User Stories applied for end-to-end web testing

HUMAID MOLLAH, University of Twente, The Netherlands

1 ABSTRACT

It is essential to test web applications in order to verify customer requirements. Since customer requirements are written as user stories during agile software development, we have developed a user-story-driven approach for writing and implementing end-to-end test cases. Initially, we conduct a literature review to identify how user stories can be transformed into end-to-end test cases. Following that, we analyze real-world web applications to determine how these test cases can be implemented efficiently. Subsequently, we interview software testers to discuss the findings and acquire insights into end-to-end testing. Finally, we present a systematic method to develop and implement end-to-end test cases with the help of user stories. The automatic generation of test cases from user stories has also been examined briefly.

2 KEYWORDS

Agile software development, user stories, end-to-end tests, web application, front-end, back-end

3 INTRODUCTION

The use of web applications in the sectors of commerce and communications has made them an important and fairly large part of the software industry [12]. With the growth of companies and firms that offer these services, efficient testing of such systems has become a major requirement. Today, most software organizations follow an agile development methodology which is an iterative approach to delivering projects to customers at a rapid pace [19]. User stories play a key role in writing customer requirements [24]. A user story describes the functionality of the web application [19], and the success of this software is based on user story satisfaction [24]. User stories can be best tested with the help of end-to-end tests. The reason why end-to-end tests are used in verifying user stories over other testing practices is that most modern web applications adopt a multi-tier architecture [18]. This means that the application framework is divided into multiple layers, like the database server and the application server [5]. The goal of end-to-end tests is to test the application as a whole and detect the deviation of the software from expected behavior [18]. Such tests can examine both the front-end (client-side) and back-end (server-side) of a web application simultaneously. This research will study how user stories can be used to write and implement end-to-end tests for a web application.

Software developers and testers face a variety of challenges when it comes to testing web applications [26]. Firstly, the usage of multiple programming languages for the implementation of different components calls for different validation techniques. Secondly, a distributed multi-tier architecture as discussed before can make it difficult to determine the cause of failure. Finally, because the development of some big web applications involves a large and complex codebase, it is very hard for testers to understand the code. To meet these challenges, the correct definition of test cases for testing individual components of the web application altogether is necessary. Test cases should be written in a way that they cover a large part of the underlying application so that faults can be discovered. This study will identify how testers can define test cases and implement them efficiently.

There exists a big gap between writing user stories and writing end-to-end tests in the development process of a web application. User stories are written before the development of a feature or functionality while end-to-end tests are written only when the feature is fully developed and ready for production [19]. This research will help fill this gap such that end-to-end test cases can be developed in parallel with user stories so that customer requirements are verified during the entire development process of the web application.

The primary objective of this research is to devise a user story-driven approach to writing and implementing end-to-end test cases for a web application. To do this, initially, a literature review has been conducted to identify the best practices and techniques for developing end-to-end test cases from user stories. Subsequently, a case study has been carried out on a sample of two real-world web applications to determine how test cases can be implemented efficiently. Finally, testers of a software development agency were interviewed to gain insights into the selection of user stories and the generation of end-to-end test cases.

4 RESEARCH GOALS

This study aims to produce a user-story-driven approach for writing and implementing end-to-end test cases for a web application in an agile development environment. This research answers the following questions :

(1) How can user stories be used to formulate end-to-end test cases for a web application?
(2) How can end-to-end test cases be implemented efficiently to identify potential errors in a web application?

5 RELATED WORKS

There have been related papers in the area of test case generation from user stories and use cases. For example, Jim Heumann (2021) has proposed a method to identify use-case scenarios for a web application by introducing the concept of *flow of events* [13]. Another example is from Massod et al.,(2017) who used a Selenium tool to automatically generate test cases from user stories [16]. This approach uses a tool to write user stories using a restricted set of keywords and rules, and test cases are generated based on these parameters. Similarly, Allala et al. (2019) have used Natural Language Processing

TScIT 37, July 8, 2022, Enschede, The Netherlands

to transform user requirements, written as use cases or user stories, into test cases [25]. There has also been a study by Chopade and Dhavase (2017) about how user stories can be treated positively or negatively for testing purposes [24]. A positive user story reflects positive or accepting actions, while a negative user story reflects negative or forbidden actions.

However, these studies are not focused on end-to-end test case generation and the efficient implementation of these test cases has not been explored in detail. This paper fills the research gap by effectively capturing how user stories can be used to write and implement end-to-end tests for a web application.

6 LITERATURE REVIEW

End-to-end testing is a software testing technique that is used to test the application workflow from the beginning to the end. These tests are necessary because they help in determining various dependencies of the web application as well as ensuring that the correct information is communicated between the different components of the application [1,2,27]. End-to-end tests are mainly used to keep track of user workflows in the web application [9]. A **workflow** is a series of processes or actions that the user performs from initiation to completion. An example of a workflow could be searching for a hotel followed by booking the hotel. Another example is creating an account (signing up) followed by logging into your account and then performing another task. The following section discusses the best practices for writing end-to-end tests for a web application such that the most important workflows can be tested. The generation of end-to-end test cases from user stories will be discussed in section 6.2. This section presents a 3 step procedure for identifying workflows, creating use-case scenarios, and writing test cases by referring to user stories. In section 6.3, the automatic generation of these test cases has been visited. Finally, in section 6.4, the implementation of these end-to-end test cases is explored.

6.1 Best practices for End-to-End Testing

Many articles discuss the best practices for writing end-to-end tests [1,2,9,19,27]. Some of these practices can be applied in our research and should be kept in mind when formulating test cases by the method proposed in section 6.2. These best practices for writing end-to-end tests are as follows:

(1) *Focus on the product's most important workflow*: The most important workflows of your web application should be tested first. For example, for Booking.com, the most important workflow is searching and reserving a place. Therefore, this workflow and all user stories related to this workflow should be tested in the beginning.
(2) *Large Workflows should be broken down*: Large workflows in the application should be broken down into smaller tests. This is because it is hard to monitor bigger tests and find errors in a large workflow as compared to smaller ones.
(3) *Avoid Low-Level test cases*: Test cases should be added as long as they add value to the user story which is being tested [19]. For example, if payment through a bank card is being tested, it is not necessary to write a test case to confirm whether charges are not applied on an expired VISA card if a test for expired Master-Cards has already been written. Such tests are covered by low-level Unit tests.
(4) *Build test cases for all possible workflows*: An optimal product must be tested for all possible interactions and micro-interactions that a user might have with the product. Therefore, we must test as many workflows as possible. Section 6.2 discusses how test cases can be formulated for these workflows.

6.2 Generation of test cases from User Stories

In agile software development, user stories define software requirements[19]. They demonstrate a sequence of actions performed by the system to provide an observable result of value to the user [16]. User stories are not written in much detail [19,25]. However, they tell the customer what to expect, a developer what to code, and a tester what to test [25]. Table 1 shows a template of a basic user story along with an example.

Template	Example
As a <type of user>	As a user
I want <some goal>	I want to login to the webapp
so that <reason> (optional)	so that I can create a blog

Table 1. Template of a basic User story.

A test case is a scenario in the web application which may or may not be associated with a set of data inputs and tells the developer or tester what should be the expected result of this scenario. Test cases help determine whether the software satisfies a particular requirement [13,25]. Various papers present different techniques and models to generate test cases from use cases and user stories [13,16,19,25]. The application of user stories for the development of end-to-end acceptance tests has been discussed extensively in the book written by Mike Cohn [19]. The conversion of use case scenarios into test cases has been studied comprehensively by Jim Heumann [13]. Some researchers have also developed an automated model-driven approach for the generation of test cases from use cases [16,25]. Using the knowledge gained from these papers, a 3 step process can be defined to identify end-to-end test cases for a web application. The procedure will be explained by referring to the *User creates a Blog* example presented in Table 1. The steps are as follows:

(1) *Identify Flow*
(2) *Develop Use-Case Scenarios*
(3) *Write Test cases*

6.2.1 *Identify Flow.*

The first step to generate test cases from user stories is to identify the flow of events in a user story. The flow of an event consists of 2 parts: Basic Flow and Alternate Flow [13]. The **Basic Flow** covers the flow of events that happen "normally".The **Alternate Flow** of events refer to the "optional" or "exceptional" behavior of that user story. Alternate flows can be considered as an alternative route to the basic flow. There can be more than one alternate flows for a user story. Figure 1 shows a representation of the basic flow and alternate flows in the use case of a web application.

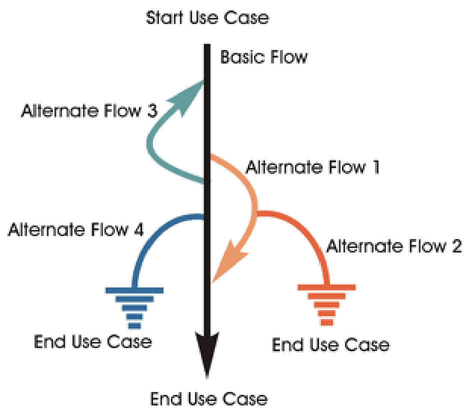


Fig. 1. Basic Flow and Alternate flows of a Use-case [10]

From the example in Table 1, the following flows can be elaborated:

Basic Flow:

(1) Login : User accesses the web-app, system asks for user ID and password, user is authenticated.
(2) Create a Blog: The system displays the homepage, the user clicks on the button "Create a Blog", and the system displays the text field of the Blog form.
(3) Submit Blog: The user writes text in the blog form, the user presses "Submit Blog", and the system displays the created blog on the user's homepage.

Alternate Flow 1: Unidentified user : Invalid user ID or password entered for Login.
Alternate Flow 2: Empty Blog : Blog text-field is empty.
Alternate Flow 3: Server error : The server is not running.

6.2.2 ***Develop Use-Case Scenarios***. The next step in this process is to define use-case scenarios for a user story. **Use-case scenarios** define a complete path for that user story [13,16,19,25]. The users of the web application can follow multiple paths to perform this user story. These can include, for example, just the basic flow, the basic flow plus alternative flow 1, or basic flow plus alternative flow 1 and 2 [13]. In theory, many combinations of flows are possible. However, the most important ones should be chosen following the *Best practices for End-to-End Testing* in section 6.1. Table 2 shows an example of some of the important use-case scenarios based on the example from Table 1.

Scenario Name	Starting Flow	Alternate
User successfully creates a blog	Basic Flow	
Unidentified user	Basic Flow	1
Empty blog	Basic Flow	2
Server error	Basic Flow	3

Table 2. Use-Case Scenarios for *User creates blog* example

6.2.3 ***Write Test cases***. The final step of this process is to formulate test cases based on the use-case scenarios we have developed in the previous section. To do this, a template can be used as shown in Table 3. This template is used to represent the expected results and data inputs for the use-case scenarios. Each scenario has a minimum of 1 test case but there can be more. The test case template can be built by following the 2 steps stated below.

1. ***Identify Expected Results***: The first step in writing test cases is to determine the expected result of each use-case scenario. The expected results describe how the system should behave for their respective use-case scenarios. For example, the expected result of the use-case scenario *Unidentified student* (see Table 2), can be determined as a *Login Error*. Test cases when implemented should assert this behavior of the web app.

2. ***Identify Data Inputs***: The second and final step of the process is to determine the different fields which are required to test these use-case scenarios, and to choose the test inputs/values for these fields. For the *User creates blog* example, 3 fields can be determined from the Basic Flow: student ID, password, and the blog text field. The following strategies can be used for choosing test inputs :

(1) *Boundary Testing* : Test inputs are chosen using boundary values [22]. For example, if the passing marks for an examination are 50 percent, the boundary values to test would be 49 and 50. In this manner, both valid and invalid boundaries are tested.
(2) *Random Testing* : Test inputs are chosen at random [8].
(3) *Partition Testing*: The set of inputs can be divided or partitioned into separate domains according to a self-defined rule, and test inputs are chosen from these subdomains [29].
(4) *Usage-based Testing* : Select test inputs based on usage of the web application [3].

The table below shows the test case template for the *User creates blog* example. The test inputs have been chosen by following the *Usage-based testing* approach In the table, "x" refers to an unused input, "null" refers to a field left blank, "blog text" refers to a valid blog text input, and "invalid" here is used to represent an invalid password input.

Scenario Name	I1	I2	I3	Expected Result
User successfully creates a blog	s123	abc123	blog text	Display blog
Unidentified user	s123	invalid	x	Login Error
Unidentified user	null	abc123	x	Login Error
Empty Blog	s123	abc123	null	Alert User
Server error	s123	abc123	x	Server Error

Table 3. Test cases for *User creates blog* example.
[Data inputs: I1: student id, I2: password, I3: blog text field.]

6.3 Automatic Test Case Generation

Literature suggests that 2 tools have been developed which automate the process of generating test cases from user stories [16,25]. In these papers, the automatic generation of test cases is referred to as Model-Based Testing. In both of these tools, a template or a meta model has to be created for every user story by following a strict set of rules and keywords. These models have to be provided with all information about the functionality that is being tested such as the name of the endpoint, the input fields, the name of the button which has to be pressed, the page that you should be redirected to, and in some cases the HTTP status codes for the action. Following these approaches, a tool to generate test cases from the method discussed in the previous sections can be developed.

6.4 Implementation of End-to-End Tests

The last step in the process is to implement the test cases developed in the previous sections. Literature suggests various techniques for the automation of end-to-end web testing [18]. These techniques can be broadly divided into 2 categories namely Capture Replay Web Testing and Programmable Web Testing. **Capture Replay Web Testing** refers to recording actions performed by a user on the web app (capture) and automatically executing the same actions (replay) which repeat the mouse movements and key-presses performed by the user [14,18]. **Programmable Web Testing** on the other hand uses test scripts to simulate the actions performed by a user with the help of specific testing frameworks [18]. The former is not a preferred option in our case for 3 reasons. Firstly, Capture Replay Web Testing is difficult to maintain and is not reusable [14]. Secondly, we cannot perform *Random Testing* by using this technique. Lastly, this technique cannot be used to test invisible web elements. Therefore, we will use the latter, that is Programmable Web Testing, for the implementation and automatic execution of end-to-end test cases.

Programmable Web Testing makes use of web elements such as input fields, links, buttons, etc for test case execution. There are 3 methods to localize these web elements:

(1) *DOM-Based* : Locate web page elements using information contained in the Document-Object-Model (DOM). A DOM is a programming interface for documents used on the web. It allows programs to change their structure, style, and content [4]. By using this approach, we can locate an element by its tag name or an attribute by its ID, etc. This technique requires good development practices specifically good naming conventions when it comes to writing code for the client-side (front-end) so that web elements can be identified by a unique identifier.
(2) *Coordinate-Based* : Locate web elements by recording coordinates of a web page. This technique produces very fragile test scripts and is therefore considered obsolete [18].
(3) *Visual-Based* : Locate web elements using image recognition to control GUI components. This technique requires the tester to make images of web elements so that parts of the web page can be located by checking for similarity with the web element. This can be a very lengthy process.

Literature suggests that the *Programmable DOM-Based* approach is the best option for implementing end-to-end tests. There are various reasons to support this claim [18]. Firstly, DOM-based test suites do not require much time for development. Secondly, the evolution of test suites can be done quickly and without much effort. Here, the word "evolution" refers to updating test suites when application requirements and functionality change. Thirdly, DOM-Based locators are proven to be more robust than Visual locators. Lastly, DOM-based approaches using tools such as the Selenium Web Driver and Cypress offer a comprehensive programming interface for the implementation of test cases. In Section 7, end-to-end tests are implemented for 3 web applications using the *Selenium Web Driver*. The usage of this tool is quite straightforward and can be found in the *Documentation of the Selenium Web Driver* [28].

7 CASE STUDY

This section analyzes the effectiveness of the method studied in the previous section by developing end-to-end test cases and implementing them using the Selenium Web driver. A case study has been conducted on a sample of 2 deployed real-world web applications. The web applications that have been chosen belong to different application domains. For each of these projects, 1 user story has been chosen to develop and implement end-to-end tests for the most important workflows. Table 4 gives a short description of the web applications chosen for this case study.

Name	Description
MyDay	Scheduling application
VRM	Monitoring application

Table 4. Web applications used for Case Study

For each web application, the test cases are implemented with the help of Python's built-in unit testing framework. These test cases have been implemented using web drivers for the latest browsers such as Chrome, Firefox, and Safari. All the test cases have been developed by following the best practices [23]. For example, while using the Selenium web driver, the Page-Object-Model (POM) has been used. POM is a design pattern in Selenium that stores all the web elements in an object repository. Moreover, if IDs have been used in HTML tags, they are used as the primary web element locators. Only if IDs are not provided to the element, other locators such as the CSS class name and XPath locators used. A code snippet of the implementation of a basic Login end-to-end test can be found in Appendix A.

7.1 Case 1: MyDay: Scheduling Web application

MyDay is an appointment scheduling application designed to connect sports professionals with their clients. Users can either be clients or professionals. The application API is built mainly using PHP and the front-end of the application is built with Vue which is a JavaScript framework. We choose the following user story for this case study as it represents the main workflow of the application (see 6.1 (1)) :

User story: As a professional, I should be able to schedule a session with a customer.

7.1.1 ***Development of End-to-End Test Cases :*** As discussed in the literature review, we first identify the Basic Flow and Alternate Flows from the user story. Thereafter, based on the basic and alternate flows, we work out the use-case scenarios. Finally, we develop the test cases by identifying the expected results and test inputs for each use-case scenario.

Basic Flow :

(1) Login: Professional accesses MyDay web app, professional clicks "Login", the system asks for email and password, the professional is authenticated and redirected to MyDay Professional homepage.
(2) Create an event: Professional clicks "Create Event", the system displays the drop-down menu, Professional clicks "Create an event with Client", and the system displays session form.
(3) Submit event: Professional fills session form, Professional clicks "save", the system displays the created event on the professional's homepage.

Alternate Flow 1 : Unidentified Professional : Invalid user email or password entered for Login.
Alternate Flow 2 : Invalid date field : Invalid (past) date chosen for scheduling an event..
Alternate Flow 3 : No client chosen : No clients selected for creating an event.
Alternate Flow 4 : Invalid client email/mobile number : Invalid email or mobile number of the client is entered to create an event.

Scenario Name	Starting Flow	Alternate
Event scheduled successfully	Basic Flow	
Unidentified professional	Basic Flow	1
Invalid date-field	Basic Flow	2
No clients chosen	Basic Flow	3
Invalid Client email/mobile	Basic Flow	4
Invalid date-field and no clients chosen	Basic Flow	2,3
Invalid date-field and invalid client email/mobile	Basic Flow	2,4

Table 5. **Use-case Scenarios** for *Professional creates event* example

Scenario Name	Test Inputs	Expected Result
Event scheduled successfully	i1,i2	Display event
Unidentified professional	i1,i2	Login error
Invalid date-field	i1,i2,i3,i4,i5,i6	Alert user
No clients chosen	i1,i2,i3,i4,i5,i6	Alert user
Invalid Client email/mobile	i1,i2,i3,i4,i5,i6	Alert user
Invalid date-field and no clients chosen	i1,i2,i3,i4,i5,i6	Alert user
Invalid date-field and invalid client email/mobile	i1,i2,i3,i4,i5,i6	Alert user

Table 6. **Test cases** for *Professional creates event* example.
[Data inputs - i1: email, i2: password, i3: date, i4: client email, i5: client mobile number, i6: client selection form]

Test inputs for this case study are chosen by applying *Random testing* and *Boundary testing*
Random Testing: A randomly generated input for the email and password should produce an error and a randomly generated client email/mobile number should alert the user.
Boundary Testing: An invalid (past) date for the date field should alert the user while a valid date should produce no errors, and an empty client field should alert the user but a correctly entered client field should produce no errors.

7.1.2 ***Implementation of End-to-End Test Cases :***

Test cases have been implemented for a real user (professional) of the MyDay web application. The Basic Flow test case (*Event Scheduled successfully*) was implemented in the beginning. This test case was divided into 2 parts namely the *Login* and *Create/Submit an Event* (see section 6.1 part 2). Implementation of the *Login* end-to-end test can be seen in Appendix A. The *creation of an event* workflow was also implemented in a similar manner. In both cases, the user actions involve clicking or filling in a few fields and submitting a form. Once the Basic flow test case was implemented, the development of tests entailing the Alternate flows was prompt. These test cases were developed from the initial Basic Flow test case by changing the data inputs and assertions.

7.1.3 ***Results :***

4 out of 7 test cases that were defined did not produce the expected results. These test cases were related to the Alternate Flows : *Invalid date-field* and *Invalid Client email/mobile*. The following bugs were discovered: 1. A professional can create an event with an invalid client email/phone number. 2. A professional can create an event on an invalid date.

7.2 Case 2: VRM: Remote Monitoring Web application

VRM is a remote monitoring web application that allows users to control energy systems such as freezers, solar chargers, and water tanks remotely. This system performs real-time data collection on energy devices and displays this to the user. The application API is built using PHP and the front-end of the application is built with JavaScript, Vue, and Less. Considering the main workflow of the web application involves monitoring devices, we choose the following user stories to test (see 6.1(1)). These user stories can be summarized as reading information from the device : *User story: As a user, I want to see:*

(1) *when my tank level was last updated*
(2) *temperature was last updated*
(3) *sum of solar power of all my installations*
(4) *my device's alarms*

7.2.1 ***Development of End-to-End Test Cases :***

Once again, as discussed in section 6.2, we first identify the Basic and Alternate flows. In this case study, the *flow of events* has been generalized for *reading information from the device*. Therefore, in the second step, we formulate use case scenarios such that all the above user stories can be tested together. Finally, we write down test cases by identifying the expected results and test inputs.

Basic Flow :

(1) Login: Login to VRM application, the system displays all the device installations
(2) User chooses device: The user chooses a device by clicking the device name, the system displays the device dashboard.
(3) User navigates to a page to see device related information : User clicks on one of the buttons in the Dashboard to navigate to a device page, system displays the required device information.

Alternate Flow 1 : Unidentified User : Invalid user email or password entered for Login.

Alternate Flow 2 : Server error : The server is not running.
Alternate Flow 3 : Device error : Device not connected or data cannot be read from the device.

Scenario Name	Starting Flow	Alternate
User can see device info	Basic Flow	
Unidentified user	Basic Flow	1
Server error	Basic Flow	2
Device error	Basic Flow	3

Table 7. **Use-case Scenarios** for *Device monitoring* example

Scenario Name	Test Inputs	Expected Result
User can see device info	i1,i2	Display data
Unidentified user	i1,i2	Login error
Server error	i1,i2	Notify user
Device error	i1,i2	Notify user

Table 8. **Test cases** for *Device Monitoring* example.
[Data inputs - i1: email, i2: password]

Test inputs for this case study were chosen by applying ***Boundary Testing***. This was used to assert that a valid email and password do indeed login a user into his account, while an invalid email or password produces a Login error.

7.2.2 *Implementation of End-to-End Test Cases :*

The Basic Flow test case (*User can see device info*) was implemented to assert whether the device information was indeed displayed to the user. This test case checks for a valid Login from the user and various button presses on the application to navigate you to the correct page so that the device information is displayed to the user (as stated in the user story). These test cases have been implemented by checking whether the web elements corresponding to the device information are present on the web app (by using their ID tags) and asserting if the data they contain is not None/null. For example, in case of the *alarm logs* user story, a table which shows these logs and the information in each column such as *device name,time started at* and *cleared after* is confirmed. The *Device error* and *Server error* test cases were implemented in a similar manner but the difference was that assertions had to be made to check for *notifications* such as such as *Device is not online* or the *connection was lost.*

7.2.3 *Results :*

All test cases passed to assert that the device information as stated in the user story is visible to the user. Notifications on the application were also tested to inform the user about a Server or Device error.

8 INTERVIEWS

To gain more knowledge regarding end-to-end test case generation and implementation, 3 testers from the software development agency *El Nino* were interviewed. The testers were asked to provide their feedback on the methods discussed in the literature and give their insights on how to *Determine important functionality to test*, *Develop test cases scenarios from user stories* and *Implement end-to-end test cases*. Each tester was asked a set of questions related to these steps. The information acquired in these interviews has been summarized in the following sections (8.1, 8.2, 8.3). In section 8.4, we report the key points which have been used to develop the final method. The transcripts of these interviews can be found in Appendix B.

8.1 *Determine important functionality to test*

The respondents mentioned that the important functionality of a web app can be identified from the product backlog. This is where the most important user stories for the current iteration of the project are marked. For example, a priority queue in Gitlab can be used where the most critical features and functionalities are denoted. However, they are not always sorted so there can be multiple user stories that can be used. In an agile working environment, the product manager is responsible for prioritizing user stories that need to be tested. Therefore, discussions should be made with the product owner to identify which functionality to test. Regarding the test cases which should be omitted, the testers agreed that *low-level test cases* can be avoided while writing end-to-end test cases. One of the testers said, "Indeed, user stories which are tested in unit tests can be skipped." These user stories are related to small bug fixes or features that may have a very low impact on the project.

8.2 *Develop test cases scenarios from user stories*

The respondents could compare the concepts of *Basic and Alternate Flows* to their own techniques for identifying test cases. One of the testers said, "I call the basic flow the happy path." Another tester explained, "alternative flows are flows that are likely to be used apart from the regular flow." The respondents mentioned that the number of Alternate flows, according to the technique described, would depend on the feature or functionality being tested. Since a lot of the alternate flows are covered via Unit tests, the Alternate flow test cases should be used to test broader "exceptional" scenarios where both the front-end and back-end of the web application are used. These would also depend on the "likelihood of breaking" as mentioned by one of the respondents. Moreover, one tester said, "80/20 principle usually does apply though.". This was a reference to the Pareto principle which states that 80% of the consequences (alternate flows) come from 20% of the causes (parts of code or a feature) [6]. Regarding the choice of data inputs, the testers mentioned that *Boundary testing* was a good approach to test both positive and negative scenarios. One of the testers said, "*Usage-based Testing* could be considered as the most commonly used option as the choice of data inputs depends on the feature being tested"

8.3 *Implement end-to-end test cases*

All the testers agreed that *Selenium* and *Cypress* are the best approaches for implementing end-to-end test cases. One of the testers said, "These are the easiest options to implement end-to-end tests". The other options, such as *Visual-based locators*, are not particularly applicable for testing big web applications where many different scenarios have to be considered.

8.4 *Interview Results*

The key points that were identified in the interview process are as follows. These points have been incorporated into the final method which is presented in the next section.

(1) The most important functionality of the web application is subject to the current iteration in the development of the project These can be found in the product backlog where the most important user stories are marked with tags or are in the priority queue.
(2) User stories are prioritized by the Product owner, who should always be consulted if it is not clear which functionality is most important.
(3) Low-level test cases such as bug fixes and those which have a very low impact on the project workflow can be omitted.
(4) The number of Alternate flows depends on the feature or functionality being tested. They should be targeted at broader scenarios involving both the client and server sides.
(5) An 80/20 Pareto rule is applicable in determining Alternate flows. This indicates that most or 80% of alternate flows are caused by a small part or 20% of the code or a feature/functionality [6].
(6) Data inputs for test cases are commonly chosen using *Usage-based testing* and *Boundary testing.*

9 METHOD

From the *literature review*, *case study* and *interviews*, a user story-driven approach to writing and implementing end-to-end test cases has been formulated. This approach can be divided into 3 parts:

(1) *Select User Story*
(2) *Transform User Story into Test Cases*
(3) *Implement Test Cases*

9.1 Select User Story

The first step in the process is to select a User story. The following points should be contemplated in the selection process :

(1) ***Identify important functionality*** : The user story which defines the most important functionality of the web application should be chosen first (see 6.1 (1)). In an agile development environment, the *Product Owner* is responsible for defining user stories and prioritizing important functionality for each iteration. Often, these user stories contain tags such as "important" or "critical." These can help identify the important user stories for which end-to-end test cases should be written.
(2) ***Omit Low-Level User Stories***: User stories that define requirements that are covered by Unit tests should be omitted (see 6.1 (3)). These user stories can be related to small bug fixes or a small feature update that does not have much impact on the web app.

9.2 Transform User Story into Test Cases

The second step in the process is to transform the user story into End-to-End Test Cases. This process can be broadly divided into 3 steps. Figure 2 shows a flowchart of this process.

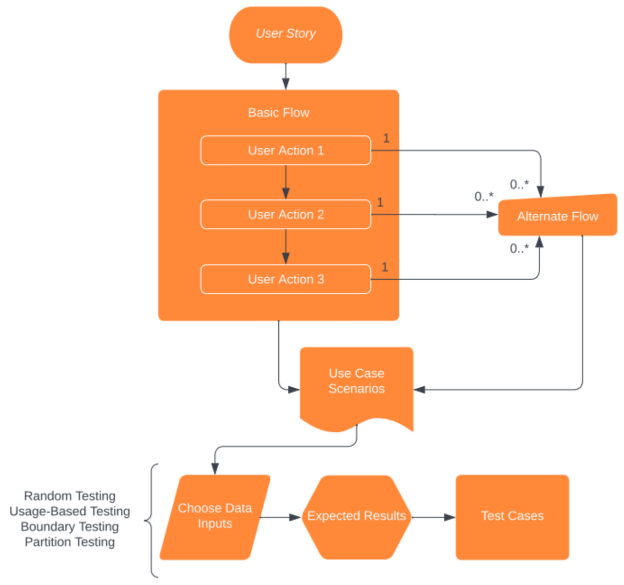


Fig. 2. Transform User stories into End-to-End Test Cases

(1) ***Identify Flow***: As discussed in section 6.2.1, In the beginning, identify the *Basic flow* of the user story. This can usually be divided into smaller workflows or *user actions* (see section 6.1 (2)). The case study discussed in section 7 shows how the *Basic Flow* of a user story can be identified and broken down into *user actions.* Next, Identify *Alternate flows* of this user story by analyzing each *user action* defined in the Basic workflow. There can be 0 or many alternate flows for each user action. (see section 6.1 (4)). This depends on the feature or functionality being tested. It is important to note that there can be many alternate flows related to one user action as compared to the others (see section 8.4 (5)).
(2) ***Develop Use-Case Scenarios***: Based on the Basic and Alternate workflows, the different paths that could be followed by the user should be worked out. A table can be used as shown in section 6.2.2 to keep track of each scenario. Use case scenarios should be worked out for complex cases involving multiple Alternate flows. To do this, Alternate flows should be combined if possible (see section 7.1.1).
(3) ***Write Test cases***: Identify the expected result and data inputs required for each use-case scenario. Data inputs should be chosen by one of the methods discussed in section 6.2.3 (2). Based on the case study and interviews, *Random Testing*,*Boundary Testing* and *Usage-Based testing* are the most commonly used options.

9.3 Implement Test Cases

The test cases developed in the previous step can be implemented using a Programmable DOM Based approach using tools such as the Selenium Web Driver or Cypress (see section 6.4). From the

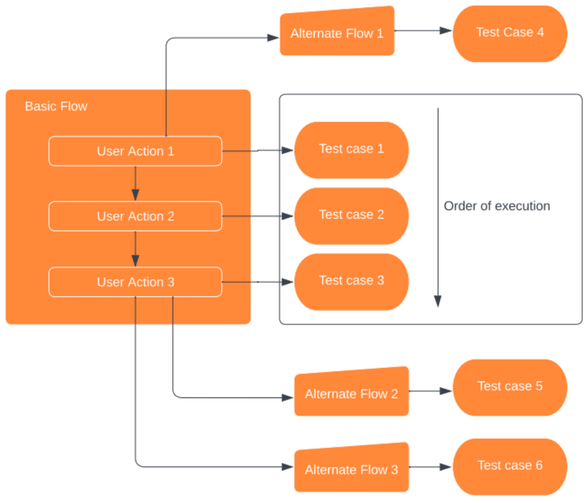


Fig. 3. Implementation of test cases by following Basic and Alternate Flows

experiments conducted in section 7, a systematic approach to implementing these test cases is defined. This process can be broadly divided into 2 steps. Figure 3 shows a flowchart of this process.

(1) ***Basic Flow Test Cases*** : The *Basic Flow* test case should be implemented in the beginning. Each small workflow or *user action* in the basic flow should be implemented as an individual test case. These test cases are executed in the same order as the user actions. Figure 3 shows how the Basic Flow test case is carried out. This is done by implementing test cases 1,2 and 3 corresponding to the user actions which are then executed in that order.

(2) ***Alternate Flow Test Cases*** : A test case containing an *Alternate Flow* is implemented such that it asserts *exceptional* behaviour only for the *user-action* it is connected with. These test cases can then be executed along with one or more basic flow test cases developed in the previous step. For example, in Figure 4, the test for Alternate Flow 2 can be carried out by executing the test cases in orders 1,2, and 5. This is done so that user actions 1 and 2 can be performed *normally* before *exceptional* behavior can be asserted in test case 5. However, these test cases can also be executed independently. For example, in Figure 4, it can be seen that the test for Alternate Flow 1 can be carried out by executing test case 4 independently because is connected to *user-action* 1.

10 CONCLUSION

In order to effectively verify customer requirements during agile software development, we have developed a user-story-driven approach for writing and implementing end-to-end test cases for a web application. A three-step method has been put together which helps identify the most important use case scenarios and implement them as test cases. The two research questions that were posed are specifically addressed by this method. The first research question is answered in the second step of this method namely *Transform User Story into Test Cases*. This step describes a systematic method by which a user story can be transformed into end-to-end test cases. The second research question is answered together through the first and third steps of the method, namely *Select User Story* and *Implement Test Cases*. The first step discusses how the most important user stories can be selected for the testing process, and the third step presents an efficient solution for implementing end-to-end test cases.

There are a few *limitations* that can be identified in this research. Firstly, due to the lack of time, a tool to automate the process of transforming user stories into test cases could not be developed. Therefore, only a literature review was conducted to identify how this process can be automated. Secondly, although the described method shows positive results in the case study, it cannot be guaranteed that the use of the method will produce 100% accuracy. This is because, theoretically, it is impossible to determine all alternate flows of a user story and bugs in software. To address the first limitation, *future work* should be targeted at automating the second step in the method, namely *Transform User Story into Test Cases*. This will make developing end-to-end test cases very economical and time-saving.

11 REFERENCES

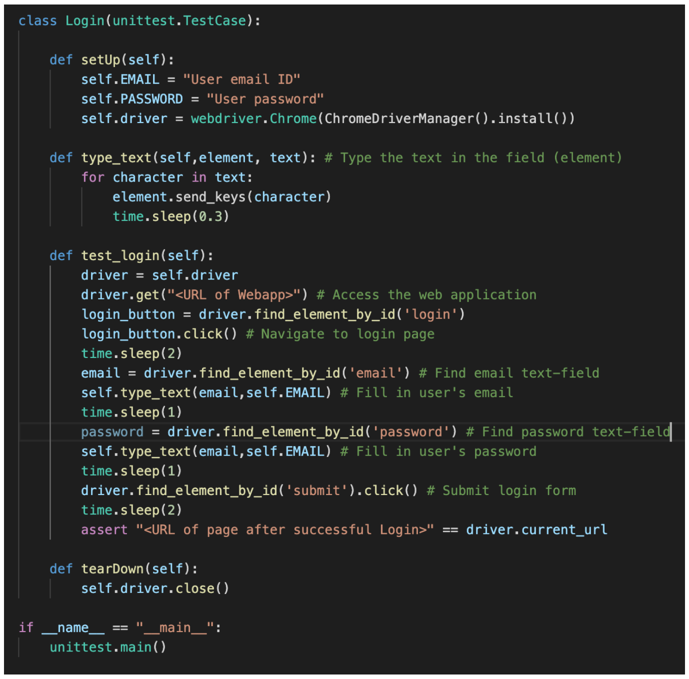
[1] Apostolov, A., and B. Vandiver. 2014. End to End testing-What should you know?. In 2014 67th Annual Conference for Protective Relay Engineers (pp. 125-131). IEEE
[2] Bai, Xiaoying and Tsai, Wei-Tek and Paul, Ray and Shen, Techeng and Li, Bing. 2001. Distributed end-to-end testing management. In Proceedings Fifth IEEE International Enterprise Distributed Object Computing Conference (pp. 140-151). IEEE.
[3] Björn Regnell, Per Runeson and Claes Wohlin. 2000. Towards integration of use case modelling and usage-based testing. Journal of Systems and Software, 50(2), 117-130.
[4] Brucker, Achim D and Herzberg, Michael. 2018. A formal model of the Document Object Model.
[5] Diao, Yixin and Hellerstein, Joseph L and Parekh, Sujay and Shaikh, Hidayatullah and Surendra, Maheswaran. 2006. Controlling quality of service in multi-tier web applications. 26th IEEE International Conference on Distributed Computing Systems (ICDCS'06) (pp. 25-25). IEEE.
[6] Dunford, Rosie and Su, Quanrong and Tamang, Ekraj. 2014. The pareto principle.
[7] Glenford J. Myers, Corey Sandler, and Tom Badgett. 2011. The Art of Software Testing. John Wiley Sons.
[8] Godefroid and Patrice. 2007. Random testing for security: blackbox vs. whitebox fuzzing. In Proceedings of the 2nd international workshop on Random testing: co-located with the 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE 2007) (pp. 1-1).
[9] Gundecha, Unmesh and Avasarala, Satya. 2018. Selenium webdriver 3 practical guide: End-to-end automation testing for web and mobile browsers with selenium webdriver. Packt Publishing Ltd.
[10] Gutierrez, Javier and Escalona, M.J. and Mejías, M. and Torres,

Jesús. 2006. Generating Test Cases from Sequences of Use Cases.. 473-476.
[11] Huang, George Q, and Mak, Kai-Ling. 2001. Issues in the development and implementation of web applications for product design and manufacture. International Journal of Computer Integrated Manufacturing, 14(1), 125-135.
[12] Jeff Offutt. 2002. Quality Attributes of Web Software Applications. IEEE Softw. 19, 2 (March 2002), 25–32.
[13] Jim Heumann. 2001. Generating Test Cases From Use Cases. Requirements Management Evangelist Rational Software.
[14] Leotta, Maurizio and Clerissi, Diego and Ricca, Filippo and Tonella, Paolo. 2013. Capture-replay vs. programmable web testing: An empirical assessment during test case evolution. In 2013 20th Working Conference on Reverse Engineering (WCRE) (pp. 272-281). IEEE.
[15] Lucassen, Garm and Dalpiaz, Fabiano and van der Werf, Jan Martijn EM and Brinkkemper, Sjaak. 2016. The use and effectiveness of user stories in practice. International working conference on requirements engineering: Foundation for software quality (pp. 205-222). Springer, Cham.
[16] Massod, Mahawish and Iqbal, Muhammad and Khan, M. and Azam, Farooque. 2017. Automated-User-Story-Driven-Approach-for-Web-Based-Functional-Testing. International Journal of Computer and Information Sciences. 11.
[17] Matt Wynne and Aslak Hellesoy. 2012. The Cucumber Book: Behaviour-Driven Development for Testers and Developers. Pragmatic Bookshelf.
[18] Maurizio Leotta and Diego Clerissi and Filippo Ricca and Paolo Tonella. 2016. Approaches and Tools for Automated End-to-End Web Testing. Adv. Comput., 101, 193-237.
[19] Mike Cohn. 2004. User Stories Applied: For Agile Software Development. Addison Wesley Longman Publishing Co., Inc., USA.
[20] Mikowski, Michael and Powell, Josh. 2013. Single page web applications: JavaScript end-to-end. Simon and Schuster.
[21] Paul C. Jorgensen. 2018. Software Testing. CRC Press.
[22] Ramachandran and Muthu. 2003. Testing software components using boundary value analysis. In 2003 Proceedings 29th Euromicro Conference (pp. 94-98). IEEE.
[23] Ramya, Paruchuri and Sindhura, Vemuri and Sagar, P Vidya. 2017. Testing using selenium web driver. In 2017 Second International Conference on Electrical, Computer and Communication Technologies (ICECCT) (pp. 1-7). IEEE.
[24] Rupali M. Chopade and Nikhil S. Dhavase. 2017. Agile software development: Positive and negative user stories. 2017 2nd International Conference for Convergence in Technology (I2CT), 297-299.
[25] Sai Chaithra Allala and Juan P. Sotomayor and Dionny Santiago and Tariq M. King and Peter J. Clarke. 2019. Towards Transforming User Requirements to Test Cases Using MDE and NLP. 2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC), 2, 350-355.
[26] Sreedevi Sampath and Sara Sprenkle. 2014. Advances in Web Application Testing. Adv. Comput., 101, 155-191.
[27] Tsai, Wei-Tek and Bai, Xiaoying and Paul, Ray and Shao, Weiguang and Agarwal, Vishal. 2001. End-to-end integration testing design. 25th Annual International Computer Software and Applications Conference. COMPSAC 2001 (pp. 166-171). IEEE.
[28] WebDriver | Selenium. Retrieved July 2, 2022 from https://www.selenium.dev/documentation/webdriver/
[29] Weyuker, Elaine J and Jeng, Bingchiang. 1991. Analyzing partition testing strategies. IEEE transactions on software engineering, 17(7), 703.

12 APPENDIX

12.1 APPENDIX A

The following is a code snippet of a basic login end-to-end test case implemented using Python and the Selenium web driver.



```
class Login(unittest.TestCase):

    def setUp(self):
        self.EMAIL = "User email ID"
        self.PASSWORD = "User password"
        self.driver = webdriver.Chrome(ChromeDriverManager().install())

    def type_text(self,element, text): # Type the text in the field (element)
        for character in text:
            element.send_keys(character)
            time.sleep(0.3)

    def test_login(self):
        driver = self.driver
        driver.get("<URL of Webapp>") # Access the web application
        login_button = driver.find_element_by_id('login')
        login_button.click() # Navigate to login page
        time.sleep(2)
        email = driver.find_element_by_id('email') # Find email text-field
        self.type_text(email,self.EMAIL) # Fill in user's email
        time.sleep(1)
        password = driver.find_element_by_id('password') # Find password text-field
        self.type_text(email,self.EMAIL) # Fill in user's password
        time.sleep(1)
        driver.find_element_by_id('submit').click() # Submit login form
        time.sleep(2)
        assert "<URL of page after successful Login>" == driver.current_url

    def tearDown(self):
        self.driver.close()

if __name__ == "__main__":
    unittest.main()
```

Fig. 4. Implementation of a basic Login end-to-end test using the Selenium web driver and Python

12.2 APPENDIX B

Interview Questions:

Select User Story

(1) How do you identify the most important functionality of a web application to test?
(2) In which cases do you not test a certain feature/functionality of a web application?

Transform User Story into Test Cases

(1) Does the concept of Basic and alternate flows help identify the most important use case scenarios? Is there a different technique that you use to identify the workflow of an app?
(2) How many exceptional/abnormal scenarios do you consider when testing a feature/functionality (referring to alternate flows)? How do you identify them?
(3) How many complex use case scenarios (with multiple alternate flows) should you test?

(4) How do you choose data inputs for your test cases? Which methods do you use?

Implement Test Cases

(1) The case study uses tools such as the Selenium Web-Driver and Cypress to implement end-to-end tests. Would you consider these as usable approaches or are there any other tools that you use?

Interview Transcripts:

Software Tester 1:
Select User Story

(1) Read issue title, look at priority queue in GitLab (not always sorted), discuss with PO's.
(2) Small bug fixes, textual changes, basically determine low-risk low impact issues and skip them.

Transform User Story into Test Cases

(1) Yes, I call the basic flow the happy path.
(2) It highly depends, bigger issues have more alternative flows and therewith abnormal ones
(3) Again, this depends. Customer expectations, available budgets, used hours on an estimate, and the likelihood of breaking. Also, input validation is easy to automatically test, i.e. it also depends on what "should you test" means (manual or automatic).
(4) I choose data inputs using all of these methods (Random, Usage-based, Boundary, Partition)

Implement Test Cases

(1) I use Cypress but Selenium also works similarly

Software Tester 2:
Select User Story

(1) Check the product backlog, choose the one with high priority, it is usually not very difficult to determine
(2) Indeed, user stories that are tested in unit tests can be skipped

Transform User Story into Test Cases

(1) Yes, my definition for alternate/abnormal flows: abnormal flows are flows that will not likely be used by end-users, alternative flows are flows that are likely to be used apart from the regular flow
(2) Depends on the functionality you are testing. The 80/20 principle usually does apply though
(3) for end-to-end tests, the broader use-case scenarios which involve both the front-end and back-end, so that big features or flows can be tested well
(4) Usage-based Testing could be considered the most commonly used option as the choice of data inputs depends on the feature being tested

Implement Test Cases

(1) Yes, These are the easiest options to implement end-to-end tests.

Software Tester 3:
Select User Story

(1) The product owner can help choose the most important one, but on GitLab also you can find the most important user stories for the sprint
(2) Bug fixes and maybe small changes in the front end can be avoided

Transform User Story into Test Cases

(1) Yes, the basic workflow should be recognized and the alternate flows.
(2) Depends on the user story you are testing, the bigger functionalities will have more alternate flows
(3) This depends on the budget and what is required really
(4) All options are usable and it depends on what you are testing, but Boundary Testing is a good technique to test both positive and negative scenarios.

Implement Test Cases

(1) Yes, these are the ones which are most used, the other ones as you mentioned like Visual locators cannot be always used like for big web apps