VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Malika Tasnim Taky

AUTOMATED TESTING WITH CYPRESS

Technology and Communication
2021

ACKNOWLEDGEMENTS

Thesis writing is a heavy task, and it would have not been possible without the supervision of Dr. Ghodrat Moghadampour, PhD, Senior Lecturer, VAMK; University of Applied Science. I express my sincere gratitude to the supervising teacher for his instructions and guidance on my thesis writing.

My heartiest gratitude is due to Jukka Ruokonen and Timo Laulajainen from Nokia team for their valuable comments on my thesis. I thank them for guiding my work and motivating me. Also, I am thankful to my co-worker Amanda Kauppinen for her contribution to the project and introducing me with an effective working strategy.

I am also deeply grateful to Marita Raja and her team as they have welcomed me at Nokia, made me comfortable to work with them and kept inspiring me always.

My cordial thanks are due to the teachers Seppo Mäkinen, Principal Lecturer, VAMK and Timo Kankaapää, Principal Lecturer, VAMK for their inspiration during my job application and guidance while choosing the thesis topic.

I express my warmest thanks to my family and friends for keep believing in me at my worst and giving me mental support to return from every failure.

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Information Technology

ABSTRACT

Author	Malika Tasnim Taky
Title	Automated Testing with Cypress
Year	2021
Language	English
Pages	40 + 3 Appendices
Name of Supervisor	Dr. Ghodrat Moghadampour

This thesis document describes a practical work completed in Nokia. There, the Roadmap Online (RON) development team was looking for an automated testing solution for an ongoing web project. The team wanted to have a Graphical User Interface (GUI) testing solution to test a web application. The purpose of this test environment was to run the test cases reliably and faster. It also required to test the whole system after each new implementation. Moreover, the testing needed to be straightforward. The team wanted to ensure the performance of the application from the users' perspective. Thus, the testing environment was built with the testing tool, Cypress and more than 100 test cases were implemented.

The tested application has various features. Some features are depending on the other features. Thus, the test cases were divided in two sets: 1) full system test cases and 2) targeted test cases. The full system test cases examined the whole cycle of the application and the targeted test cases were to check specific features.

With the help of numerous built-in commands of Cypress, it was convenient to implement the test cases. Moreover, having the flexibility of building custom commands made the execution simple. Cypress comes with an interactive dashboard named "Test runner" which was very useful to visualize the tests running gradually. The efficiency of testing was 75% since it varied according to the processor speed. Before the application deployment, the test cases were able to find intended bugs. Testing with Cypress was easy and well guided. Cypress has a vast community with developers and professional testers all around the world who are very cooperative. The RON development team was pleased with the solution. It seemed promising to them.

Currently, the application has a fully working automated test environment which reduces the team's effort of testing the GUI side manually. It has test cases for both full system test as well as targeted test.

Keywords Software, system testing, Cypress, full system test, targeted test

CONTENTS

LIST OF FIGURES

LIST OF CODE SNIPPETS

1 INTRODUCTION

Software development is a dynamic and demanding job. Each software development team and company tackle the issue in a different way. However, it does not matter which approach the team uses, it is still possible to know what specific steps the employees follow.

To develop a software, there are few steps. It requires designing, implementation and testing. Developing a software following these steps is a process which is called Software Development Life Cycle (SDLC). SDLC is an infrastructure of planning in details that how a software can be developed, maintained, altered or enhanced. A typical SDLC has seven stages, namely, Planning and Requirement Analysis, Defining Requirements, Design the Production Architecture, Developing the Product, Testing the Product, Deployment in the Market, and Maintenance. /1/

On the other hand, the method where the specification, design, development and testing of a software application are being managed is called Application Life-cycle Management (ALM). It covers the entire life cycle of an application. From designing through to the process of implementation, development, testing, support and ensuring the user experience are supervised in ALM. /2/

ALM is often confused with SDLC, because they both deal with the software development process. The key difference is that SDLC focuses mainly on the implementation process, where ALM deals with the whole life cycle of the application. SDLC, particularly during the stages of development, testing, and implementation, can be considered part of application life cycle management. /3/

In both ALM and SDLC methods, one of the common stages is Testing. Software testing is a method of evaluating the functionality of a software application in order to determine whether the designed software satisfies the specified requirements. Moreover, it is crucial to detect defects to ensure that the product is free of bugs before production. In order to detect bugs, software testing refers to the system of testing using scripts, tools, or any testing automation frameworks. It allows teams to release fault-free and stable applications to the real-world. Moreover, in the early stages of development, it helps teams to find bugs and save time. There are two

ways of testing an application: Manual testing and Automation testing. Manual testing, as the term suggests, is a type of testing where the Tester has to check each element, feature, functionality manually. Contrariwise, Automation Testing is performed with the help of another software. /4/

Investing in manual testing is ineffective and inefficient when it comes to regression testing. Regression testing is the process of rerunning functional and non-functional tests to ensure that previously developed and tested software continues to work after a transition /19/. It is much more rational to program a computer to do the same, instead of expecting humans to repeat same steps with same speed, consistency and energy. Automated testing is a main aspect of continuous integration and delivery. As developers add new features to an application, it is a great way to scale the quality assurance process. There are several automated testing tools to make developers life easier. One of them is Cypress.

Cypress is a modern front-end testing tool for web applications. When testing an application, it covers the main struggle points that developers and QA testers often face. The tool is famous among engineers who use JavaScript frameworks to build applications. Cypress is a locally installed test runner with a dashboard service for recording tests. It is free and open source, released under the MIT licence. Cypress is written in JavaScript. The installation of the software is effortless, and it enables writing test cases frequently while building the application. Cypress makes testing quick and easy. It can test anything that runs in a browser. One of the useful features of the tool is, it takes snapshots as tests run. It has a command log, whereby hovering over, it is possible to see specifically what happened at each step. /5/

In Nokia, the Roadmap Online development team was looking for a reliable solution for testing a web application with automation testing method. The developers aimed to build an automated test environment which would ease the process of testing User Interface before the deployment. Moreover, every time after new implementations, the team wants to ensure both the new and old features are working as they are expected. The team wishes to confirm the application performance from the end user’s perspective. In this case, their requirements matched with Cypress features. This thesis discusses testing, testing with Cypress, installation of the tool and utilizing it to enhance the quality of the web application.

2 SOFTWARE TESTING

Software is a set of instructions or programs that tell a computer what to do. A software may or may not contain bugs. In most cases, software bugs occur due to human mistakes in software design and poor-quality code. A software with bugs causes loss in business revenue. It also declines in customer loyalty and brand reputation. According to ANSI/IEEE 1059 standard, software testing is: /6/

"A process of analyzing a software item to detect the differences between existing and required conditions (i.e., defects) and to evaluate the features of the software item."

In general terms, Software Testing is a method of evaluating the functionality of an application with the goal of determining whether the designed application satisfies the stated requirements and identify bugs to ensure that the application is error-free. /6/

2.1 Importance of Software Testing

Software Testing is a crucial part of Software Development Process. It is impossible to develop an entire software without any defects or bugs. Since engineers are not machines, mistakes may occur. There are high chances of errors in functionality and design in the final code. Deploying a defected software in the market would not be worthy. It may harm the industry in several ways. It is a requirement for carrying out software testing to detect the problems prior to the occurrence in the critical environment.

Next, we discuss some reasons why Software Testing is important.

2.1.1 Cost Effectiveness

Software testing consists of a number of projects. If bugs are found in the early stage, it costs a reduced amount of cash to fix them. Therefore, it is required to get the test completed as soon as possible. To undertake this process, it is beneficial to

invest on experienced testers who carries the professional knowledge in it. They can be a great resource to reap the advantages of a project. /7/

2.1.2 Security

Security of a software is most fragile and sensitive part. There are a variety of cases in which users' data and information are cracked and used for detrimental intentions. The customer cannot be confident that they obtain a quality product if the particular product is not properly tested. After testing, he can be sure that his personal details will be safe. With the assistance of software testing, customers may obtain vulnerability-free products. /7/

2.1.3 Efficiency of Product

The purpose of a product should be serving its customer. As per pledge, it is essential that it carries value to the customers. It should therefore work entirely to ensure an efficient user experience. Moreover, checking the compatibility of the system is needed. For instance, if a developer intends to deploy an application in the market, the compatibility of it must be tested with a wide range of computers and operating systems. /7/

2.1.4 Customer Satisfaction

The primary goal of the product provider is to give customers the highest satisfaction. The fact that it provides the prerequisite and perfect user experience is the reason why it is important to opt for software testing. By providing the best project, it is possible to earn the reputation of reliable clients. Thus, by software testing, company can gain long-term benefits. It is definitely not a simple task to gain trust of consumers, in case the product is found to work and fail one time or another. If one software fails to offer a good impression, customers will find another alternate of it, which will fulfill their requirements. /7/

2.1.5 Development Process Improvement

To avoid reproduction of mistakes, software testing is essential. Varity of errors can be found with the assistance of testing. Finding a bug in the development phase and fixing it immediately is rather easy and effective. Thus, it is highly recommended

that software testers would be working with the developers simultaneously. It is helpful in accelerating the process of growth. /7/

2.1.6 Consistency of Software Performance

An application or program with decreased performance lowers the prestige of the company in the market. Users are not going to trust the company, and it is highly possible that its reputation will suffer. If a company launches a software in the market without testing, and after few days its performance does not match customers' standards or specifications, it would be very hard to convince them in future. Thus, software testing is a better solution for determining a software performance. /7/

2.2 Testing Approaches

For different cases, there are different software testing approaches.

1. White Box Testing
2. Black Box Testing
3. Grey Box Testing

2.2.1 White Box Testing

White Box Testing is a type of software testing in which the tester understands the internal structure, design, implementation of the item being evaluated. To exercise paths through the code, here the tester selects the inputs and decides the required outputs. During the White Box Testing, having the know-how in programming and the implementation knowledge is essential. In this process, testing happens into the system and the tester is aware of both the internal and external behavior. Hence it is called white box testing. /8/

2.2.2 Black Box Testing

Black Box Testing is a type of software testing where the internal structure, design or implementation is unknown to the tester. Such tests may be functional or non-functional, while typically those are functional. Black Box Testing can be used at

almost every stage of software testing: unit, integration, system and acceptance testing. Specification-based research is another name for Black Box Testing.

This approach is named as its program, like a black box in the eyes of a tester, where the program code is invisible. This approach tries to find errors in the following categories:

1. Incorrect functions
2. User Interface errors
3. Data structures or external database access errors
4. Performance errors
5. Initialization and termination errors

For instance, a tester, without understanding a website's internal framework, uses a browser to test the web page. He provides inputs, such as clicks or keystrokes, to check if the outputs against the predicted result. /9/

2.2.3 Grey Box Testing

Grey Box Testing is a hybrid between White Box and Black Box testing techniques. In the White Box Testing, the internal program is known to the tester. Contrary, in Black Box Testing it is unknown. Since Grey Box Testing is a mixture of these two processes, the program structure is partially known to the tester. /10/

2.3 Testing Levels

There are several test levels in Software Testing:

1. Unit Testing
2. Integration Testing
3. System Testing
4. Acceptance Testing

Next, we discuss these in more details.

2.3.1 Unit Testing

Unit testing is the first phase of software testing, where individual software components are tested. It is also familiar by the name of Module Testing or Component Testing. During the creation of an application, through this method developer can assure if an individual unit or module of the application is functioning properly or not. Mostly developers are responsible for doing it in the development environment. /11/

2.3.2 Integration Testing

Integration Testing is the phase of software testing which is combined by both Unit Testing and System Testing. Identifying the faults in the interaction between integrated units is the goal of integration testing. In Integration Testing, test drivers and test stubs are used to assist. It is the next level after Unit Testing. There are primarily two forms of research for integration: 1) Unit Integration Testing and 2) System Integration Testing. /12/

2.3.3 System Testing

System Testing is a form of Black Box Testing. It is also known as End-to-End Testing. By following this testing method, it is possible to test the program on all of the planned target systems. It verifies the desired outputs of every input in the application. This method is commonly used to test the user's experience with the application. /6/

2.3.4 Acceptance Testing

Acceptance tests are the type of tests which perform to determine if a program satisfies its business specifications. The entire program must be operational, with a focus on replicating user patterns. If those targets are not reached, they may also go further, assess the system performance and reject changes. /13/

3 APPLICATION DESCRIPTION

Throughout the world, Nokia has a massive number of products to implement, enhance, manage and maintain. Those are being executed by the teams from different regions and continents. Each team follows its own method of work. Even though the core concepts of presenting the products are same. But, because of assorted working strategy, the presentation varies. Thus, to solve this complicity the Roadmap Online (RON) Development team came up with a better solution. They proposed to build an application which would be convenient for every team to use to handle the products information.

The application is a solution in Nokia for the portfolio of product and its roadmap dialogue process, aka the interlock process. This process is used to update the roadmap materials in global scale. It also

- Defines steps which need to be done before business groups can update roadmaps and communicate them.
- Provides a common roadmap template to product management teams to get a harmonized look and feel for all roadmaps.
- Provides portal for storing roadmaps.
- Provides transparency of proposed changes.
- Gives an opportunity to mitigate issues for customers, thus reducing risk and customer dissatisfaction.

There are over 4000 users around the world for this application. Based on user role, the application enables certain features. People from product management team gets the full access right to the application. In a quarterly update, all proposed changes of product roadmaps will be inserted by creating new interlocks and roadmap drafts. The application will then generate a complete change log comparing the previous and new version of roadmap for the product. Then the roadmap will be set for evaluation where the customer dissatisfaction, positive changes and risks can be noted down. After that, roadmap owner teams and the customer operation teams will discuss to mitigate the risks and set an agreement. In the end, after approvements, the last step is approval and publication of the roadmaps.

4 TEST AUTOMATION TOOL

In this modern software development era, web testing has become a crucial part. The demand for firm and secure test automation tool is raising day by day. There are several testing tools nowadays focusing on resolving developers' expectations and Cypress is one of those tools. It is one of the popular tools which got renowned in no time for its quality and performance.

4.1 Cypress

Cypress is an end-to-end test automation tool for graphical user interface of web applications. The tool is primarily designed to make the lives of developers and quality assurance engineers easier. Developers can easily synchronize and resolve problems with Cypress. Even though it is often compared to Selenium (a portable framework for testing web applications), both are architecturally different.

Cypress uses JavaScript for writing test cases since the tool is built on Node.js. It comes packaged as a *npm* module. Moreover, Cypress has numerous built-in commands for writing tests. Those commands are very convenient and understandable for the users. It also contains jQuery methods to identify UI components and simplify DOM, HTML tress traversal and manipulation. In addition, it simplifies CSS, Ajax and event handlers. /20/

The tool can test anything that can be run on a browser. Other tools may run outside of a browser and execute remote commands. But Cypress engine directly operates inside a browser. Thus, the tester can visualize how users would see. Cypress also provides time travel facility. While the test cases are running, Cypress takes snapshots of individual steps. It is very useful since the tester can hover over each command in the test runner and see the history of actions. /20/

4.2 Setting up the Environment

Cypress is a desktop application which supports the following operating systems:

- macOS 10.9 and above (only 64-bit)
- Linux Ubuntu 12.04 and above, Fedora 21 and Debian 8 (64-bit only)
- Windows 7 and above

The tool is built on Node.js and it uses JavaScript to write test cases. Cypress does not require Node.js for installation but having Node.js in the project makes the work easier. It would then enable more features to use which would not be possible from a direct downloaded application. Also, maintaining and updating the application become easier using *npm*. If there is no Node.js or *npm* in the project, yet Cypress can be installed via downloading it form Cypress CDN link.

In this project, the installation has been done via *npm*. The process has described below in 'Cypress Installation' section.

4.2.1 Node.js

Node.js is a cross platform, runtime environment of JavaScript. It is also an open source which can generate dynamic page content. On the server side, Node.js can create, open, read, write, delete and close files. It is also capable of collecting form data. In Node.js when a file has requested, it sends the task to the file system of the computer and gets ready to handle the next request. When the system has opened and read the file, the server returns the content to the client. Node.js is fast. It eliminates waiting and continues with the next request. /21/

Cypress is built on Node.js but it does not require one to run. If Node.js or *npm* is not available in a project, it is still possible to install the tool via downloading directly from Cypress CDN link. The direct download will always grab the latest version of the application. After the download, the folder is unzipped and double clicking on the application will start it running without needing to install any dependencies.

To utilize the most out of Cypress, however, Node.js can be used. Cypress can be required as a node module from an application under test and run via Node.js. This

is useful when the developer wants access to the test results directly after the run. With this workflow, it is possible to

- Send a notification about failing tests with included screenshot images.
- Rerun a single failing spec file.
- Kick of other scripts

Since the purpose of using Cypress in this project is to generate test report for the GUI, we run the application as a node module. Thus, Node.js needed to be installed. The steps for installing Node.js has described below.

1. Go to the Node.js official website and download the recommended version of it.
2. Run the downloaded installer.
3. It will welcome to the Node.js Setup Wizard – click next.
4. Agree to the terms of license agreement – click next.
5. Then the installer will ask for the installation location. Leave it as default and click next.
6. Now the wizard will prompt to select components to include or remove from the installation. Leave it as default – click next.
7. Finally click on the install button and finish it when the installation is done.

To verify if the installation has done properly, the computer must be restarted, and the following commands are given sequentially:

- *node -v*; This will show the installed version of node.js in the computer.
- *npm -v*; This will show the installed version of *npm* in your computer.

After the successful installation of Node.js, it is possible to initiate *npm* inside the project folder and install Cypress as a node module. To initiate *npm*, the steps are the following:

1. Open the project in an editor. For instance- Visual Studio.
2. Open a terminal in Visual Studio and type *npm init*.

It will then walk through the process of creating a *package.json* file. It only covers the most common items and tries to guess sensible fields. It is also possible to insert the information later in *package.json* file. See Code snippet 1.

```
See `npm help init` for definitive documentation on these fields
and exactly what they do.
Use `npm install <pkg>` afterwards to install a package and save it
as a dependency in the package.json file
Press ^C at any time to quit.
package name: (webgui)
```

Code snippet 1. Initializing npm.

After the initialization, all the inserted data can be found inside *package.json* file.

4.2.2 Cypress Installation

Now that Node.js has been installed and *npm* is available to get third party' packages, the environment is set up to install Cypress. There are two ways of getting the package, *dependency* and *devDependency*.

The difference between above two types is that when a module is required during development, it is installed as a *devDependency*. On the other hand, when a module is required at run time, it is suggested to install as a dependency. In this project, Cypress is needed as *devDependency*. For that, Cypress has installed through the following command:

```
npm install cypress –save-dev
```

After the successful installation, it is noticeable that along with Cypress files and folders, a file named *package-lock.json* has been automatically generated. It is for any operations where *npm* modifies either the *node_modules* tree, or *package.json*. It describes the exact tree that was generated, such that subsequent installs can generate identical trees, regardless of intermediate dependency updates.

Cypress has now been installed to *./node_modules* directory with its binary executable form *./node_modules/bin*. To open from the project root, any one of the below commands can be run on the terminal.

```
./node_modules/.bin/cypress open
```

Or, `npx cypress open`

npx can be run only if *npm* version is greater than 5.2. Now after the installation, Cypress works, and can be open the test runner.

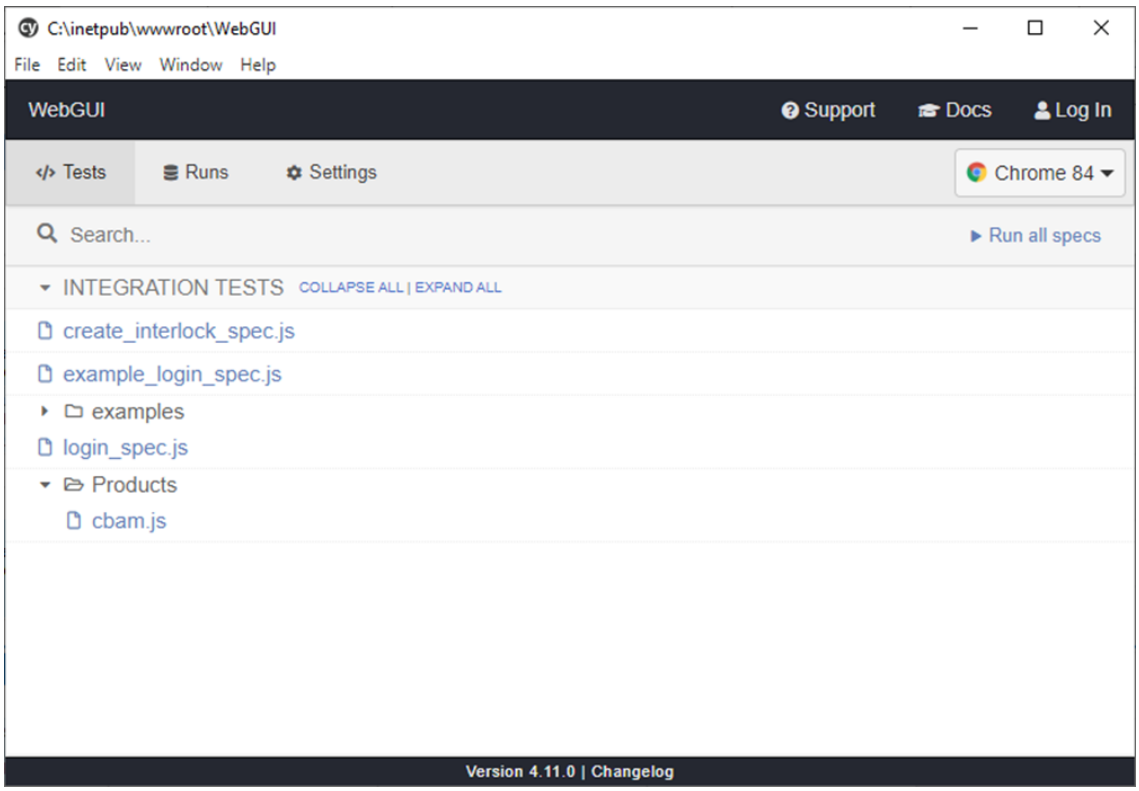

Figure 1. Cypress test runner.

5 TEST ARCHITECTURE / STRATEGY

The architecture of testing is the practice of gazing at a production flow and finding out what, how and when to test to produce the best possible result /14/. It creates a diagram of test activities. Constructing a test strategy before implementation helps to ensure that everyone in the team is on the same page. It visualizes the testing process to all the members. Moreover, if any step is missing, it can be indicated here. In an agile testing strategy, the planner needs to keep account of future implementations as well. Eventually, if users demand for new addition of features, testing of those would also be needed. Hence, the architecture needs to be in such manner that the previous implementations would not require massive change.

Primarily, the testing method of this web application has divided into two parts- Targeted test and Full system test.

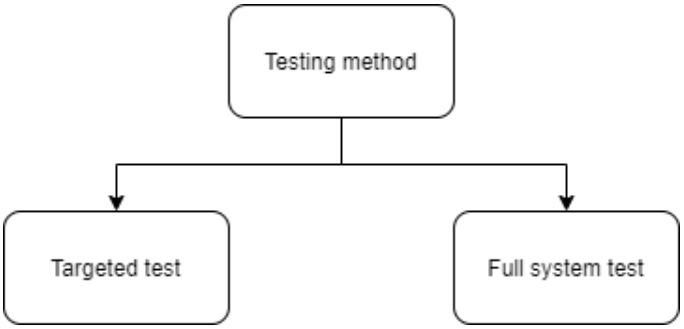

Figure 2. Division of testing method.

5.1 Targeted Test

Targeted test means testing each component or features of the application separately. If a developer implements something new to an existing feature, he would like to be assured that none of the changes made over have caused new bugs. So, to avoid end-to-end testing every time for each new development, the team came up

with such testing solution. This saves time, exclude hassles and response to the request specifically asked for.

5.2 Full System Test

Full system test validates the complete process of the roadmap functionality of the product. It is a chain flow of targeted test cases. From creating an interlock for a product to completing it, adding several contents to its roadmap, as well as the other functionalities, all are being checked in full system test. It is basically an end-to-end testing process for a product interlock.

The Roadmap Online application has the functionality of inserting proposed changes of products by creating new interlocks and roadmap drafts. It requires several steps to complete one interlock creation. The process of creating an interlock to completing it is shown in Figure 3.

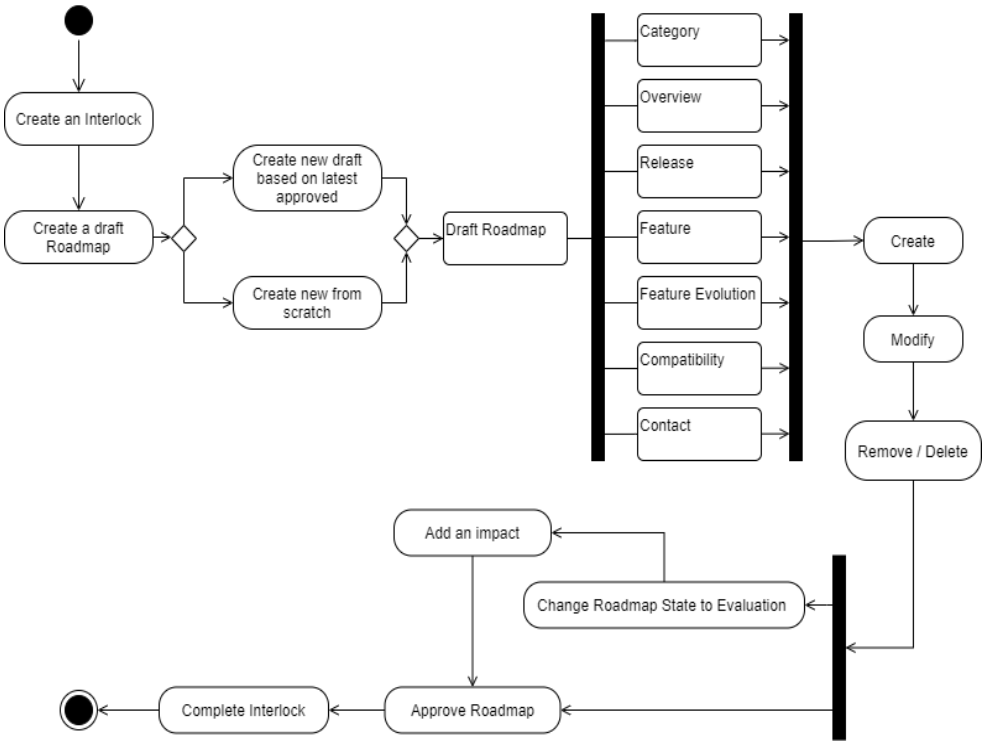

Figure 3. Activity diagram of a product's interlock.

To initiate inserting proposed changes or information for a product, the user needs to create an interlock first. Based on the role, the user retains the right of creation. After successfully creating the interlock, the application enables options for creating a draft roadmap. The draft roadmap is where the user can add product information. There are two options for it, *Create new draft from scratch* and *Create new draft based on latest approved*. *Create new draft from scratch* constructs an empty roadmap. On the other hand, *Create new draft based on latest approved* sends a request to the database to get the data from the previous version of the same product and create a draft roadmap including those data.

A roadmap has different modules for storing different information. Those are (shown in Figure 3.) *Category*, *Overview*, *Release*, *Feature*, *Feature Evolution*, *Compatibility* and *Contact*. All these modules are diverse from each other but have same action functionalities, such as *create*, *modify* and *remove*.

After completing the necessary changes, it is possible to modify the roadmap state. A roadmap has several states. Based on the selection, it enables the next module. For instance, if the state of a roadmap has changed to *Draft for Evaluation*, it will open the option for adding new impacts.

Impact is where the possible risks, positive changes and the customer's dissatisfactions are noted. However, it is not possible to add an impact for a roadmap which has created from scratch. There needs to be a previous roadmap of the same product for comparison. At last, to complete an interlock, the prior task is to approve the roadmap. This is also known as the last state of a roadmap.

In the full system test, the complete chain explained above has been tested with Cypress. With the automated tool, it is much faster to test the system than testing manually. Since the purpose of testing is to check the application process from the user's perspective, Cypress made the task very easy and plays a conducive role in it.

6 TEST TECHNIQUES

Creating and writing test suites for software testing is known as test design. Test architecture and detection of test conditions provide us a general idea for testing that refers to a variety of scenarios. However, when it comes to create a test case, it should be more specific. /18/

In this project, the test files were created based on individual features of the application. Since there are several features, the test files were divided into subfolders. In this manner, the files are organized and easy to look into. Below, the design of folder and file structures are explained.

6.1 Folder Structure

For a better understanding, it is strongly recommended that test files be organized into separate folders. By default, Cypress scaffolds out a suggested folder structure. Though it is fully configurable, in this project the default folder structure has been followed.

After the installation of Cypress in the project, a folder named *cypress* had appeared. Inside it, there were couple of sub-folders with some example **.spec.js* files. **.spec.js* files are basically the files which contain the test cases. Those example **.spec.js* files are good examples of test cases that can be created with Cypress.

The sub-folders inside */cypress* are- *integration*, *support*, *fixture* and *plugin*. Considering the application needs, the *integration* and *support* folders have been utilized mostly.

Since the *integration* and *support* folders are the main focus for this project, they are shown in Figure 4.

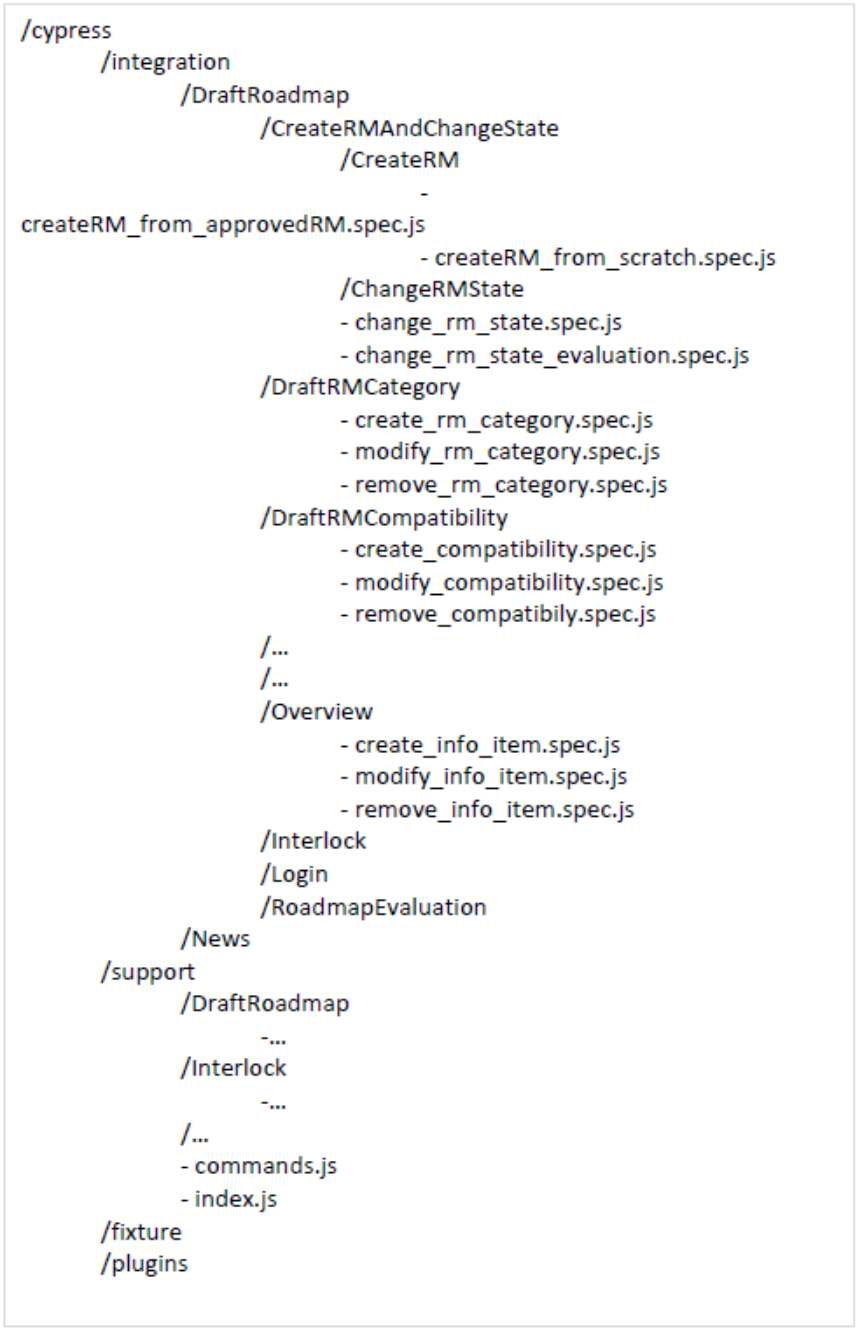

Figure 4. Folder structure for organizing test files.

6.1.1 Integration

This is the folder which contains all the test files *(*.spec.js)*. In this project, the test files have been created based on individual modules and components. However,

storing all those files at once inside Integration would have made finding difficult. Therefore, there are feature based folders, as sub folders which contain the test files. For instance, in *Draft Roadmap*, the user either can create a roadmap from scratch or change an existing state of the roadmap.

The creation of a roadmap has two different ways of doing it; one is creating a roadmap from scratch and the other is creating a roadmap based on the latest approved roadmap. The first option creates an empty roadmap which does not contain any information at the beginning. On the contrary, the second option would copy the contents from the previous roadmap and creates a new one based on it. To have this option available, there must be a completed roadmap of the same product which had created earlier.

In other roadmap features, there are three common actions noticeable: *create*, *modify* and *delete*. These actions are for organizing information of a roadmap. Even though the actions might seem similar, but the modules are distinct. For instance, the module for creating a category is unique than creating a compatibility.

6.1.2 Support

The support folder contains the support file (*cypress/support/index.js*). It runs before the individual **.spec.js* files. It has been done as a convenience mechanism to avoid repetition of importing it into other **.spec.js* files. It is possible to set the initial imported support file to another file or use the *supportFile* configuration to turn it off completely. It is a great place to add reusable actions to all the **.spec.js* files, such as custom commands or global overrides to apply and access. To keep test files organized, it is important to import or request other files from the support file.

Before the **.spec.js* file, Cypress executes the support file. For instance, when a **.spec.js* file is clicked, Cypress executes the *support/index.js* first. But when all the **.spec.js* files are running for full system test, the support file executes once and then continues executing other test files one after another.

6.2 File Structure

The test files have built based on the components. Each module or component has different functionalities to check. However, the similarity among those is in the structure. The following Code snippet is representing the common skeleton.

```
// Define variables
var title = "Add information title";
// Start Cypress tests
describe('Create Roadmap Information Items', () => {
// This is a root level hook; it runs before every test
beforeEach(() => {…});
// Conditional test case
// Check mode: either "full_test" or "targetted" -> set in /cy-
press.json
if(Cypress.env('mode') == "targeted"){
      // Conditional test
      it('Sets up the environment', () => {
            /* Execute the custom command to procced by setting the
            environment for targeted test case */
            cy.ajax_redirect("roadmap_setup");
      });
}
it('Check error messages', () => {…….});  // Failing Test
it('Create two roadmap info items', () => {……}); // Success Test
});
// End Cypress tests
```

Code snippet 2. Primary structure for test files.

Details about the file structure is described in the "Test Case Implementation" chapter.

7 TEST CASE IMPLEMENTATION

In this project, each test script has a common structure which helps to read and write test cases. The structure is divided into three parts: 1) Set up the environment, 2) Check the error messages and 3) Check the successful implementation.

In our code, on the very top of the file, the main function for initiating test cases is called *describe(). describe()* came from a bundle tool called Mocha that Cypress bakes in /15/. In Mocha, *describe()* is used to group the tests. It has the ability to nest tests in as many groups as necessary. *describe()* requires two arguments: a callback function and the name of the test party. /16/

Inside *describe()*, the very first function is *beforeEach()*. It is also borrowed from Mocha. *beforeEach()* is a root level hook. The hook is a functionality that allows to plug into a module to change its actions or response when anything happens /17/. All the conditions which are expected to run before the tests, are defined inside *beforeEach()*.

After that, based on the testing mode, the initial preparation is set under conditional test function. For example, if the tester wants to test a specific feature, the condition will state true, and it will send a request to the database for completing the initial preparation. Since in the application there are features which has dependency on others, this conditional test is needed to continue. Inside *it()* (a function of Mocha), a customized function is called which sends request to the database for imitating values and sets the environment to proceed. On the other hand, if the state false, this function is ignored.

Then, there is a new function initiated which handles all the possible errors that may occur. It checks if the program gives correct error messages for individual mistakes.

At the end, the last test is to check if the successful act on the action requested by the user has been executed. If it fails, it means there is a bug in the development which needs to be fixed. Vice versa, if it passes then it assures that the module is working as expected.

Though most of the test files are following the similar structure, the cases are scripted differently for different functions. The functions which are common and needed to execute several times, are defined as a custom command in the support file (*index.js*).

Next, considering the test file of a feature (*create_info_item.spec.js*) as an example, the steps and commands for testing are described in detail.

7.1 Taking Initial Preparation

Before running these cases for individuals, the first step is to take initial preparation. Each time the viewport, a successful login, respond time of the server need to be defined. Instead of repeating these conditions inside each cases, they are initiated under *beforeEach()* hook, which executes every time before test case runs.

```
// Start Cypress tests
describe('Create Roadmap Information Items', () => {
      beforeEach(() => {
      cy.viewport(1500, 800); // Setting the screen size
      cy.login();
// After programmatically login, need to indicate where to visit
      cy.visit('https://localhost:8080/index.php');
      cy.url().should('include', '/index.php');
      cy.server();
      cy.route({
                  method: 'POST',
                  url: '/includes/ajax_draft_details.php'
      }).as('ajax_draft_details');
});
```

Code snippet 3. Commands under *beforeEach()* hook.

In the above code snippet, inside *beforeEach()* hook, there are few built-in commands visible. Those commands are initiated in Cypress. Below, the commands and their tasks are explained.

cy.viewport(): It is a command to set the screen size. It is useful when tester wants to test the application for different devices. In this manner, the interface can be tested for various screen sizes.

cy.login(): It is a custom command defined in the support file *(/support/index.js)*. It goes through the steps of login and results a successful access to the application.

cy.visit(): It is a built-in command for visiting a local new page or any URL. It redirects to the given site address and continues.

cy.url().should(): It is an assertion. Details about it is described in "Make an assertion" section.

cy.server(); cy.route(): This command is very useful for imitating server response. Here, we have used it to capture the time duration of completing a request. Later we called this duration as waiting.

After the *beforeEach()* hook, there is a conditional test case for setting up the environment. It checks how the environmental variable *mode* is defined. *mode* has initiated in *cypress.json* (the configuration file).

```
// Conditional test case
/* Check mode: either "full_system_test" or "targeted" -> set in
/cypress.json */
if(Cypress.env('mode') == "targeted") {
      // Conditional test
      it('Sets up the environment', () => {
            /* Execute the custom command to procced by setting the
            environment for targeted test case */

            cy.ajax_redirect("roadmap_setup");
      });
}
```

Code snippet 4. Setting up the environment with custom command.

If the tester is testing a targeted test case, then before testing the actual feature, there are some prior actions need to be done. To add a piece of information in a roadmap, the system must have an interlock and an open roadmap for that product. Therefore, before executing the actual test case, the application needs to be set. This setting is done through the conditional test. There is a custom command which completes the actions of setting. If the user changes the mode value to *targeted*, it sends a request to the database for imitating values for setting and then continues to the next step. Contrariwise, if the mode value is *full_system_test*, the condition states false and it skips the stage.

7.2 Taking an Action

Taking an action means commanding Cypress to do a task, for example, querying an element, or clicking an element. There are numerous numbers of commands Cypress provides to execute such tasks. In the next Code snippet, there are some examples.

```
// Failing Test
it('Check error messages', () => {
      cy.get('button#addRMInfoItemOpenModal').should('be.visible')
      .click()
      .get('#addRMInfoModal').should('be.visible').wait(500)
            .within(() => {
            cy.sweetAlert('button#RMInfoAddButton'
            , 'Oops...'
            , 'You forgot to enter all needed data.'
            , 'OK')
      cy.get('input#addRMInfoTitle').should('have.css',  'border',
      '1px solid rgb(255, 0, 0)')
            .type(title).should('have.value', title);
      });
});
```

Code snippet 5. Test case for testing possible error messages.

In the above Code snippet, it is testing the error message when the user clicks on the *Add Information* button without filling the information fields. Here, *cy.get(), cy.click(), cy.type()* are default commands which help to execute the task.

cy.get(): It is a default command in Cypress. It helps to find elements from the testing page. It can be used to find a specific element by mentioning the id (with a hashtag "#") or class (with a dot ".") beside the tag name as an argument.

cy.click(): It is an action which performs like a mouse click.

cy.type(): It acts like keyboard strokes. To type something, this command can be used by passing the value as an argument.

If the code is read out loud, it might sound like this:

1. Get the button with id *addRMInfoItemOpenModal*.

2. Click on it.
3. Then get the modal with id is *addRMInfoModal* and wait for 500ms.
4. Within this modal, execute the custom command, *cy.sweetAlert(),* which would click on the button with id *RMInfoAddButton*. As a result, it will pop up an error message. Check if the message contains correct headline, body text and footer.
5. Then check the styling.

7.3 Making an Assertion

Cypress commands are built to fail if it does not find what is expected. Without adding any assertion, if a test file runs with some action, it will execute accordingly. It will automatically wait and try to complete the action. It is because, Cypress commands have default assertions. Cypress will keep retrying to finish the task within the default waiting time. It will not fail immediately.

However, Cypress gives the user the full control of changing the default values. The tester can add assertions as his convenience. In the following code snippet, we can see an assertion, *.should()* which indicates how the result of the action should be.

```
// After programmatically login, need to indicate where to visit
cy.visit('https://localhost:8080/index.php');
cy.url().should('include', '/index.php');
```

Code snippet 6. Asserting to an action.

In the above Code snippet, it is asked to redirect to the page *https://localhost:8080/index.php*. On the next line, with command *cy.url().should()* we are asserting that the URL should have extension */index.php* included. So, if the visit fails to redirect to the exact page, the assertion will catch the error.

8 TEST RUNNER AND CONFIGURATION IN CYPRESS

One of the benefits of using automation framework for testing is that it provides a test runner to execute test cases. It is one of the essential parts and Cypress is renowned for its easy to execute and well-structured visual test runner. It allows to follow step by step of a test case when it runs. Also, it has a command log which helps to time travel throughout the steps.

In this project, the testing method was divided into two parts, targeted test and full system test. Each of these methods/modes requires different configuration before executing. Therefore, it is essential to configure the runner beforehand. Cypress provides an easy way of doing it. After the installation of framework in the project, it creates a file named *cypress.json*, which allows to configure the default settings provided by Cypress. For this project, the configuration of *cypress.json* is shown and explained below.

```
{
    "baseUrl": "https://localhost",
    "reporter": "mochawesome",
    "reporterOptions": {...},
    "testFiles": [ "Login/login.spec.js",
      "Interlock/create_interlock.spec.js",
      ......
    ],
    "screenshotsFolder": "./mochawesome-report/assets",
    "video": false,
    "env": {
      "mode": "full_system_test",
      "product": "5G RAN",
      "interlockID": "1000007"
    }
}
```

Code snippet 7. Config file, cypress.json.

Cypress offers a range of default configurations for a variety of application resources. In this project, a few of those default values have been overridden. Below, the objects of configuration file are explained.

baseUrl : This setting determines the URL to use as a prefix for the command like *cy.visit().* Since the aim for running these test cases locally, it has been defined as *https://localhost.*

reporter: Cypress enables the option for assigning 3rd party packages as reporter. In this project, Mochawesome has been used for this.

reporterOptions: This option is for customizing the report generating process.

testFiles: By default, Cypress scaffolds the test files in alphabetic order, which is okay for running targeted test cases. But when the full system test runs, the execution needs to follow the order as the application requires. For this purpose, all the test files are organized in order so that the execution follows the correct sequent.

screenshotsFolder: Directs to the path where the screenshots of fail cases should be stored while generating test report.

video: It is possible to capture a video clip of the tests run by enabling this option.

env: Here the environment variables or the global variables are defined. This option is very important since the mode of the testing is defined here. When the mode value is changed to "targeted", the product value also needs to be changed to "GSM". Consequently, it sends a request to the database for setting up the application environment to run targeted test cases. After the set up, then the actual test cases run. On the other hand, for "full_system_test", the value of the product

needs to be changed to “5G RAN” and then the database set the application for executing the end-to-end test.

Now that the configuration file has set up, it is time to run the test cases. To open the Cypress runner the following command needs to be typed in a terminal:

```
npx cypress open
```

It will open the Cypress test runner which shows the test cases defined in *testFiles* in *cypress.json*. A picture of the test runner is shown in Figure 5 below.

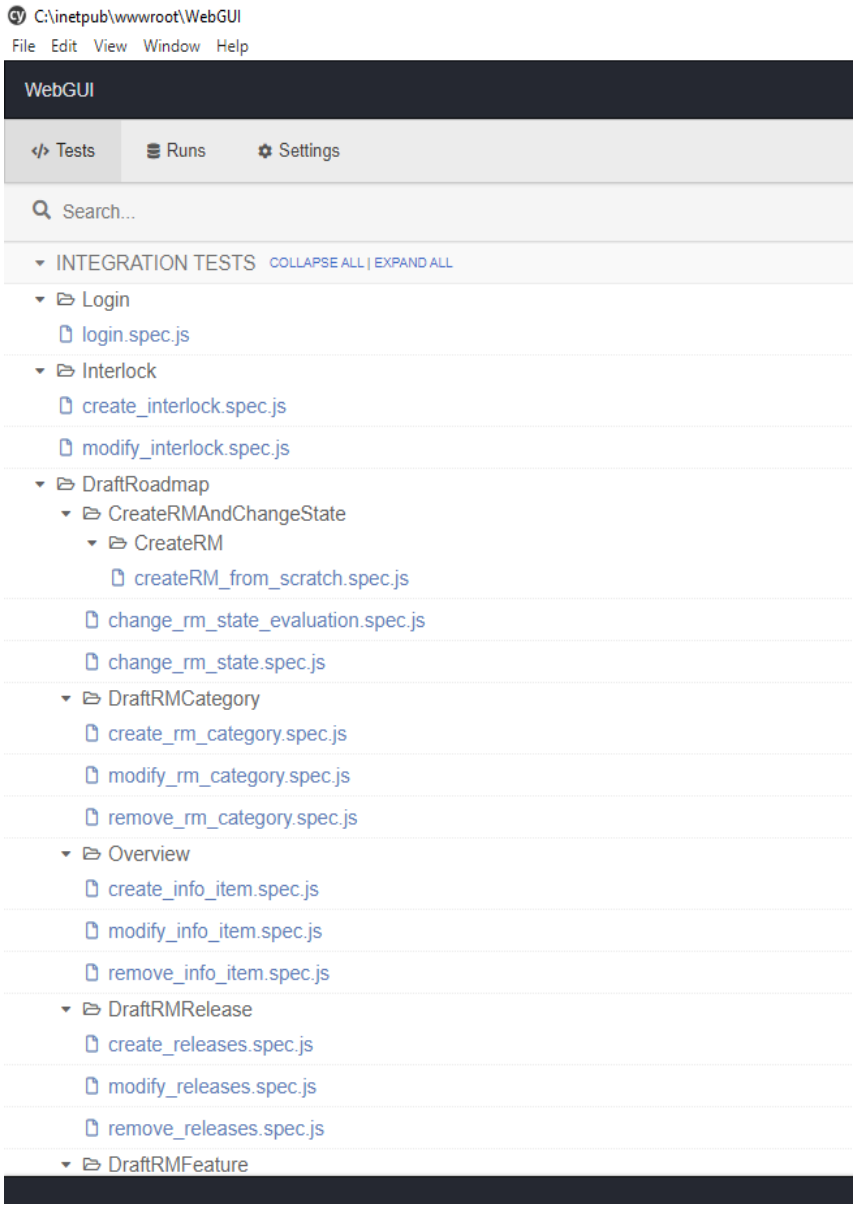

Figure 5. Cypress test runner with test files.

9 CONCLUSIONS

Software testing is a method of evaluating the functionality of an application with the goal of determining whether the developed application meets the stated specifications and identifying errors to ensure that the developed application is flawless. Throughout fundamental software testing, the quality of software can be assured. Software testing can be done in different levels and in different forms. Automated testing can help carrying out testing process more efficiently with less resources. In Nokia, the RON development team was looking for a solution to test the GUI of a web project. They ended up with solving this problem with the help of an external framework named Cypress. Cypress is an open source, free, frontend testing automation tool. In the project, Cypress is installed with Node.js.

The test cases for the project were divided in two parts: 1) targeted test and 2) full system test. The full system test covers end-to-end testing of the whole system. Contrariwise, the targeted test focuses on testing a specific feature. For convenience, test cases are divided in sub-folders and files. However, most of the test cases follow a common structure for testing. Based on the testing mode, it first takes care of initial preparations and then starts implementing test cases to find possible errors in the software. Cypress provides a large number of default commands for taking actions. However, it was also possible to create own custom commands. Each of Cypress action command has a default assertion, which specifies the expected result. The tool also gives the flexibility of overwriting the default values.

Testing with Cypress was trouble-free. Visualizing the tests running in the test runner, shows the user's possible actions on the application. Cypress takes snapshots while it runs which made identifying errors easily and fix the bug instant. The team was pleased with the solution and found it promising to continue.

The possible improvement of the process could be updating the test files with the latest Cypress version. In the Cypress documentation, there are some recommended approaches for getting better results.

The application has dependencies. While running the test cases, testing those components which are built with the help of third-party packages failed. Over that period, to solve the issue, those packages were downloaded and added to the project. But it was not a fruitful solution, especially in the future when the application would get larger. So, to avoid downloading a complete package, it is possible to install only required parts of packages. Moreover, the latest Cypress version has better solutions for network requests, which affects test results.

10 REFERENCES

/1/ Tutorials Point 2014. SDLC – Overview. Accessed 3.2.2021. https://www.tutorialspoint.com/sdlc/sdlc_overview.htm

/2/ Guru99. What is ALM? Application Lifecycle Management. Accessed 13.2.2021. https://www.guru99.com/alm-tutorial.html

/3/ RedHat. What is Application Lifecycle Management (ALM)? Accessed 13.2.2021. https://www.redhat.com/en/topics/devops/what-is-application-lifecycle-management-alm

/4/ Unadkat, Jash. 2020. Beginner's Guide to Software Application Testing. BrowserStack. Accessed 13.2.2021. https://www.browserstack.com/guide/learn-software-application-testing

/5/ Cypress 2021. Why Cypress? Accessed 13.2.2021. https://docs.cypress.io/guides/overview/why-cypress.html#Cypress-in-the-Real-World

/6/ Rajkumar. 2021. What is Software Testing | Everything You Should Know? Software Testing Material. Accessed 14.2.2021. https://www.softwaretestingmaterial.com/software-testing/

/7/ Parthiban, Pradeep. 2019. 7 Reasons Why Software Testing is Important. Indium Software. Accessed 14.2.2021. https://www.indiumsoftware.com/blog/why-software-testing/

/8/ Software Testing Fundamentals 2020. White Box Testing. Accessed 14.2.2021. https://softwaretestingfundamentals.com/white-box-testing/

/9/ Software Testing Fundamentals 2020. Black Box Testing. Accessed 14.2.2021. https://softwaretestingfundamentals.com/black-box-testing/

/10/ Software Testing Fundamentals 2020. Grey Box Testing. Accessed 14.2.2021. https://softwaretestingfundamentals.com/gray-box-testing/

/11/ Rajkumar. 2019. Unit Testing Guide | Software Testing Material. Software Testing Material. Accessed 14.2.2021. https://www.softwaretesting-material.com/unit-testing/

/12/ Software Testing Fundamentals 2020. Integration Testing. Accessed 14.2.2021. https://www.softwaretestingmaterial.com/integration-testing/

/13/ Pittet, Sten. The different types of Software Testing. Atlassian CI/CD. Accessed 14.2.2021. https://www.atlassian.com/continuous-delivery/software-testing/types-of-software-testing/

/14/ SQC. Test Architecture. Accessed 21.2.2021. https://www.sqc.co.uk/perform/assurance/test-architecture/

/15/ Cypress 2021. Writing Your First Test. Accessed 23.2.2021. https://docs.cypress.io/guides/getting-started/writing-your-first-test.html#Write-your-first-test

/16/ Morelli, Brandon. 2017. How to test JavaScript with Mocha -The Basics. Code Burst. Accessed 23.2.2021. https://codeburst.io/how-to-test-javascript-with-mocha-the-basics-80132324752e#:~:text=describe()%20is%20simply%20a,second%20is%20a%20callback%20function.

/17/ Micha. 2009. What is meant by the term “hook” in programming. Stack Overflow. Accessed 23.2.2021. https://stackoverflow.com/questions/467557/what-is-meant-by-the-term-hook-in-programming

/18/ Try QA. What is Test design? Or How to specify test cases? Accessed 27.2.2021. http://tryqa.com/what-is-test-design-or-how-to-specify-test-cases/

/19/ Wikipedia 2021. Regression Testing. Accessed 27.3.2021. https://en.wikipedia.org/wiki/Regression_testing

/20/ Khetarpal, Aashish. 2020. What is Cypress: Introduction and Architecture. Tools QA. Accessed 22.3.2021. https://www.toolsqa.com/cypress/what-is-cypress/

/21/ Node.js 2021. Introduction to Node.js. Accessed 29.3.2021. https://nodejs.dev/learn/introduction-to-nodejs