A Proposed Method in Agile Practices to Create Requirements Documentation and Test Cases

Palash Bera[1] and Abhimanyu Gupta[2]

[1] Saint Louis University
pbera@slu.edu
[2] Saint Louis University
abhimanyugupta@slu.edu

Abstract. Two problems are common in agile based software development approaches. First, software requirements change frequently and therefore they are difficult to maintain and document. Second, test case development takes time and effort and thus the testing process is often delayed. To provide solution to these problems, we suggest using Action-Triad method for capturing the software application requirements. This method can be used to create conceptual models that can be used as documentation in agile development. The method can also create requirements traceability matrix (RTM) and test cases. When the requirements change, the conceptual models, RTM, and test cases can be regenerated. The method is demonstrated here using a case study.

Keywords: Requirements Documentation, Conceptual Models, Test Case

1 Introduction

Agile practices advocate on iterative and test-driven software development strategy where the focus is on working software and not on extensive documentation [1, 2]. Agile practices focus on capturing the requirements through user stories which are constantly updated in the development cycle [3]. In a survey on agile practice, [4] found that most organizations struggle to identify requirements and maintain the changes in the requirements. A test case is a set of inputs, execution conditions, and expected results to verify that a software program is compliant to specific requirements [5]. Agile practices suggest that test cases based on the requirements should be written prior to the codes written by the developers [6]. However, based on a case study, [1] found that incremental software development was not tested as test implementation was too slow. Developing test cases from the requirements took lot of time and effort. As a result, much testing was postponed until the later stages of the project, resulting in generating large number of software defects.

Kaner [7] mentions that designing good test cases is a complex art as the process of test creation is subjective and is based on testers' domain knowledge. Also there is no clear method in writing test cases. For example Kaner mentions that if an application

has 20 variables, then should we create one test case combining all variables or multiple test cases for each variable. In agile development, creating test cases is particularly challenging as the application changes frequently during the development process. Based on the above discussion, two questions can be raised in the context of agile practices: (1) *how to document requirements in a standardized way so that the requirement changes are captured quickly?* and *(2) how to create test cases consistently and quickly prior to software application development?* To answer these questions, this paper proposes an action based method for capturing requirements in a structured way and creating test cases based on these requirements. The structured requirements can be documented and changes in the requirements can be controlled through the use of the method. The method can be also used for creating test cases thus test cases can be created before the software application is developed. In section 2, this method is described and in section 3, a sample case study is presented. Section 4 is the discussion section where the benefits for practice are discussed.

2 Action Triad method

To develop requirements documentation and test cases, we propose a method that can convert software application requirements to a set of triads called *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 1. Action triad concepts

Concepts in Action Triad	Definition
Agent	An agent is an entity that can interact with objects or other agents [8].
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.

Software application requirements can be decomposed into a set of triads. These triads can have precedence (i.e. function of one triad needs to be performed earlier than the other). Additional concepts - instances, scenario, expected results, and requirement ID are identified that are relevant to software testing. A dimension can have multiple values or 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).

When the action triads are processed by a software tool then two types of outputs are generated- a set of conceptual models and tables and set of test cases. Conceptual models are generally graphical representations (e.g. UML Use Case and BPMN process models) of the domain that need to be reflected in the Information Systems [9]. Conceptual models are used for documenting the features of the domain that needs to be reflected in the Information Systems [9]. [10] mention that conceptual models could be highly relevant in agile development methods as agile development requires efficient communication between the stakeholders. The conceptual models and their descriptions can be used as requirements documentation in agile development. Using the precedence of the triads, a high level BPMN process model can be created and using the agents and the action performed by them, UML Use Case model can be developed. The dimensions of the functions can be used to create test case steps and the instances can be used in these test cases. As a large combination of instances can create large number of test cases therefore an optimization engine (part of the software tool) could be used to come up with minimum number of test cases that can cover maximum combination of instances. As the action triad method captures the requirements ID, therefore a requirements traceability matrix (RTM) can also be generated. RTM maps the requirements with the test cases. A set of guidelines is proposed so that application requirements can be captured in to set of action triads. These guidelines are:

- Decompose an application to be developed to sets of action triads.
- Indicate the sequence of the action triads for the application.
- Ensure that each action triad is unique for an application under testing.
- Ensure that each dimension has an instance with at least one positive scenario.

In this method, functions must have dimensions but agents and objects may not.

3 A Case Study

To illustrate the application of the Action-Triad method, a small case study is used. 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 conceptual models, RTM, and automated test cases to test the functionalities as described.

Table 2. Requirements Example

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 1.

Fig. 1: 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 two functions- login and logout are the actions that users can perform in the application. Note that the number of triads is not dependent on the number of application screens but depends on the functions that users perform. The dimensions are elaborated in Table 3 using instances, scenarios, expected results, and requirements ID. The details of this information are obtained from Table 2 and Figure 1. The number of instances that can be used for dimensions in functions will depend on the requirements. If high number of instances is used in the triads then more number of test cases may be generated. A mix of positive and negative scenarios for the instances is recommended to test the application properly.

Note that the username and password appear as dimensions in both user agent and the login function. In the former case, the instances of these dimensions have positive scenarios, meaning the instances are *actual* username and password that are assigned to the user. In the latter case, the instances are the ones that a tester will input in the application to test it. Some of these instances will have negative scenarios such as when password is Blank.

Table 3. Details of the dimensions of the Login and Logout functions

Dimension	Instance	Scenario	Expected Result	Req. ID
UserUsername	John	Positive		
UserPassword	1234!	Positive		
LoginUsername	John	Positive	Username John should be displayed	1.1, 1.4
LoginUsername	Blank	Negative	No username is displayed	1.1
LoginPassword	1234!	Positive	Encrypted password is displayed	1.1, 1.4
LoginPassword	Blank	Negative	No password is displayed	1.1
LoginPassword	Password	Negative	Encrypted password is displayed	1.1
LoginTrigger	Login	Positive	On successful login the user is taken to the browse page	1.2
LoginUsername	JohnInvalid	Negative	Username JohnInvalid should be displayed	1.1
LogoutTrigger	Logout	Positive	On clicking the logout, the user is taken to the alert	1.3
LogoutConfirmTrigger	Yes	Positive	User is logged out of the application	1.3
LogoutConfirmTrigger	Cancel	Negative	User is not logged out of the application and taken back to the browse application page	1.3

The outputs of the action-triad method are described next. In Figure 2, high level Use Case and BPMN diagram are generated from the method. As the agent –user is associated with two actions therefore the Use Case shows these two actions. Login application is followed by the logout application and they are shown accordingly in the BPMN. It is to be noted that these models are high level as detailed level information (e.g. join operators in BPMN) is not captured in the method.

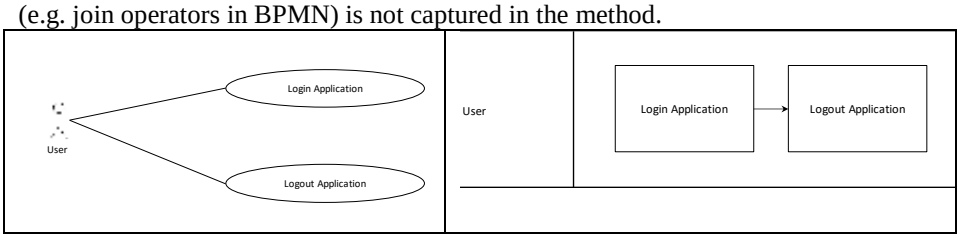

Fig. 2: Use Case and BPMN models of the application

In addition to the above conceptual models, an action path model for each function is created. Figure 3 shows the action path model for the Login function where the valid and invalid ways of login are shown. This action path model can be useful in designing or modifying the interface (e.g. login screen).

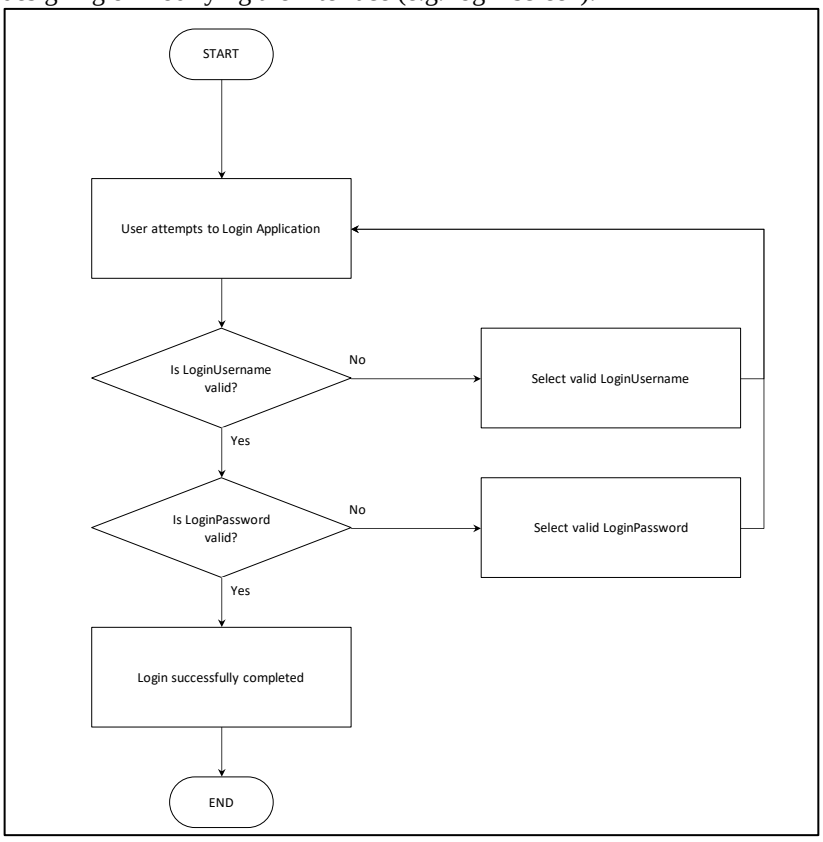

Fig. 3: Action path model for Login function

Based on the triad information, the tool uses an optimization engine to create test cases with specific steps. A test case generally contains specific steps that testers need to perform to test a certain function of an application. Along with the steps, a test case contains test objective, expected results, and traceability (i.e. which steps correspond to specific part of the requirements) [11]. The engine optimizes the dimensions and their instances to create test cases. The dimensions (column 1 of table 3) become test case step descriptions and the instances (column 2 of table 3) 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. 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 (type is text). Similarly when dimension type is button then the keyword "click" is used in the step description.

For the case study described here, the action triad method generated 5 test cases (1 positive and 4 negative) for the login function. Table 4 shows two test cases of login function that are generated using the action-triad method. 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 executed successfully.

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 4. Sample test cases created for the login function

Test Case #	Test Case Description	Step #	Step Description	Expected Results	Traceability
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	1.1, 1.4
		Step 3	Enter "1234!" as LoginPassword	Encrypted password should be displayed	1.1, 1.4
		Step 4	Click "Login" button	On successful login the user is taken to the browse page	1.2
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	1.1, 1.4
		Step 3	Enter "Password" as LoginPassword	Encrypted password should be displayed	1.1
		Step 4	Click "Login" button	On successful login the user is taken to the browse page	1.2

The action triad method also creates a standardized form of test cases called- *Gherkin syntax* (Figure 4). Gherkin syntax is a business readable language that describes the behavior of the software. The syntax includes status parameter and the values of status are either successful or unsuccessful. If a test case has a negative value of an instance then the status is unsuccessful or else the status is successful. Gherkin syntax (Figure 4) is a rearrangement of the test cases written in tabular format (Table 4). In some agile projects, user stories are written in Gherkin syntax and used to generate automated test scripts using test automation tools.

```
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. 4: Gherkin syntax of the test cases for the login function

RTM is created where the pairwise combination of each dimension is mapped with the test cases. The complete RTM shows that all the possible pairwise input values as defined in the login function have been covered by at least one test case. For example in Table 5, the partial RTM shows that the pair "UserPassword = 1234! and LoginUsername = John" (row 3) is mentioned in the test cases 1, 2, and 5. This table thus ensures that the requirements are covered completely in all these optimized 5 test cases.

Table 5. Partial Requirements Traceability Matrix (RTM) for the Login function

Pairwise Combination	Test Case 1	Test Case 2	Test Case 3	Test Case 4	Test Case 5	Total
UserPassword = 1234!, UserUsername = John	1	1	1	1	1	5
UserPassword = 1234!, LoginUsername = John	1	1			1	3
UserPassword = 1234!, LoginPassword = 1234!	1		1	1		3
UserPassword = 1234!, LoginTrigger = Login	1	1	1	1	1	5
UserUsername = John, LoginUsername = John	1	1			1	3
UserUsername = John, LoginPassword = 1234!	1		1	1		3
UserUsername = John, LoginTrigger = Login	1	1	1	1	1	5
LoginUsername = John, LoginPassword = 1234!	1					1
LoginUsername = John, LoginTrigger = Login	1	1			1	3
LoginPassword = 1234!, LoginTrigger = Login	1		1	1		3

4 Discussion

In agile practices, software development is performed by teams that include business analysts, testers, and developers [1]. Due to short implementation cycles (sprints) in agile, the analysts face difficulty in developing requirements documentation. Further as the application requirements change, the documentation needs to be updated frequently. The testers in the agile team face the challenge of developing the test cases prior to the codes written by the developers.

The proposed action-triad method helps to resolve these challenges by capturing the requirements in a systemic way. The method generates conceptual models that can be used for requirements documentation. The RTM and test cases in multiple formats are also generated in this method. When the application requirements change, the conceptual models, RTM, and test cases can be regenerated.

The requirements documentation, RTM, and test cases are currently written manually but the use of the action-triad method can help the agile teams to develop these outputs automatically. Thus the software applications can be developed in shorter time.

Currently, the action-triad method is applied to few real-world agile practice based projects using a software tool. To test the effectiveness of the method, it has to be applied to complex application development projects and modified as necessary.

State diagrams as models have also been used for creating automated test cases. A state diagram depicts the states that a system can assume and shows the events that cause and/or result from a change from one state to another [11]. Test cases are derived from the state diagrams by identifying valid and invalid state transitions [11]. However, there are two main challenges in using such models. First, developing such models is difficult due to the complexity of state diagrams. As the application gets complex, the number of states and the transitions grow rapidly creating explosion of states (a phenomenon called – *state explosion* [12]). Second, creating optimized test

cases from these models is challenging because of large number of states and the frequent change in the functionalities of the application. A complex application will have a large number of states and transitions. Thus modeling such application using state diagrams is challenging and so is deriving the test cases from the diagrams. Alternatively, we suggest using ER based Action Triad method as there is no need of modeling transitions and only the higher state changes are modeled using triads. Decomposing an application into set of triads is still a modeling skill that modelers need to apply. However, once the triads are identified, the methodology helps to create the model elements consistently.

References

1. Heeager, L., *Introducing Agile Practices in a Documentation-Driven Software Development Practice: A Cast Study*. Journal of Information Technology Case and Application Research, 2012. **14**(1): p. 3-24.
2. Fowler, M. *The new methodology*. 2003.
3. Beck, K. and C. Andres, *Extreme Programming Explained: Embrace CHange*. 2004, Boston: Addison- Wesley Professional.
4. Sillitti, A., et al. *Managing Uncertainty in Requirements: A Survey in Documentation-Driven and Agile Companies*. in *11th IEEE Int'l Symp. Software Metrics*. 2005. IEEE Press.
5. IEEE, *IEEE Standard 610*, in *IEEE Standards Collection: Software Engineering*. 1990.
6. Boehm, B. and B. Turner, *Management Challenges to Implementing Agile Processes in Traditional Development Organizations.* IEEE Software, 2005. **22**(5): p. 30-39.
7. Kaner, C. *Architectures of Test Automation*. in *STAR West*. 2000. San Jose, Canlifornia.
8. Wooldridge, M., *Reasoning about Rational Agents*. 2000, Massachusetts: The MIT Press.
9. Dobing, B. and J. Parsons, *Dimensions of UML Diagram Use: A Survey of Practitioners.* Journal of Database Management, 2008. **19**: p. 1-18.
10. Rubin, E. and H. Rubin, *Supporting Agile Software Development Through Active Documentation.* Requirements Engineering, 2011. **16**: p. 117-132.
11. Board, I.S.T.Q. *Standard Glossary of Terms Used in Software Testing* 2015. **Version 3.1**.
12. Valmari, A. *The State Explosion Problem, Lectures on Petri nets: Advances in Petri nets*. 1998. Berlin-Heidelberg: Springer-Verlang.