A Software Tool to Convert Requirements to Test Cases

1st Abhimanyu Gupta
Operations & ITM Department
Chaifetz School of Business, Saint Louis University, Saint Louis, USA
abhimanyu.gupta@slu.edu

2nd Palash Bera
Operations & ITM Department
Chaifetz School of Business, Saint Louis University, Saint Louis, USA
palash.bera@slu.edu

***Abstract*—As software test design is a manual process, test cases are generally prepared manually. We use a structured method to extract business requirements and feed these requirements in a tool (TestAlgo). The tool translates these requirements and automatically creates test cases and business requirements models such as process models. We suggest that this tool can not only save time and effort in creating automated test cases but also handle changes in requirements efficiently.**

***Keywords*—requirements, test cases, requirements traceability matrix**

I. Introduction

Developing test cases is considered to be a complex art and the process is subjective and is based on testers' domain knowledge [1]. Structured business requirements and test cases are the two sides of the same coin. If requirements are captured systematically in a structured method, then it is possible to convert these requirements to test cases. In this paper, we discuss about our tool – TestAlgo (www.testingalgorithms.com) that uses a structured methodology to capture requirements and then converts them to business requirements (e.g. process models, UML Use Case, User Stories) and testing outputs (e.g. manual test cases, Requirements Traceability Matrix). A key advantage of this tool is if requirements changes, then the structured requirements in the tool is updated and the outputs such as test cases are regenerated.

II. Methodology used in the tool

A. *Action Triad Method*

To develop automated test cases, we propose a model-based method that can convert business requirements to conceptual models. Conceptual models are used for documenting the features of the domain that needs to be reflected in the Information Systems [2]. We use a modified version of Entity Relationship (ER) modeling as a conceptual modeling technique. ER models [3] describe the domain concepts using entities and relationships and the technique is a popular conceptual modeling technique in practice [4]. The traditional ER models can be used for modeling domain concepts of an organization but ER models cannot be used directly for modeling software applications. Therefore, we adopted certain changes to the ER modeling technique and termed it *Action Triad Method*.

Action is the focus of this method and is modeled as a relationship between two concepts. As actions are performed by specific agents on other agents or objects thus this model is described as a set of action triads consisting of Agent-Action-Concept. Accordingly, we define an action triad as <x, y, z> where x is an agent, y is an action performed by the agent, and z is the concept (agent or object) on which the action is performed on. The concepts of action triad are described in Table 1.

TABLE I. Core concepts of action triad

Concepts in Action Triad	Definition
Agent	An agent is an entity that can interact with objects or other agents [5].
Function	A function represents activities that are performed by agents.
Object	An object represents non-agents in the domain with which the agents act. Objects can be tangible (e.g. Phone) or intangible (e.g. Web site).
Dimension	Dimensions describe the objects or agents in measurable form.
Instance	Dimensions have instances that are generally expressed in text or numbers.

A key feature of this method is to instantiate each dimension with additional concepts that are relevant to software testing- instances, scenario, expected results, and requirement ID. A dimension can have multiple values as instances. Each instance can have a positive or a negative scenario. Positive scenario means that the action with the specific instance can be performed successfully. If the user cannot perform the action successfully with a specific instance, then the scenario is negative. For example, if password is null then the null instance is considered as a negative scenario as the action login (dimension of which is password) cannot be completed successfully. Expected results indicate the outcome when an action is taken using a specific instance (e.g. null password should result in incomplete login). Requirement IDs correspond to the details of the instances mentioned in the functional requirements document.

III. TESTALGO TOOL

A. Approach

The philosophy that we adopt for developing Action Triad Method is software testing can be considered as a game where a tester "*pokes*" an application under test using various input actions and data combinations and then the tester compares the observed behavior of the application with its expected behavior. Thus, the application should be modeled in such a way that it can be poked based on different combination of values in test cases. These test cases can be created using the dimension and instance concepts of the method. However, if all possible combinations of values are considered then large number of test cases will be created. Therefore, an optimization engine should be used to come up with minimum number of test cases that can cover maximum combination of values. This approach is shown in Figure 1.

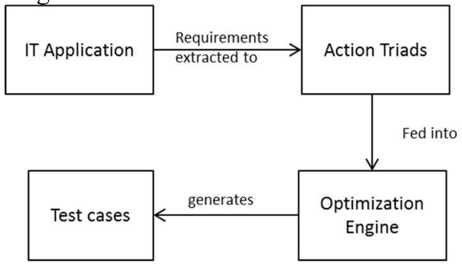


Fig. 1. Approach to generate automated test cases

B. A Case Study

To demonstrate the application of our tool to generate test cases automatically, we use a sample case study. In this case study, a user logs in and logs out of an application. The description of the requirements is provided in Table 2. The objective of this case study is to create automated test cases to test the functionalities as described in Table 2.

TABLE II. FUNCTIONAL REQUIREMENTS OF THE CASE

Description	Requirement ID
The user needs to provide a valid username and password to successfully login to the application.	1.1
After the user logs into the application, then she is taken to the browse application page where the user can click on different reports.	1.2
On clicking the logout button the user gets an alert to logout. If the user clicks on yes then she is logged out otherwise on clicking cancel the user is taken back to the browse application page.	1.3
A user has a valid username (John) and a valid password (1234!).	1.4

The screen shots of the interface of the application are shown in Figure 2.

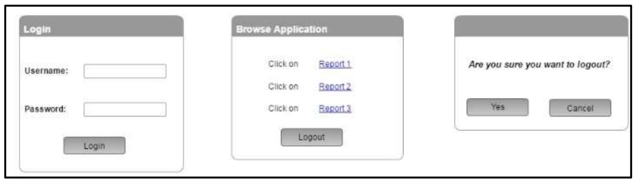
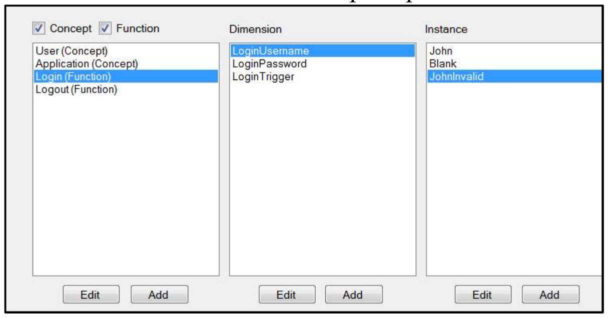


Fig. 2. Login and Logout functions of an application

To apply the action triad method, the application description is decomposed into two action triads: <User, Login, Application> and <User, Logout, Application>. The dimensions of the functions and the agents are identified using the action triad method.

C. TestAlgo tool

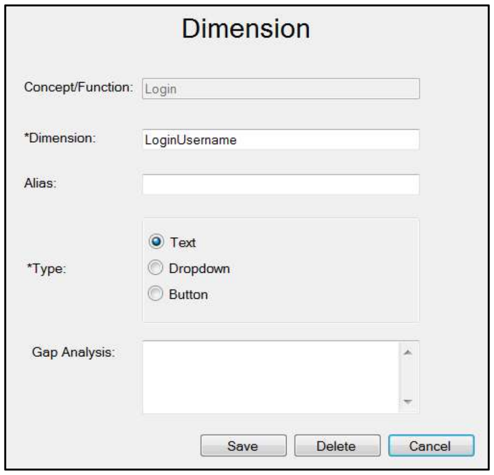
A tool for the Action Triad method has been developed that facilitates the input of the triad concepts. The triads and its details can be entered in the tool. A screen shot of the tool is shown in Figure 3. The tool checks for the implementation of the method. For example, if the user forgets to create an instance of a dimension then it will prompt an error.



☑ Concept ☑ Function

Concept / Function	Dimension	Instance
User (Concept)	LoginUsername	John
Application (Concept)	LoginPassword	Blank
Login (Function)	LoginTrigger	JohnInvalid
Logout (Function)		

Fig. 3. TestAlgo tool to input the concepts of Action Triad



Dimension

Concept/Function: Login

*Dimension: LoginUsername

Alias:

*Type: [✓ Text] Dropdown Button

Gap Analysis:

Fig. 4. Screen to enter dimension values in TestAlgo

Once the triad information is obtained, the tool uses a statistical pairwise optimization engine to create test cases with specific steps. The engine optimizes the dimensions and their instances to create test cases. The dimensions become test case step descriptions and the instances become the values of these descriptions. As all the instances will not fit into one test case therefore the optimization engine will create multiple test cases ensuring that a pair of instances is covered in at least one test case. To make the test case step description readable, specific user actions (e.g. enter or click) are used in the step descriptions. In the tool, each dimension can be classified as: text, dropdown, and button (Fig. 4). These keywords are linked to action words that users perform to test the application. For example, if text is selected as a dimension type then the keyword "Enter" will be used in the test case step description. Thus, a test case step description could be "Enter John as LoginUsername" where John is an instance of the dimension LoginUsername whose type is text. Similarly, when dimension type is button then the keyword "click" is used in the step description.

IV. BUSINESS REQUIREMENTS OUTPUTS OF THE TOOL

Two popular business models such as BPMN and UML use cases are automatically generated from the tool. Use case describes the functionalities performed by the agents and BPMN shows the sequence or flow of the functionalities performed by the agent. A key advantage of all these business models is they are created automatically.

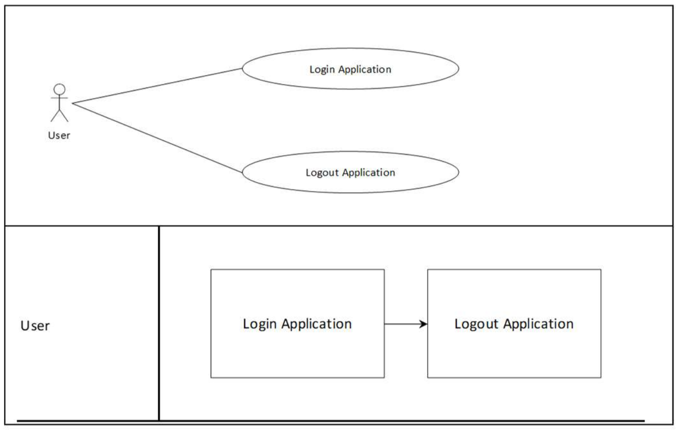


Fig. 5. Business models generated by TestAlgo

```
Feature: User Login Application

 As a User
 I want to Login Application

Scenario Outline: User attempts to Login Application with various input parameters

Given UserPassword is '<UserPassword>'
 And UserUsername is '<UserUsername>'

 When User Login Application
 And LoginUsername is '<LoginUsername>'
 And LoginPassword is '<LoginPassword>'
 And LoginTrigger is '<LoginTrigger>'

 Then status of Login should be '<Status>'

 Examples:

 | UserPassword | UserUsername | LoginUsername | LoginPassword | LoginTrigger | Status |
 | 1234! | John | John | 1234! | Login | Successful |
 | 1234! | John | John | Password | Login | Unsuccessful |
 | 1234! | John | Blank | 1234! | Login | Unsuccessful |
 | 1234! | John | JohnInvalid | 1234! | Login | Unsuccessful |
 | 1234! | John | John | Blank | Login | Unsuccessful |
```

Fig. 6. User story generated by TestAlgo

In Agile methodology, an important task is to develop the "user stories" that describe software features from end-user perspectives. Two main advantages of the user stories developed by our algorithms are: (1) standardized format of the stories and (2) elimination of human resources for creation of these stories. The acceptance criteria can also be represented in test cases which are also generated in optimized numbers.

Another important model generated by the tool is the flow chart model. This model shows the graphical flow of the application. This model can be very useful to visually validate the logics of the application and design/modify the application interface.

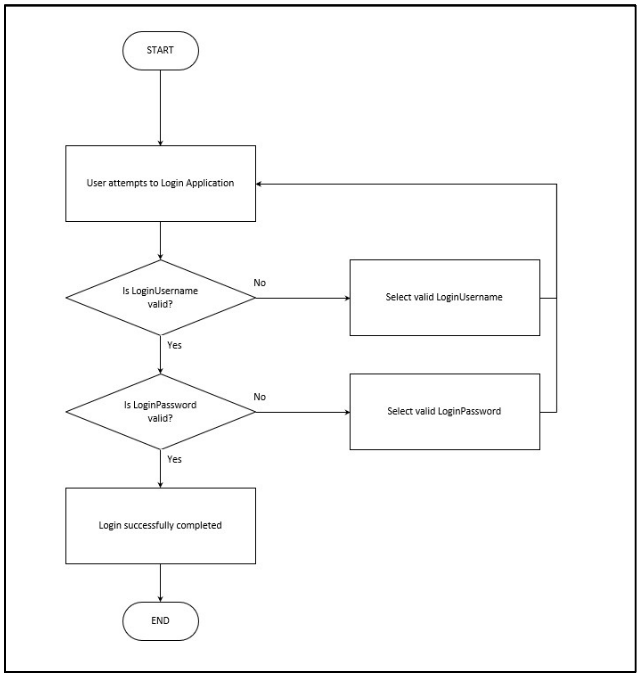

START
User attempts to Login Application
Is LoginUsername valid?
No
Select valid LoginUsername
Yes
Is LoginPassword valid?
No
Select valid LoginPassword
Yes
Login successfully completed
END

Fig. 7. Flow chart model developed by TestAlgo

V. TESTING OUTPUTS OF THE TOOL

For the case study described here, the tool generated 7 test cases (2 positive and 5 negative) for the two triads. Table 4 shows two test cases that are generated from the tool. Each test case has a description, step number, step description, expected results, and traceability. The step description has specific actions with specific values (e.g. Enter "John" as LoginUserName). However, each test case is different as different combinations of instances are used. If one of the instances has a negative scenario then the test case is considered negative meaning the actions mentioned in the test case should not be successfully executed. If no instances have negative scenarios then the test case is considered as positive meaning the actions mentioned in the test case should be successfully executed.

Each test case starts with the instances of the dimensions (e.g. UserPassword and UserUsername) of the entities (e.g. user). This step is considered as a pre-requisite i.e. the condition that is required before the test case can be run. A pre-requisite step does not have expected results and traceability.

Each test case can have only one negative instance scenario. This is because from a tester's perspective, if a test case has two or more negative instances (e.g. username is null and password is null) then it is not possible to identify the exact cause of failure of the test (e.g. whether the test failed because password was incorrect or it failed because the username was incorrect).

TABLE III. PARTIAL LIST OF TEST CASES BY TESTALGO

Test Case #	Test Case Description	Step #	Step Description	Expected Results
Test Case 1	(Positive Scenario) Validate Login function with all valid values	Step 1	Identify User with UserPassword = 1234!, UserUsername = John	
		Step 2	Enter "John" as LoginUsername	Username John should be displayed
		Step 3	Enter "1234!" as LoginPassword	Encrypted password should be displayed
		Step 4	Click "Login" button	On successful login the user is taken to the browse page
Test Case 2	(Negative Scenario) Validate Login function with invalid LoginPassword	Step 1	Identify User with UserPassword = 1234!, UserUsername = John	
		Step 2	Enter "John" as LoginUsername	Username John should be displayed
		Step 3	Enter "Password" as LoginPassword	Encrypted password should be displayed
		Step 4	Click "Login" button	On successful login the user is taken to the browse page

Table 3 shows the sample test cases that are automatically generated from the TestAlgo tool. Test case 1 is a positive scenario as the test is expected to pass with the actions mentioned. Test case 2 is a negative scenario test as it is expected to fail when executed. This is because in step 3 of the test case 2, the password is not a valid password. Traceability to the requirements are also shown in these test cases (not shown here due to lack of space). These test cases are developed in ALM format and can be easily uploaded in a test management tool.

A key output that is generated from the tool is the requirements traceability matrix (RTM). This matrix traces each test case with the business requirements. All possible combination of pairs of instances are identified and mapped with the test cases. This table is a proof that at least one pair is covered in each test case. A partial RTM is shown in Table 4. For example, UserPassword 1234! And LoginPassword 1234! combination can be found in test case 1 but not in test case 2.

TABLE IV. RTM DEVELOPED BY TESTALGO

Pairwise Combination	Test Case 1	Test Case 2
UserPassword = 1234!, LoginPassword = 1234!	1	
UserPassword = 1234!, LoginTrigger = Login	1	1
UserUsername = John, LoginUsername = John	1	1
UserUsername = John, LoginPassword = 1234!	1	

In addition to the RTM, a similarity index of the test cases is provided as an output of the tool. This mapping (Table 5) shows how each test case is related to the other. For example, test case 2 is 60% similar to test case 3. This index is useful to identify the group of test cases that are similar (or dissimilar) to each other. If a bug is identified and related to a test case, this similarity index can tell whether other related test cases should also be tested.

TABLE V. SIMILARITY INDEX OF TEST CASES

Test Cases	1	2	3	4	5
1	1	0.8	0.8	0.8	0.8
2	0.8	1	0.6	0.6	0.8
3	0.8	0.6	1	0.8	0.6
4	0.8	0.6	0.8	1	0.6
5	0.8	0.8	0.6	0.6	1

VI. CONCLUSION

Test case development has been considered as a manual process because of the subjective nature of interpretation of business requirements. But when application requirements change very frequently, testers are forced to redevelop the test cases constantly increasing the testing time and execution. To address this problem, we developed a conceptual model based method (termed action triad) and implemented this methodology using a tool – TestAlgo. We feed the structured requirements in the tool and it generates multiple business requirements and testing outputs. The test cases that are generated are useful as they are automatically generated based on structured requirements that are fed in the tool. In addition to the saving of time to manually create the test cases, a major advantage of the method is it handles the change of requirements efficiently. When the changes in the requirements are updated in the tool, then a new set of test cases is regenerated.

Our action triad method and the implementation of it through our tool create opportunities for Business Analysts (BA) to serve as the role of testers. In agile based environment, our deliverables are generated at lightning speed and thus can be used for requirements documentation. The changes in the business requirements can be made in the tool and a new set of requirements and testing deliverables can be created automatically by the software tool. Finally, if BAs prepare requirements documents using our deliverables then test design is automatically taken care of without spending any additional effort and time.

REFERENCES

1. Kaner, C. *Architectures of Test Automation*. in *STAR West*. 2000. San Jose, Canlifornia.
2. Dobing, B. and J. Parsons, *Dimensions of UML Diagram Use: A Survey of Practitioners*. Journal of Database Management, 2008. **19**: p. 1-18.
3. Teory, T.J., D. Yang, and J.P. Fry, *A Logical Design Methodology for Relational Databases Using the Extended Entity-Relationship Model.* ACM Computing Surveys, 1986. **18**(2): p. 197-222.
4. Dobing, B. and J. Parsons, *How UML is used.* Commun. ACM, 2006. **49**(5): p. 109-113.
5. Wooldridge, M., *Reasoning about Rational Agents*. 2000, Massachusetts: The MIT Press.