributes can be used when working with dynamic IDs. Another effective strategy is JavaScript execution, which makes it possible to interact with components that would be difficult to reach with standard Selenium instructions.

It becomes essential to catch and handle StaleElementReferenceException in cases when DOM modifications cause elements to become stale. This error happens when an element is removed from the DOM, frequently because of updates or page refreshes. These methods enable Selenium scripts to smoothly adjust to dynamic web elements, guaranteeing dependable and strong web automation.

Headless Browser Testing

In Selenium, "headless browser testing" means using a browser without a graphical user interface (GUI). When the browser is in headless mode, it runs in the background, which saves resources and speeds up the process. This is especially helpful for automated testing in situations like server environments or continuous integration (CI) pipelines where UI rendering is not a crucial component. Headless browser testing is supported by Selenium for several browsers; setting up the WebDriver for headless operation is the process.

Faster test execution, less resource usage, and the flexibility to run tests in situations without a graphical user interface are just a few benefits of headless browser testing. Teams may quickly validate changes to web applications using this method, which is useful for continuous integration processes because it eliminates the need to run a visible browser (Selenium And Mobile Test Automation – Will Selenium Make The Cut?, n.d.).

3.5 Appium for Mobile Automation

An open-source automation framework for mobile apps, Appium works with both the iOS and Android operating systems as shown in Figure 10. It is a well-liked option for testing mobile applications across many platforms and devices as it offers a single, cross-platform API for mobile test

automation. Java, Python, and JavaScript are just a few of the programming languages you may write tests in with Appium.

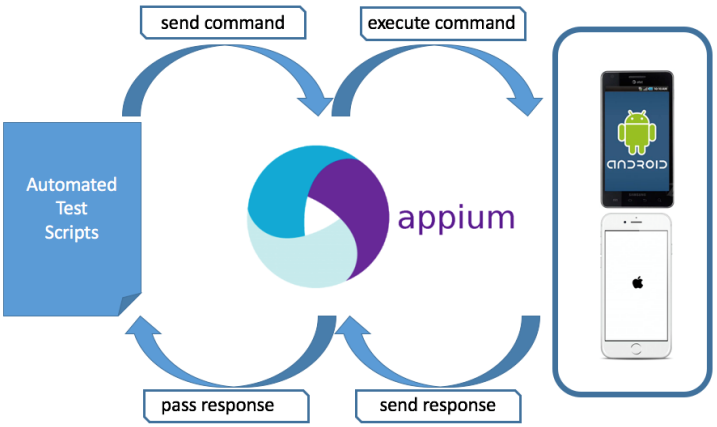

Figure 10: Implementation of Appium (Appium Mobile App Automation , n.d.)

Here is how Appium adds to mobile automation solutions:

Cross-Platform Mobile Testing

Using a single set of APIs, Appium enables the automation of both iOS and Android applications. Teams may save time and money by using the same test scripts for testing on both popular mobile platforms, as opposed to needing to maintain separate test suites. REST API is exposed by Appium, which functions as a server. It takes instructions from your test scripts and uses mobile devices to carry them out.

A set of desired capabilities is supplied to the Appium server when you start a session. They contain details about the automation settings, device, application, and platform. You can include any pertinent information along with the platform name (iOS or Android) for cross-platform testing. You may manage the app's installation and uninstallation on the test devices with Appium. Ensuring a clean testing environment is crucial. You can run tests in parallel on several devices at once to expedite testing. Parallel execution is supported by Appium, which is useful for cross-platform testing.

Real Device Testing

Both emulators and simulators as well as actual devices may be automated with Appium. Teams may utilize emulators and simulators for scalable and affordable testing in addition to testing apps on actual hardware to guarantee authentic user experiences. An application's performance in a real-world setting is more accurately represented by actual device testing, which considers a variety of device-specific aspects. The physical devices must be connected to the computer running the Appium server to conduct actual device testing. If you are using distant devices, you can make this connection via USB or, if you choose a cloud testing service. You can manage the orientation of the device, take screenshots, install, and uninstall apps, and perform other operations related to the physical device with Appium. When testing scenarios involving device-specific behaviors, this degree of control is essential.

Wide Range of Programming Language Support

Programming languages supported by Appium include Java, Python, Ruby, JavaScript, and C#. When utilizing Appium for mobile automation, teams may use current skill sets and remain flexible by selecting their programming language. WebDriver is a standardized protocol for automating web browsers, and Appium adheres to it. This makes it possible for many language bindings to be consistent. Developers use the WebDriver API in their preferred programming language to write Appium scripts. Members of teams working on mobile automation initiatives may have varying levels of programming language competence. Teams can work together more successfully when Appium supports many languages, allowing each member to contribute to their language.
Developers who are already familiar with a specific programming language can leverage their existing skills to write Appium scripts. This reduces the learning curve and allows for a smoother transition to mobile automation testing. Members of teams working on mobile automation initiatives may have varying levels of programming language competence. Teams can work together more successfully when Appium supports many languages, allowing each member to contribute to their language (Mobile Automation with Appium, n.d.).

3.6 Disadvantages of Test Automation

Initial Investment

An initial investment in tools, equipment, and qualified personnel is necessary to set up test automation. The initial expenses might be high, and businesses would have to set aside money for instrument purchases and training. Recruiting qualified automation testers and upskilling your current staff may provide difficulties. The procedure may initially take longer due to the learning curve associated with test automation tools and frameworks. To account for modifications to the application being evaluated, automated tests must be updated frequently. Regular modifications to the application may need a large maintenance effort.

Not every kind of testing can be automated. Manual testing may be more appropriate in certain situations, such as exploratory or usability testing. If automation is your only option, you can miss certain important features of the program. Over-reliance is one potential risk. Even though automation has many advantages, manual testing is still necessary sometimes. Manual involvement may still be necessary for many areas of testing, particularly those that call for human touch or intuition.

Maintenance Overhead

To stay current with changes in the program, automated test scripts need constant updating. Automation may become less cost-effective over time if there are frequent modifications made to the user interface or functionality of the program. Automated test scripts may need to be adjusted to reflect updates, enhancements, or changes made to the application under test. This can entail changing test logic, updating test data, or tweaking locators for web items.

Updates and improvements are frequently released in new versions of test automation tools. Teams must put in time to make sure their test scripts are compliant with the most recent tool versions and to stay up to speed with these updates. Inaction on your part may result in script errors and compatibility problems. Automated test stability and reliability can be affected by alterations to the testing environment, such as operating system, browser, or third-party integration updates. Maintaining automated test functionality across many contexts calls for constant attention.

Limited Test Coverage

Certain testing situations, particularly those involving subjective human judgment, may not be covered by automation. Manual testing may be more appropriate for some areas, such as exploratory

testing, visual validation, and usability. Automated tests are frequently made to handle functionalities and scenarios. However, it could be difficult to automate complicated or unusual situations that call for human intuition or investigation.

Automating exploratory testing, which involves testers actively navigating the program to find unforeseen problems, is difficult. Automated tests may miss problems resulting from unforeseen user interactions or environmental factors since they adhere to pre-set routines. It can take a lot of effort and resources to create automated tests for every situation.

Difficulty in Testing User Experience

Automation technologies find it difficult to evaluate the user experience, including aspects like ease of use and aesthetic appeal. Manual testing may be necessary since automated scripts may not be able to sufficiently test some important parts of the user experience. The subjective concept of user experience encompasses elements like overall attractiveness, simplicity of use, and beauty. These components are difficult to measure and automate since they frequently rely on personal preferences and perceptions. Functional features and interactions with the application's interface are usually the subject of automated tests. But many elements of human interaction like the sense of responsiveness, the timing of feedback, and the overall feel are hard to precisely duplicate by automation.

Users engage with software in particular circumstances, and depending on those contexts, their expectations could change. It's possible that automated testing misses some of the contextual details that affect the user experience.

Inability to Adapt to Rapid Changes

Automation could find it difficult to swiftly adjust to frequent changes in specifications and the architecture of the application. Maintaining automation scripts at the same speed may be difficult in agile development settings with frequent releases. The application's current functionality serves as the basis for the creation of automated test scripts. The automatic scripts may become out-of-date if the application undergoes frequent modifications, such as bug fixes or feature upgrades. The duration required for these scripts to be updated and maintained may cause a delay in responding

to sudden changes. Automated scripts' test data may be impacted by sudden changes in program features. It can take time to manage and update test data to reflect changes in the application, which makes it more difficult to quickly adjust to sudden changes.

Test automation calls for a certain skill set, and delays may occur if the team is unable to quickly modify scripts in response to evolving needs. It might be difficult to learn and retain the skills required to keep up with the rate of change (Exploring Advantages and Disadvantages of Automation Testing, n.d.).

4 SOLUTIONS FOR TEST AUTOMATION

4.1 Testing Framework

In the cycle of software development, testing will always play a vital role because any product that is developed needs to be tested before the user can get it and the requirements need to be met beforehand. Although exhaustive testing is not possible and there will still be ongoing issues, it is the goal to cover as much as possible before releasing the application or a system. In the e-commerce business, testing is even more important because revenue is lost if the store is down for maintenance or people are unable to complete purchases due to any bug in the system. There are big companies that lose millions in some hours of downtime and this fact can emphasize enough on the need for and importance of testing procedures.

Considerations for Software Testing

Many factors influence the testing types and procedures to be selected for any product. The continuous delivery methods in most organizations presently require quick delivery of new software versions in a short span of weeks or even days. However, there are limitations in budget and time along with the preference of certain features which can influence the testing approach to be used.

Budget and Time Limitations

The resources of a team are to be allocated properly towards testing so that maximum efficiency can be achieved. Everything cannot be tested but there cannot be anything left behind as well. So, the balance between these two criteria is up to the managers and other factors. Testing at lower unit levels is required to avoid catastrophic failures at a later stage as they will cost a lot more to fix at later stages. Manual testing is also required as everything cannot be automated, and human observation is required to find errors. Bug reporting and rectifying these bugs is also required and then regression testing must be performed after every bug fix or new release of the product.

If time and budget do not allow for testing everything, some of the tasks are put in the backlog so that they can be checked in the next release. The features related to these tasks are also withheld

from the release so that they can be correctly implemented in the next one. Nevertheless, there can be no serious issues or even visual issues left in the product as they harm the organization and cause the loss of customers. More and more automation procedures are being integrated into testing so that time can be saved by automating as much as possible, especially in organizations with shorter release cycles. Also, small bugs can always surface as more and more people use the product and they must be removed periodically so resources are always required for testing to successfully keep the system running.

Requirements and Coverage

In software testing, no matter what approach or method is used or what tools are used for the evaluation, certain things remain the same. What are the requirements and coverage criteria? Requirements are provided by the clients, users, or any experts who are familiar with similar products. These requirements are the focus while developing and testing the product and it is a priority that all these requirements must be accommodated before the final product is ready for delivery. Requirement coverage for testing is defined as the percentage of requirements that are successfully tested and approved. The coverage percentage must be as high as possible. The figure below shows the relationship between the development stages and coverage.

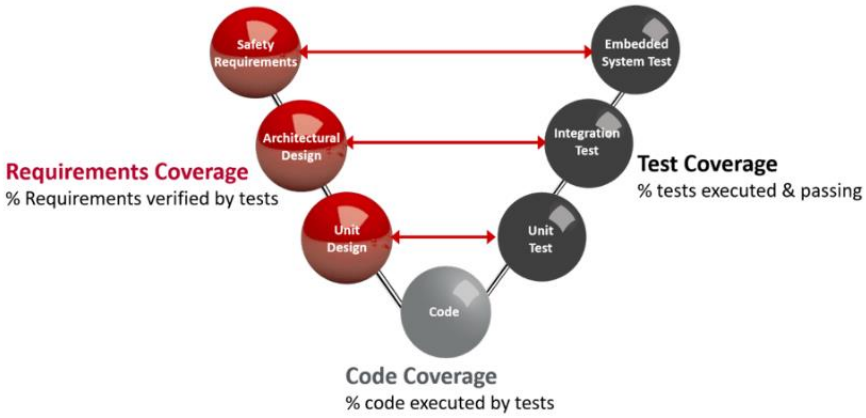

Figure 1: (What is meant by Structural Code Coverage?, n.d.)

The same criteria are valid for code coverage and test coverage. The number of lines of code that are executed and tested defines code coverage and the test coverage overall defines how much

testing has been performed from the unit level to the system level. High coverage does mean that as much testing has already been done but it does not still guarantee that there will be no bugs in later stages. After these stages, further testing is performed specifically to check the standardization or legal requirements for the product. Usability and user experience are also factors that need to be tested even if the system is covering all the requirements provided at the start of the development cycle.

4.2 Scripts and Suites

Test Scripts

A test script is a collection of commands or instructions written in a programming language that are intended to carry out operations and checks on the application that is being tested.
Depending on the automation technology being used, test scripts are usually written in scripting or programming languages like Java, Python, JavaScript, Ruby, or others. Scripts are made up of discrete test stages that perform various interactions with the program, such as data entry, button clicks, page navigation, and anticipated outcome verification. Assertions are used in test scripts to confirm that the program operates as intended. During the test, assertions verify if specific requirements are met.

Test Suites

A collection of test scripts and/or test cases kept in one container is called a test suite. Several tests may be arranged and run together using test suites.
It is simpler to combine relevant tests when test cases are managed and organized with the aid of test suites. The sequence in which tests are run is specified by test suites. Maintaining dependencies across tests depends on this. Certain test suites permit parameterization, meaning that the same test scripts may be run with various input values (OTM Testing Automation, n.d.)

Types of Test Suites

Different types of Suites have been explained below:

Smoke Tests: A collection of brief tests that swiftly determine if the application's most important features are operational following a build or release.

Regression Suites: Suites with an extensive collection of tests to make sure that recent modifications haven't affected any of the features already in place.

Functional Suites: Suites designed to test aspects or functions of the program.

End-to-End Suites: Suites that are all-inclusive and cover every step of the user's application journey (What is a Test Suite & Test Case? (with Examples), n.d.)

Relationship between Test Suites and Test Scripts

Test suites offer a greater degree of organization by combining several test scripts into logical collections, whereas test scripts are the smallest units of automation. Reusing test scripts across many test suites reduces duplication and facilitates effective maintenance. Scalability and maintainability are greatly impacted by grouping test scripts into test suites as the number of test scripts increases. Test suites offer a methodical way to perform a suite of tests by coordinating the execution of several test scripts.

4.3 Successful and Failed Tests

A test is considered successful or passed when the application or system meets the predetermined requirements and operates as expected. Important traits of effective tests consist of:

- There are no exceptions or failures throughout the automated test's execution.
- The test script contains assertions that all evaluate to true, meaning that the application behaves as intended.
- To make sure that outside influences don't affect the test outcomes, the exam is conducted in a steady and predictable setting.
- When the test is run again, it consistently yields the same predicted results.

When the system or application being tested deviates from the predetermined criteria and behaves differently than expected, the test is considered unsuccessful. Important traits of unsuccessful tests include:

- The automated test encounters errors, exceptions, or unexpected results during execution.
- One or more assertions within the test script are evaluated as false, indicating a mismatch between the actual and expected outcomes.
- Environmental problems including misconfigured systems, shaky network connections, or a lack of resources can lead to failures.
- Modifications to the application code that create defects or unexpected side effects may cause tests to fail (Test Automation Best Practice, n.d.)

Handling Test Results

Generally, comprehensive test reports with information on the progress of individual test cases are produced by automated testing frameworks. Both successful and unsuccessful tests are highlighted in these reports, along with any pertinent error messages or stack traces. By including logging in test scripts, extra information is captured during test execution, which makes it easier to identify and diagnose problems later if a test fails. Debugging tools can be used by developers or testers to go through the code and check variables to determine the core reason for a test failure. Failed tests in CI/CD pipelines can stop new code changes from being deployed to production, ensuring that only stable and dependable code is released. Regression testing is frequently done after problems found in failed tests are fixed to make sure the modifications don't create new ones or interfere with already-existing functionality. To keep the test suite current and functional, update and maintain automated tests regularly to account for changes in the application (Gharagzlou, 2018)

4.4 Analyzing Automated Logs

One of the most important steps in the automated testing process is automated log analysis. Logs offer comprehensive data on the performance of test scripts, which aids in problem identification, comprehension of the execution process, and effective troubleshooting. Here's a how-to guide for automated log analysis during automated testing (What Is Log Analysis? Process, Techniques, and Best Practices, n.d.):

- Make sure that the logging in your automated testing tool or framework is set up correctly. Most automation programs include settings for enabling logging and determining the degree of information to be captured.
- Different levels, such as INFO, DEBUG, WARN, and ERROR, are frequently seen in logs.
- Logging frameworks (e.g., Log4j, Logback, or the built-in logging in languages like Python or Java) are used by many automation programs. Learn the setup options and syntax of the logging framework that is being utilized.
- Examine the logs for important details such as test case names, timestamps, and the order in which the actions were performed. This data aids in comprehending the environment in which the exam is being conducted.
- Observe closely the error messages and stack traces that are displayed in the logs. They frequently provide important details on the type and location of the problem.
- Information about claims or verifications made in the test script may be found in the logs. Examine these signals to determine which section of the test was unsuccessful and why.
- The values of input parameters during test execution may be included in logs if your tests employ parameterized data. This may help pinpoint the precise test data that resulted in an unsuccessful run.
- Link log entries to certain test cases. This aids in focused troubleshooting and helps isolate problems in specific test situations.
- To determine the order of events and the length of the test run, look for timestamps in the logs. This is very helpful in determining performance-related problems.
- Integrate the logging data into your CI/CD tool if you are using automated tests as a component of a pipeline. As a result, you may see logs right from the CI/CD interface.
- Make use of the debugging resources that your automation language or framework offers. While a test is running, these tools may be used to go through the code and check variables.
- Distribute log entries across team members, particularly if more knowledge is required for problem analysis and resolution. Working together can expedite the debugging process considerably.
- Even in cases where tests are passing, it is still advisable to periodically examine logs to spot possible areas for test script optimization or enhancement.

5 CASE STUDY ANALYSIS

Let us consider two e-commerce platforms and perform the UX analysis on both platforms.

1: ElectroTechHub (https://electrotechhub.shop/)

2: FashionFiesta (https://fashionfiesta.shop/)

5.1 ElectroTechHub

ElectroTech Hub keeps a reliable design language, guaranteeing a bound-together client experience across item pages, cart, and checkout. The utilization of a moderate design approach adds to a spotless and current taste, upgrading the overall design. The platform integrates an intuitive navigation menu and a prominent search bar, facilitating seamless exploration. ElectroTech Hub demonstrates strong adherence to discussed points, with alt text for images and well-structured HTML to support screen readers. Integration with popular assistive technologies is successful, providing a positive experience for users with disabilities. Automation tests cover critical functionalities like product search, selection, and checkout, ensuring the reliability of the platform's core features. Automated tests effectively identify and report errors, contributing to a robust error-handling system.

5.2 FashionFiesta

FashionFiesta shows a moderate degree of consistency in design components, with slight varieties across item classifications. While this takes into consideration inventiveness, it might influence the general cognizance of the client's experience. The stage succeeds in visual planning, utilizing dynamic tones, excellent symbolism, and creative design structures. This adds to an outwardly captivating and attractive point of interaction, adjusting greatly to the brand's character. FashionFiesta somewhat conforms to the discussed points, tending to alt text for pictures yet requiring improvement in giving text options to non-text content. While the platform upholds fundamental assistive innovations, there is a requirement for upgraded similarity and enhancement for a more extensive scope of supportive devices. Automation tests on FashionFiesta cover essential functionalities, including product search and selection. However, there is room for expanding the test coverage to ensure comprehensive coverage of the entire user journey.

5.3 Comparison of both Platforms

ElectroTech Hub emerges as a leader in user-centric design and accessibility, setting high standards for competitors. FashionFiesta excels in UI creativity, particularly in presenting visual content, and establishing a strong brand identity. ElectroTech Hub's mobile responsiveness falls short on certain devices, requiring optimization for a more inclusive experience. FashionFiesta lacks certain advanced accessibility features, such as improved keyboard navigation, hindering inclusivity.

There is still need for these two platforms to excel in the areas where they are lacking but the overall analysis results seem satisfactory. There are no strict rules for each and every platform but there are guidelines which have been discussed in the previous chapters and these guidelines help the businesses to grow and make more loyal customers. Acquiring a new customer might not be a challenge for such platforms but retaining that customer in a competitive market is the challenge in the e-commerce world.

6 CONCLUSION

This thesis has undertaken a comprehensive exploration of user interface, accessibility, and automation testing principles within the realm of e-commerce platforms. The journey commenced with an in-depth analysis of software testing theories, encompassing various testing levels and methodologies. The study delved into the core aspects of user experience, emphasizing their significance in the constantly evolving landscape of the software industry.

The integration of accessibility considerations into the design of e-commerce user interfaces emerged as a pivotal element, acknowledging the diverse needs of users and championing inclusivity. The examination of testing framework considerations provided guidance on addressing challenges arising from constraints related to time, budget, and coverage requirements. Furthermore, the capabilities and limitations of prominent automation tools such as Robot Framework, Selenium, and Appium were elucidated through a comprehensive analysis. To demonstrate the practical application of the outlined principles, a case study investigated two e-commerce platforms, revealing both strengths and weaknesses in their respective testing methodologies. This comparative analysis furnishes practitioners with valuable insights to enhance their testing approaches.

In essence, this thesis presents an integrated perspective on the intricate interplay between accessibility, user interface design, and automated testing within the context of e-commerce platforms. The ideas and information conveyed here are intended to serve as a practical guide and knowledge source for practitioners and researchers striving to deliver reliable, user-friendly, and inclusive digital experiences in the dynamic field of e-commerce. The findings underscore the importance of considering accessibility from the inception of design, the strategic application of testing frameworks, and the judicious use of automation tools to foster optimal user experiences in the ever-evolving digital landscape. It is evident from the discussion that competitive market always needs the businesses to be on top of the game by adopting all the recent trends and methods when it comes to user experience. Some of the aspects are less important than others but even insignificant additions to the user experience can be a reason of retaining the customers.

7 REFERENCES

10 Usability Heuristics for User Interface Design. (n.d.). Retrieved from Nielsen Norman Group: https://www.nngroup.com/articles/ten-usability-heuristics/

5 BENEFITS OF AUTOMATED REGRESSION TESTING & BEST PRACTICES. (n.d.). Retrieved from Explicit Success: https://explicitsuccess.com/automated-regression-testing-benefits/

6 Steps Of Automation Testing Process Methodology. (n.d.). Retrieved from Slide Team: https://www.slideteam.net/6-steps-of-automation-testing-process-methodology.html

App & Browser Testing Made Easy. (n.d.). Retrieved from Browser Stack: https://www.browserstack.com/guide/software-testing-strategies-and-approaches

Appium Mobile App Automation . (n.d.). Retrieved from Medium: https://medium.com/automationmaster/appium-mobile-app-automation-406bf8b0fd80

Application Security - Black Box Testing. (n.d.). Retrieved from Imperva: https://www.imperva.com/learn/application-security/black-box-testing/

Automated Regression Testing: A Detailed Guide. (n.d.). Retrieved from Browser Stack: https://www.browserstack.com/guide/automated-regression-testing

Automatic tests in eCommerce: what is their purpose? (n.d.). Retrieved from Divnate: https://www.divante.com/blog/automatic-tests-in-ecommerce-what-is-their-purpose

Bentley, J. E. (2018). Software Testing Fundamentals—Concepts, Roles, and Terminology.

Black Box Testing. (n.d.). Retrieved from Guru 99: https://www.guru99.com/black-box-testing.html

Camacho, R. (2022, 03 17). Software Testing Methodologies Guide: A High-Level Overview. Retrieved from Parasoft: https://www.parasoft.com/blog/software-testing-methodologies-guide-a-high-level-overview/

Cross Browser Testing using Selenium. (n.d.). Retrieved from Guru 99: https://www.guru99.com/cross-browser-testing-using-selenium.html

Cross-Platform & Cross-Browser Testing: The Right Way Towards Quality Software. (n.d.). Retrieved from Test Fort: https://testfort.com/blog/cross-platform-cross-browser-testing-the-right-way-towards-quality-software

Ehmer, M. (2012). A Comparative Study of White Box, Black Box and Grey Box Testing Techniques.

Exploring Advantages and Disadvantages of Automation Testing. (n.d.). Retrieved from The Knowledge Academy: https://www.theknowledgeacademy.com/blog/advantages-and-disadvantages-of-automation-testing/

Gharagzlou, Y. (2018). AUTOMATED TESTING SYSTEMS.

GreyBox Testing. (n.d.). Retrieved from Javatpoint: https://www.javatpoint.com/grey-box-testing

Mobile Automation with Appium. (n.d.). Retrieved from Isha Training Solutions: https://www.ishatrainingsolutions.org/mobile-automation-with-appium/

More Than Usability: The Four Elements of User Experience. (n.d.). Retrieved from UX Matters: https://www.uxmatters.com/mt/archives/2012/04/more-than-usability-the-four-elements-of-user-experience-part-i.php

OTM Testing Automation. (n.d.). Retrieved from FLO Group: https://www.flo-group.com/wp-content/uploads/2023/03/OTM-Testing-Automation.pdf

Regression Testing. (n.d.). Retrieved from EDUCBA: https://www.educba.com/regression-testing/

Robot Framework User Guide. (n.d.). Retrieved from Robot Framework: https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html

Selenium And Mobile Test Automation – Will Selenium Make The Cut? (n.d.). Retrieved from Test Sigma: https://testsigma.com/blog/selenium-and-mobile-test-automation-can-selenium-be-used-for-mobile-testing/

Software Testing Fundamentals. (n.d.). Retrieved from https://softwaretestingfundamentals.com/black-box-testing/

Software Testing Strategies. (n.d.). Retrieved from Geeks for Geeks: https://www.geeksforgeeks.org/software-testing-strategies/

Test Automation Best Practice. (n.d.). Retrieved from Ranorex: https://www.ranorex.com/blog/10-best-practices-test-automation-6-resolve-failing-test-cases/

Test Environment: A Beginner's Guide. (n.d.). Retrieved from Browser Stack: https://www.browserstack.com/guide/what-is-test-environment

Trends in Accessibility in E-Commerce. (n.d.). Retrieved from Medium: https://medium.com/geekculture/the-importance-of-accessibility-in-e-commerce-4592336ea62f

Unit Testing Tutorial – What is, Types & Test Example. (n.d.). Retrieved from Guru 99: https://www.guru99.com/unit-testing-guide.html

Usability 101: Introduction to Usability. (n.d.). Retrieved from Nielsen Norman Group: https://www.nngroup.com/articles/usability-101-introduction-to-usability/

What is a Test Suite & Test Case? (with Examples). (n.d.). Retrieved from Browser Stack: https://www.browserstack.com/guide/what-is-test-suite-and-test-case

What is automated testing? (n.d.). Retrieved from Target Tech: https://www.techtarget.com/searchsoftwarequality/definition/automated-software-testing

What is Automation Testing? (n.d.). Retrieved from Guru 99: https://www.guru99.com/automation-testing.html

What is BLACK Box Testing? Techniques, Types & Example. (n.d.). Retrieved from Guru 99: https://www.guru99.com/black-box-testing.html

What is Integration Testing? Definition, How-to, Examples. (n.d.). Retrieved from Katalon: https://katalon.com/resources-center/blog/integration-testing

What Is Log Analysis? Process, Techniques, and Best Practices. (n.d.). Retrieved from Exabeam: https://www.exabeam.com/explainers/log-management/what-is-log-analysis-process-techniques-and-best-practices/

What is meant by Structural Code Coverage? (n.d.). Retrieved from QA Systems: https://www.qa-systems.com/blog/what-is-meant-by-structural-code-coverage/

What is Regression Testing? (n.d.). Retrieved from Tag Line: https://taglineinfotech.com/what-is-regression-testing/

What is Software Testing | Everything you should know. (n.d.). Retrieved from Software Testing Material: softwaretestingmaterial.com

What Is System Testing – A Ultimate Beginner's Guide. (n.d.). Retrieved from Software Testing Help: https://www.softwaretestinghelp.com/system-testing/

White Box Testing. (n.d.). Retrieved from Imperva: https://www.imperva.com/learn/application-security/white-box-testing/

Why Usability Is Important? (n.d.). Retrieved from Info Beans: https://www.infobeans.com/usability-testing/