International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:11, No:1, 2017

Automated User Story Driven Approach for Web-Based Functional Testing

Mahawish Masud, Muhammad Iqbal, M. U. Khan, Farooque Azam

***Abstract*—**Manual writing of test cases from functional requirements is a time-consuming task. Such test cases are not only difficult to write but are also challenging to maintain. Test cases can be drawn from the functional requirements that are expressed in natural language. However, manual test case generation is inefficient and subject to errors. In this paper, we have presented a systematic procedure that could automatically derive test cases from user stories. The user stories are specified in a restricted natural language using a well-defined template. We have also presented a detailed methodology for writing our test ready user stories. Our tool "Test-o-Matic" automatically generates the test cases by processing the restricted user stories. The generated test cases are executed by using open source Selenium IDE. We evaluate our approach on a case study, which is an open source web based application. Effectiveness of our approach is evaluated by seeding faults in the open source case study using known mutation operators. Results show that the test case generation from restricted user stories is a viable approach for automated testing of web applications

***Keywords*—**Automated testing, natural language, user story modeling, software engineering, software testing, test case specification, transformation and automation, user story, web application testing.

I. INTRODUCTION

A remarkable spread of web applications into the areas of commerce and communications has made web applications a significant and fairly larger part of the software industry [1]. Web applications and services will change the landscape of IT architecture, by providing a new means of service delivery for enterprises. As predicted by Hagel and Brown [2], companies are outsourcing or purchasing their services over the internet. The economic significance of web applications also increases the importance of controlling and improving the quality of web applications [3]-[5]. Web applications therefore require through and systematic test approaches for testing.

A common approach for testing web applications is by manually writing record-and-replay test cases and executing them in a test execution tool, such as Selenium or HPE Unified Functional Testing. Manual system level testing of web applications from requirements can be a significant challenge. Manually writing test cases from requirements is not only time consuming, but also difficult to maintain when the requirements change [6]. Moreover, the tester has to write the test cases manually.

When testing is done manually then the tester has to write the test cases covering most or all of the features of the system depending on the available resources, such as time and manpower. Moreover, instead of following a thorough testing procedure, the tester might consider exploring the user interface of the application and cover as many features as possible. The success of such a manual testing approach largely depends on the expertise, domain knowledge, and enthusiasm of the tester [7]. A lack of these factors might lead to poor testing of the application. As the software evolves, it becomes a difficult task to maintain the traceability of requirements to test cases. Some requirement changes may result in test cases becoming obsolete and because of the lack of traceability, identifying such test cases is labor intensive.

Our aim in this paper is to automate the test case generation process from requirements expressed through user stories in natural language. The intension to employ user stories for test generation is twofold: (i) It is a prevalent practice in industry for gathering the functional requirements of web based applications from user's point of view (ii) User stories are written in natural language, which makes it easier for the user to express, authenticate and prioritize their requirements. It is focused on the result that the user wants to achieve, and in the language that he/she understands. This helps to understand user requirements clearly and in short span of time. Moreover, the user stories outline functionality that is required from a system without getting into lengthy details. The user may like to accept the user story or delay it for development at a later stage, without wasting much time on gathering lengthy and detailed requirement's first. By postponing details for the later stage, the user stories can help well in understanding of problem domain.

The presented approach of test generation from user stories utilizes the requirements written in user stories. The user stories are to be written as per our suggested template. The template uses a restricted language that allows the user stories to be processed for test generation. The approach has been applied on an open source case study, Moodle, which is a course management system written in PHP. The evaluation has been done by using known mutation operators of HTML and PHP. Results show that the approach was able to detect all the 195 seeded equivalent non-equivalent mutants.

Even though the testable user stories may seem to be tightly coupled to the graphical interface, this is not a major obstacle in its practical application. A common practice in web application testing is to write test cases in record-and-replay tools that are coupled with graphical interface (GUI). In case of a change in GUI, the entire test cases (potentially in hundreds) relating to the change have to be manually

Mahawish Masud is a Student at CE&ME, National University of Sciences and Technology (NUST). Pakistan (e-mail: mahawish.masud@nust.edu.pk)
Dr. Farooque Azam, is a Professor at CE&ME, NUST.

modified. In our case, the modification is simpler as the user only needs to change a few user stories rather than the test cases. Overall, the contributions of this paper are: (i) we suggest an approach for writing user stories in a format that allows automated test generation; (ii) we have developed a test generation strategy to obtain test sequences and test data from the user stories; (iii) the approach is successfully applied on an open source case study.

The rest of the paper has been organized as: Section II explains the related work. Section III explains the proposed approach. Section IV explains evaluation of our technique, and in Section V, we conclude our work.

II. Related Work

There are works that derive test cases from requirements expressed in natural language, by employing natural language processing (NLP) techniques, but most of these techniques do not fully support automatic test case generation [8]; for instance, Zhang et al. [7] and Sarmiento et al. [9] generate the test cases from natural language without test data. Escalona et al. [10] can generate test cases but need manual intrusion for providing input values to the generated test cases. Nevertheless, there are techniques available in literature [6] that have used NLP for test case automation and also support the test data generation, but they use the domain models and OCL constraints to generate the test data and invariably help in the test case generation process. All these approaches are contrary to our approach in the sense that we want to fully automate the test case generation from requirements expressed in natural language and through user stories. We also tend to avoid dealing with the domain models. Hametner et al. [11] and Carvalho et al. [12] have also generated the test cases from requirements with the help of restricted grammar and dictionary, which is quite similar to our approach too, but the dictionary used in their approach is subject to change for every project, and the restricted grammar is not able to fully express the requirements of various systems. Unlike their approach, we tend to use our approach flexible enough to be used on different kinds of systems.

Literature suggests that user stories are a much better way of gathering user requirements [13]-[15]. For web applications, writing requirements through user stories have shown to be much more practical [13], [16] since user stories are written in natural language and simple enough to be understandable by both the user and the domain experts. To investigate, if literature suggests any approaches that have used the user stories for the test case automation, it was found out that Cucumber testing frame work [16]-[18] suggests writing the user stories in a natural language; however, it involves writing the user stories using a particular language called “Gherkin”. However, “Gherkin” is a very inflexible camouflaged form of Ruby, which apparently allows user to write user story in a syntax which is close enough to English language, but it is not English [19], [20]. Such a technique is not feasible in our case, because primarily it does not support the automatic test case generation right from the requirements and secondly, and we want to keep the user stories simple and flexible enough to be understandable by both user and domain experts.

To summarize, various techniques are available in literature that do support automated test case generation from requirements that are expressed in natural language. However, issue with almost all of the techniques is that either they require certain level of manual intervention for their correction and completion or they involve complex models. Such approaches are contrary to our aim, which is to automatic test case generation from requirements given in natural language through user stories.

III. User story based test generation approach

Our approach mainly focuses on derivation of automatic test cases from testable user stories. The approach consists of two parts: (i) A user story driven modeling approach that deals with how to write the user story and (ii) An automated approach for test generation. Our approach shall encompass training steps:

A. User Story Modeling Approach

In this section, we will go over the important features of user story modeling approach for the automated test generation. The approach consists of a user story template, and a set of restriction rules that can be used to improve the understanding and reduce the ambiguity in the use of natural language. Next we have defined a keyword list of data input types that are commonly used to input data in many web based applications. The attributes of these data input types give the user the flexibility to pick and choose the test data. We have also defined action type that would translate the actions on keywords, and invariably help in the process of automation. We have also illustrated how a user story could be written using our user story template, restriction rules, keywords, and action types.

This section discusses various techniques used in the literature for generation of automatic test cases from requirements.

1) Template for Writing User Stories

The proposed template for writing the user stories allows the writer to express the user story in restricted natural language. The template includes basic information about the user story, e.g. action type, data input types, and attributes of data input types that can be used for test generation. The idea is to keep the user story template simple enough to be understandable by both the users and the domain experts. The fields in the user story template are pretty simple to comprehend and are part of many templates. The Conditions of Satisfaction (CoS) do not contain any branching or conditional flows. We recommend separate user story for each branching flow.

The CoS should be written using our pre-defined data input types and action keywords, which shall help in the automation of the user story. Moreover, the user stories have to be written keeping in view the set of grammar rules that should not be

violated.

TABLE I
USER STORY TEMPLATE

User Story Name	User Story name, and it generally starts with a verb.	
Pre-conditions	What should hold true before the user story is executed.	
CoS	Also, called "Happy Path" and specifies the steps that constitute the main successful path.	
	Steps (1….n)	List of events in the main successful path.
	Post-condition	What should hold true after the CoS terminate

2) Writing Testable User Stories

In order to better understand how a user story should be written, using our user story template, let us consider an example: Consider a user who intends to make a new account on Moodle which is an open source course management system. The user would have to follow a few steps to make the new account. Ideally, he/she would have to go to the sign-up page, fill in the required information, and hit 'create my new account' button. Now, this user story when written as per our prescribed format is depicted in Table II.

TABLE II
USER STORY OF MOODLE WRITTEN AS PER USER STORY TEMPLATE

Story Name	Moodle Signup
Description	To test the signup functionality of Moodle
Actors	User
Pre conditions	User has a working internet connection.
Conditions of Satisfaction	
1.	The user *goes to* the **PAGE**:/r.php
2.	The user *enters* user name in the **TEXTFIELD_T** : id_username, Admin
3.	The user *enters* password in the **TEXTFIELD_T**: id_password, Admin@22
4.	The user *enters* email in the **TEXTFIELD_email**: id_email, administrator@hotmail.com
5.	The user *enters* email in the **TEXTFIELD_email**: id_email2, administrator@hotmail.com
6.	The user *enters* first name in the **TEXTFIELD_T**: id_firstname, Amy
7.	The user *enters* last name in the **TEXTFIELD_T**: id_lastname, Smith
8.	The user *enters* city in the **TEXTFIELD_T**: id_city, ISB
9.	The user *selects* country in the **DROPDOWN_N:** id_country,Pakistan
10.	The user *clicks* the **BUTTON**: id_submitbutton
11.	The user *sees* the **TEXT**: Welcome Admin
	end_story

The user story steps have to be written keeping in mind three things (i) no restriction rules are violated; (ii) keyword list; (iii) action type. Note that the user story contains number of keywords specified to the approach that we will discuss.

3) Restriction Rules

We adopt the natural language restriction rules proposed by Yue et al. for writing user stories [21]. The rules have been well defined to remove the ambiguities of the natural language and invariably help in improving the structure of the sentences in user stories and bring clarity to sentences. The rules have been divided into not-allowed keyword (Nak) rules and sentence structure rules. The user story steps are written keeping in line the restriction rules. Note that due to space issues we enlisted only few rules. In order to understand how these Nak rules are used in user story, let us consider the first Nak, which defines that IF/AND/BUT are not allowed in user story steps. Now, in order to comprehend this, consider our user story steps in example expressed in Table II. The Step 1 and Step 2 of the user story cannot be joined in a single step, using "IF", "AND", and "BUT". If this rule is violated and two steps are merged into one, then it would be difficult to parse the user story steps. The intention is to bring clarity to the use of natural language, and at the same time get a clear and precise meaning of the user action.

TABLE III
LIST OF NOT ALLOWED KEYWORDS (NAK 1-NAK 11)

#	Not Allowed Keywords(Nak)	Explanation
Nak 1	No IF/AND/BUT allowed in user stories	Compound sentences are difficult to parse
Nak 2	No use of UNLESS/ EXCEPT	
Nak 3	Do not use model verbs. For instance, MIGHT/POSSIBLY/COULD	Model verbs as well as adverbs introduce uncertainty.
Nak 4	Adverbs should not be used. (e.g. very)	
Nak 5	Negative adverbs are not allowed for instance NEVER and HARDLY. However, it is permissible to use no or not.	
Nak 6	The sentence should be simple, should not contain any conjunctions. One action per sentence is allowed.	It is hard to predict the correct sequence of more than one action in a sentence.

TABLE IV
LIST OF SENTENCE STRUCTURE RULES (R1-R6)

	Description	Explanation
R1.	Action steps should be sequential.	Implements use of our restricted user story template and the interaction types.
R2.	Only present tense is allowed.	Describes what the system does rather than what the system has done or will do.
R3.	Make use of active voice rather than passive voice	
R4.	The subject of the sentence in CoS should be either a system or an actor.	Explicitly defining sender and receiver would remove ambiguity in NL.
R5.	No Actor to Actor interactions are allowed	
R6.	The interactions between the actors and the systems should be clearly defined, without omitting the sender and receiver	

The sentence structure rules help to improve the structure of the sentences, and bring clarity to natural language. Table IV lists all sentence structure rules from R1 to R6.

The sentence structure rules are pretty easy to comprehend; for instance, rule 1 of sentence structure rules states that action steps should be sequential. Refer to Table VIII, each user story step contains only one action keyword and all the user story steps are written in a sequential order.

4) Keywords List of Data Input Types and their Definitions

We have proposed a list of keywords of commonly used data input types that are used in web-based applications to input certain type of data. Then, list can be extended further; however, we have defined a total of 14 keywords that define the most commonly used data input types and are outlined in Table V.

TABLE V
KEYWORD LIST AND THEIR DESCRIPTIONS

Sr.	Data Input Types	Description
1.	TextField_T	Use for combinations of text and numbers. 0 character minimum 255 characters maximum.
2.	TextField_M	Memo: It is used for bigger amounts of text. 0 character minimum 65,536 characters maximum.
3.	Page / URL/ Hyperlink	Contains links to other web pages or files.
4.	email	["[a-zA-Z0-9\\.]+ @ [a-zA-Z0-9\\-_\\.]+\\ . [a-zA-Z0-9]{3}“]‘ ’
5.	date	Format DD:MM:YY
6.	time	Format: HH:MM:SS
7.	Year	2-digit or 4-digit format allowed. Allowed values in 4-digit format: 1800 - 2100. Allowed values in 2-digit format: 60 - 69, representing years from 1960 – 2069
8.	Image	Maximum 500MB. Format JPEG, TIFF, GIF, BMP, PNG
9.	currency	15 digits of whole numbers, and 4 decimal places. Define the country currency.
10.	phone number	Contains a combination of numeric digits (0-9)
11.	Postal code	5-digit number from (0-9)
12.	Boolean	Either TRUE or FALSE
13.	Id number	13-digit number (0-9)
14.	Calendar	A combination of numeric digits (0-9) and characters (A-Z) 9-character long

The 14 data input types as defined in Table IV have been elaborated further. For instance, the first data input type TEXTFIELD_T is explained in Table VI. Due to space limitation, rest of the date input types is not appended here.

The data input type TEXTFIELD_T is used to input small amounts of texts in a textfield; for instance, when a user intends to write a “user name” in some website, he/she may choose TEXTFIELD_T as the data input type. Next he/she can choose from the attributes of the data input type TEXTFIELD_T. For instance, the user can then choose that the maximum no. of characters that he/she allows is 8, and only upper case letters from A-Z are allowed and numerical values from 1-10 are allowed. So, the user name will contain values within the defined attributes of the data type TEXTFIELD_T.

TABLE VI
DATA INPUT TYPE TEXTFIELD_T

Data Type: TEXTFIELD (TEXT)		
1.	Format:	0 character minimum 255 characters maximum.
2.	Type:	**TEXTFIELD _T** (for TEXT)
3.	Max no. of characters	26
4.	Null values allowed	Yes/No
5.	Upper case(A-Z) allowed	Yes/No
6.	Lower case (a-z) allowed	Yes/No
7.	Numerical Values Allowed	Yes/No
8.	Alphanumeric_characters_allowed	a-z, 0-9, Punctuation*, Symbols*
*punctuations: [ ] ( ) { } * , : = ; ... # ‘ “ “ < << > >> ! ? _ - ‘		
*symbols: & * \ @ • ^ ¿ + – ′ ″ ‴ § ¡ ¶ ~ ÷		

With the purpose of better understanding how the keywords and their attributes are used while writing the user story, refer to step 2 in Table II. The keyword used in step 2 is TEXTFIELD_T, which indicates that the user intends to input some data in the text field, like in the said case the user wants to input the user name. Now, once the keyword TEXTFIELD_T has been defined by the user, the next step is to define the attributes of the keyword selected, for instance the attributes related to keyword TEXTFIELD_T as outlined in Table VI. The user may like to allow only upper case alphabets to be used for user name, or he/she may allow a combination of upper and lower case alphabets with numerical values. So, for each keyword used, the attributes of the keywords have to be selected.

5) Action Type

The steps in user stories are written using the predefined action types, which are enlisted in Table VII.

TABLE VII
LIST OF ACTION TYPE

	Action Type	Keywords
1.	Goes	PAGE
2.	Enters	TEXTFIELD_T TEXTFIELD_M TEXTFIELD_EMAIL TEXTFIELD_DATE TEXTFIELD_YEAR
3.	Clicks	Button Link
4.	Selects	DROPDOWN_T RADIOBUTTON_T CHECKBOX_T
5.	Sees	TEXT

B. Automated Approach for Test Generation

These action types define the intended user actions and are used in conjunction with the keywords. For instance, the action type “goes” suggests that the user wants to go to a page (URL). So, the action keyword “goes” is used with the keyword PAGE. The step in a user story then could be like “The user goes to the PAGE”. Similarly, action type “enter” shall be used along with a keyword that is used to input data; for instance, it shall be used with data input type “TEXTFIELD _T” or with data input type “date” or “email”.

The generation of test case from user story involves three steps: (i) writing and validating the user story as per the prescribed format (ii) approach for mapping from user story to test sequence (iii) approach for test data generation.

1) Proposed Strategy for Writing and Validating the User Story

Given the user story has been written as per our prescribed format. It will then be checked for any limitations. First, it will be ensured that user story is in the required format.

Each first level field name which is also called label shall be read first. It shall be ensured that user has entered data against all the labels. The data entered by user along with the label is called a step.

Each step will be sequentially read. When the label has CoS, then second level fields, i.e. CoS steps will be read. For each CoS.step, the algorithm will check three things. First, it will check if there is any violation of restriction rules, like if

some Naks have been used or Sentence Structure Rules have been violated. Next, it will check if only defined keywords from keywords list have been used. Lastly, it will check if it contains only defined action types. If the three conditions are not met, an error will be generated else the keyword name and action type will be saved in CoSStep Result.

2) Approach for Mapping User Story to Test Sequence Illustrated.

To convert the user story into a test case, three things have to be recorded from the user story (i) Test Sequence (ii) Test data (iii) Action type.

Both test sequence and test data shall be recorded from the given example in a manner identified at Table VIII.

TABLE VIII
EXTRACTION OF TEST SEQUENCE AND TEST DATA FROM USER STORY

Sr.	Test Sequence		Test Data
	Data Input Types	Target	
1.	PAGE	signup.php	
2.	TEXTFIELD_T	id_username	Admin
3.	TEXTFIELD_T	id_password	Admin@22
4.	TEXTFIELD_email	id_email	administrator@hotmail.com
5.	TEXTFIELD_email	id_email2	administrator@hotmail.com
6.	TEXTFIELD_T	id_firstname	Amy
7.	TEXTFIELD_T	id_lastname	Smith
8.	TEXTFIELD_T	id_city	ISB
9.	DROPDOWN_N	id_country	Pakistan
10.	BUTTON	id_submitbutton	
11.	TEXT		Module Testing Page for Thesis

The "test sequence" and "test data" extracted from the given user story will be mapped to the Selenium commands. The mapping is explained in Table X. Note that the approach is specific to Selenium.

TABLE IX
MAPPING TO SELENIUM COMMANDS

	Test Sequence			Test Data
	Data Input Types	Selenium Command	Target	Value
1.	**PAGE**	Open	/login/index.php	
2.	**Textfield_T**	Type	id= uname	Admin
3.	**Textfield_T**	Type	id=upass	Admin@22
4.	**BUTTON**	ClickandWait	id=loginbtn	
5.	**TEXT**	VerifyTextPresent		Module Testing Page for Thesis

The test data can be generated randomly by generating values among the allowed attributes of the data input type. For instance, if the user intends to input a user name in textfield_t. The user can choose that the user name is 5-character long with no numerical value and only alphabets in upper and lower case are allowed. So, the test data can be generated within the allowed parameters defined by the user.

Action types are defined to express what action has to be applied on the data input types. In order to better understand the Action type lets first extract the action type from the given example and enlist them in Table X.

TABLE X
ACTION KEYWORD MAPPING

	Action Keyword	Data Input Type	Required Parameters
1.	Goes	PAGE	URL
2.	Enters	TEXTFIELD_T TEXTFIELD_M TEXTFIELD_EMAIL TEXTFIELD_DATE TEXTFIELD_YEAR	Element ID through: Element Name Element Class Name Element Tag Name Element CSS Selector Element Partial Link Element Xpath
3.	Clicks	Button Link	Button ID Button Name Button Class Name Button Tag Name Button Link Text Button CSS Selector Button Partial Link Button Xpath
4.	Selects	DROPDOWN_T RADIOBUTTON_T CHECKBOX_T	Element ID
5.	Sees	TEXT	Text

The action keyword "goes to" will be mapped to the data input type PAGE, which expects a URL of the page. Similarly, the action keyword enters is mapped to data input type TEXTFIELD_T, TEXTFILED_M, and expects an element id, which could preferably be the id of the element, but could be done through link text, name, partial link, tag name or CSS select.

3) Approach for Mapping from User Story to Test Sequence

The algorithm shall explain the mapping from user story to test sequence. It reads the steps in user story under the label "condition of satisfaction", and extracts the action type. The action type is then mapped to Selenium commands. Also, the format of allowed data input types in each step are recorded and saved.

Algorithm of Mapping from User Story to Test Sequence

Require: *CoSStepResult*, (step number in Condition of satisfaction, Action name, Keyword, Keyword format)
Require: *AcK*, List of Action types e.g. Clicks/Types/Open
Require: *Format*, List of Action type
Ensure *CoSStepResult*, *List of pairs* <ActionName, AllowedDataType, and Allowed Data Type Format>

// Read CoSStep from CoSStepResult list
//Mapping the Action type from CoS
// Command is the name of Selenium command. Action types are mapped to Selenium commands.
1: **If** *CoSStep has a AcK* **then**
2: *Action. Name* $\leftarrow$ *AcKName*
3: *ActionKeywordMapping(Action.Name)*
4: *CoSStep* $\leftarrow$ *Action.Name U Command*
5: CoSStepResult $\leftarrow$ CoSStep
6: CoSStep.next
7: **end if**
8: **If** *CoSStep has a ADT*
9: *ADT. Name* $\leftarrow$ *ADTName*
10: *AllowedDataTypes(ADT.Name)*
11: *CoSStep* $\leftarrow$ *ADT.Name U Format*

12: *CoSStepResult* ← *CoSStep*
13: *CoSStep.next*
14: **end if**

4) Approach for Test Data Generation

An important aspect of software testing is the generation of test data, which is the process of creating dataset for checking the adequacy of web based applications. The test data could be either be actual or artificial test data.

The test data can be generated using the attributes of the allowed data input types. The user first selects the data input type for instance the user has selected textfiled_T as the allowed data input type. The user then defines the attributes of that data input type, for instance the user may choose that only 26 characters are allowed, with a lower-case letters and numerical values between 1-5. Now the data can be generated by varying combinations of data attributes that have been allowed by the user. We have generated random test data.

5) Tool for Writing User Story

A tool "Test-o-Matic" has been developed which can automate the testing of web applications using user story. It takes a user story as an input written in a specified format and converts it into an executable test case to run with the help of a Selenium. Selenium web driver helps to automate the interaction with the web application.

The tool has a built-in text editor which aids the user in writing the user story or multiple user stories to build up a test suite. It also extends the facility to open up a pre-saved user story in the editor in .txt format. To write a user story the user has to identify the identification factors of the web elements, which could easily be located using the Webpage inspector within the tool.

The user story must contain the action, data type, id to identify, web elements and value to pass. Fig. 1 elaborates a user story written in Text editor of the tool. The action type is highlighted in green, the input data types are highlighted pink and web element id's are on the right hand column under the 'Identifiers List'. The test data are on the extreme right of every line. For instance, the test data in line 2, Fig. 1 is 'admin'. The user story can be written in the text editor or a pre-save user story in .txt format can also be opened.

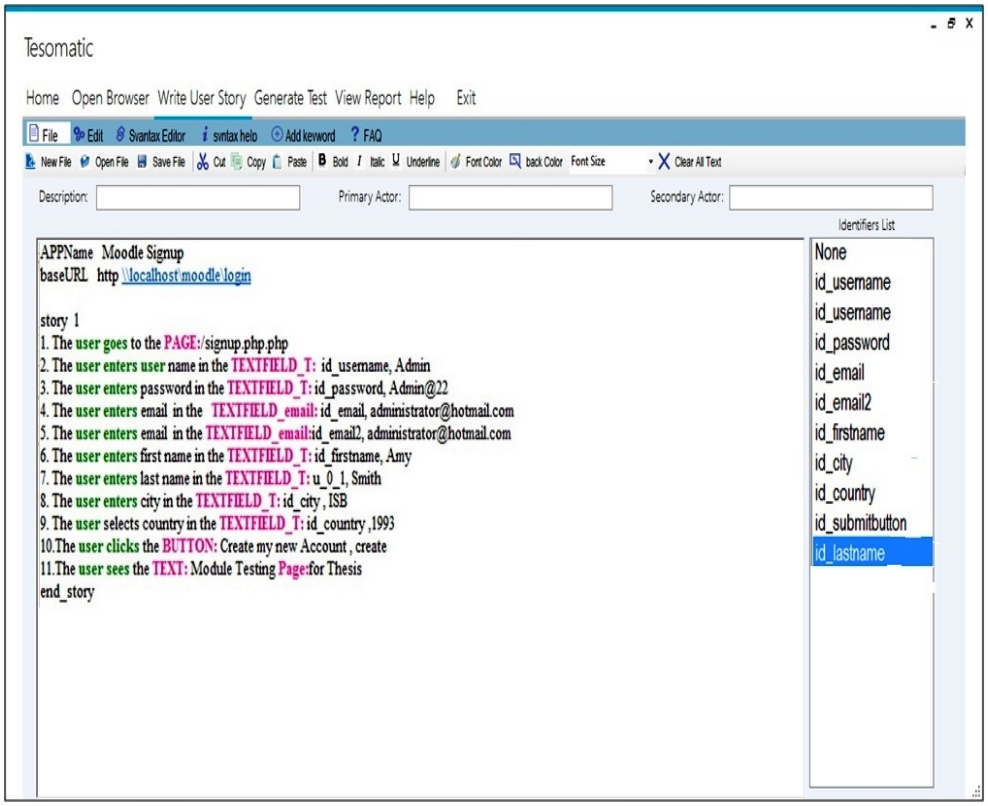

Fig. 1 Writing a user Story in Test-o-Matic tool

Once a user has written and saved the user story, the "generate test" automatically generates the test cases and redirects to a browser and runs the test case.

IV. Evaluation

We evaluate the presented approach by applying it on web based application called "Moodle", which is an open source course management system written in PHP. We selected this application as it is a widely-used system and is of sufficient

complexity. We conduct mutation analysis by seeding X faults and evaluating the fault detection effectiveness of our proposed approach.

For the experiment, we wrote a total of 12 user stories to document the requirements of Moodle. A total of 33 test cases were then written against the corresponding user stories.

For mutation analysis, we seeded mutants corresponding to mutation operators of both HTML and PHP. The HTML mutation operators are taken from Upsorn et al. [22], whereas the PHP mutation operators are taken from Mutagenesis testing framework [23]. We seeded a total of 79 mutants corresponding to HTML mutation operators and 98 mutants corresponding to the PHP mutation operators. The HTML mutation operators are outlined in Table XI, whereas the PHP mutation operators used in the analysis are highlighted in Table XII.

TABLE XI
HTML MUTATION OPERATORS

	Mutation Operator Abbreviation	Mutation Operator
1	WLR	Simple Link Replacement
2	WLD	Simple Link Deletion
3	WFR	Form Link Replacement
4	WTR	Transfer Mode Replacement
5	WHR	Hidden Form Field Replacement
6	WHD	Hidden Form Field Deletion
7	WIR	Server-Side Include Replacement
8	WID	Server-Side Include Deletion

TABLE XII
PHP MUTATION OPERATORS

	Mutation Operator Abbreviation	Mutation Operator
1	Add	Addition
2	CNE	Conditional Negation Equal
3	Equal	Equal
4	False	False Value
5	GT	Greater Than
6	Identical	Identical
7	LT	Less Than
8	LLA	Logical Lower And
9	LLO	Logical Lower Or
10	LN	Logical Not
11	LO	Logical Or
12	NE	Not Equal
13	NI	Not Identical

Refer to Table XII, the mutation operator Web Link Replacement (WLR), replaces destination of a simple link transition specified in <a> tag with another destination, Simple Link Deletion (WLD), removes the destination link specified in <a> tag, Form Link Replacement (WFR), changes destination of a form link transition to another destination in the same domain of the targeted web applications, Transfer Mode Replacement (WTR), replaces all POST requests to GET and vice versa, Hidden Form Field Replacement (WHR), alters the value attributes of <input> tag to a space, null, zero or an empty string, Hidden Form Field deletion (WHD), removes an entire block of tag < input> of type hidden, Server-Side include replacement (WIR), changes file attributes of include directives to another destination in the same domain of the targeted web applications, server-side include deletion(WID), removes an entire include directive from the html file.

The results of application of our proposed approach are shown in Table XIII for HTML mutants and in Table XIV for PHP mutants. For HTML mutants, our approach was able to kill all the seeded mutants. For PHP mutants, the test cases generated corresponding to our approach was able to kill 96 out of 97 mutants, whereas the remaining one live mutant had no effect on the behavior of the application and was therefore classified as an equivalent mutant.

TABLE XIII
SUMMARY OF MUTANTS AND RESULTS

Html File	Mutants (HTML)								Total	Killed	Score (%)
	WLR	WLD	WFR	WTR	WHR	WHD	WIR	WID			
Index form	2	5	3	4	6	2	4	2	28	26	92
index	-	1	-	-	-	-	-	2	3	3	100
Forgot password form	1	-	-	-	-	-	1	1	3	3	100
Forgot password	7	-	-	-	-	-	5	2	14	14	100
logout	3	-	-	-	-	-	3		6	4	66
signup	-	-	-	-	-	-	2		2	2	100
Signup form	2	1	-	-	-	-	3		6	6	100
Change Password	10	1	-	-	-	-	4	3	18	16	89
Change Password form	1	-	-	-	-	-	4	1	6	3	50
Set password form	-	-	-	-	-	-	2	-	2	2	100
	-	-	-	-	-	-	-	-	86	79	92

The mutation operator Addition, replaces '+' with '-', Conditional Negation-Equal(CNE), replaces '==' to '!=', Equal, replaces, FalseValue, replaces FALSE with TRUE, Greater Than, replaces '>' with '>=', Identical, replaces '===' with '!==', LessThan, replaces '<' with '>=', LogicalLowerAnd, replaces 'and' with 'or', LogicalLowerOr, replaces 'or' with 'and', LogicalNot, replaces '!' with a blank string LogicalOr, replaces '||' with'&&', NotEqual, replaces '!= or <>' with '==' and NotIdentical, replaces '!==' with '==='. A total of 97 mutants were hand seeded into the application, out of which 96 were killed, and 1 was live mutant, and 1 was equivalent mutatnt. The live mutants were not killed because it was dealing with user session data, which was not dealt in our test cases. Same could be dealt in future for better results. One equivalent mutant was ignored, yielding to a total of 97 mutants. Table XIV outlines the results.

V. CONCLUSION

Web Applications form a large portion of the overall software industry. These applications range from simple

discussion based application to highly critical financial applications. An essential element for precise functioning of these websites require testing, which should be thorough as well and systematic.

TABLE XIV
SUMMARY OF MUTANTS AND RESULTS

	Mutants (php)													Total	Killed	Score (%)
	Add	CNE	Equal	False	GT	Identical	LT	LLA	LLO	LN	LO	NE	NI			
Index form	-	5	2	-	-	-	-	-	-	7	1	-	-	15	14	93
index	1	4	2	1	1	2	2	3	1	17	-	-	6	40	40	100
Forgot pwd form	-	-	-	-	1	-	-	1	-	4	-	-	-	6	6	100
Forgot Pwd	-	-	-	-	-	-	-	3	1	2	-	-	-	6	6	100
logout	-	-	-	-	-	-	-	-	-	2	-	-	-	2	2	100
signup	-	-	-	-	-	-	-	1	-	3	-	-	-	4	4	100
Signup form	-	-	-	-	-	-	-	-	-	1	-	-	3	4	4	100
Change Pwd	-	-	-	-	1	1	-	-	-	5	-	-	-	7	7	100
Change Pwd form	-	1	1	-	-	-	-	-	-	5	-	-	1	8	7	87
Set Pwd form	-	-	-	-	1	-	-	-	-	3	-	-	1	5	5	100
Total	-	11	5	1	4	3	2	8	2	49	1	1	4	97	96	98

A common approach for testing such applications is by either writing or use a record and replay test tool which could run test cases corresponding to the user stories. Manual writing of such test cases is not scalable, especially because of the frequent changes in applications. We have presented an approach that could test web based applications automatically from restricted user stories. The restricted user stories can be written using the restricted natural language, following some set rules. The rules remove the ambiguity in natural language and improve sentence structure. A well-defined restricted user story template has been defined which allows the tester to write the restricted user story. The process is simplified using our Restricted User Story (RUST) tool 'Test-o-Matic" which enables a tester to write the user stories in restricted user story format using the restricted grammar. It also automatically checks if the restriction rules have been followed or not. The tool then automatically generates the test cases by processing the developed restricted user stories. The tool is integrated with a well-known open source record-and-replay tool, Selenium. The fault detection effectiveness of the presented approach is evaluated on an open source course management system (Moodle) using mutation analysis. The approach was tested on open source web-based application by seeding faults into the source code and the results prove that test case generation from restricted user stories can be an effective technique for automated web based application testing.

REFERENCES

[1] Offutt, J., Quality attributes of web software applications. IEEE software, 2002. 19(2): p. 25.
[2] Hagel, J. and J.S. Brown, Your next IT strategy. Harvard business review, 2001. 79(9): p. 105-115.
[3] Kreger, H., Web services conceptual architecture (WSCA 1.0). IBM Software Group, 2001. 5: p. 6-7.
[4] Zo, H., D.L. Nazareth, and H.K. Jain. Measuring reliability of applications composed of web services. in System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on. 2007. IEEE.
[5] Ricca, F. and P. Tonella. Analysis and testing of web applications. in Proceedings of the 23rd international conference on Software engineering. 2001. IEEE Computer Society.
[6] Wang, C., et al. Automatic generation of system test cases from use case specifications. in Proceedings of the 2015 International Symposium on Software Testing and Analysis. 2015. ACM.
[7] Zhang, M., et al. A systematic approach to automatically derive test cases from use cases specified in restricted natural languages. in International Conference on System Analysis and Modeling. 2014. Springer.
[8] Sinha, A., S.M. Sutton Jr, and A. Paradkar. Text2Test: Automated inspection of natural language use cases. in 2010 Third International Conference on Software Testing, Verification and Validation. 2010. IEEE.
[9] Sarmiento, E., J.C.S. do Prado Leite, and E. Almentero. C&L: Generating model based test cases from natural language requirements descriptions. in Requirements Engineering and Testing (RET), 2014 IEEE 1st International Workshop on. 2014. IEEE.
[10] Escalona, M., et al., An overview on test generation from functional requirements. Journal of Systems and Software, 2011. 84(8): p. 1379-1393.
[11] Hametner, R., D. Winkler, and A. Zoitl. Agile testing concepts based on keyword-driven testing for industrial automation systems. in IECON 2012-38th Annual Conference on IEEE Industrial Electronics Society. 2012. IEEE.
[12] Carvalho, G., et al. Test case generation from natural language requirements based on SCR specifications. in Proceedings of the 28th Annual ACM Symposium on Applied Computing. 2013. ACM.
[13] Lucassen, G., et al. Forging high-quality user stories: towards a discipline for agile requirements. in 2015 IEEE 23rd international requirements engineering conference (RE). 2015. IEEE.
[14] North, D., Introducing bdd. Better Software, March, 2006.
[15] North, D., What's in a story. 2007, February.
[16] Saravana. K.M, G.N.B., Rajkumar, Dr. A. Kovalan, " Case Study On Agile User Stories Prioritization Using Imaginative Standard". IJERA Vol. 2(no. 5): p. pp. 472-480.
[17] Cucumber, https://cucumber.io/. Retrieved June, 2015.
[18] JBehave, http://jbehave.org/. Retrieved July, 2015.
[19] Testing, C., http://www.jackkinsella.ie/2011/09/26/why-bother-with-cucumber-testing.html. Access Date, March 2016.
[20] Drawbacks, C., https://www.jimmycuadra.com/posts/please-don-t-use-cucumber/. Access Date, March 2016.
[21] Yue, T., L.C. Briand, and Y. Labiche. A use case modeling approach to facilitate the transition towards analysis models: Concepts and empirical evaluation. in International Conference on Model Driven Engineering Languages and Systems. 2009. Springer.
[22] Praphamontripong, U. and J. Offutt. Applying Mutation Testing to Web Applications. in ICST Workshops. 2010.
[23] Framework, P.M.t., A PHP 5.3+ Mutation Testing framework. https://github.com/padraic/mutagenesis, January 2016.