Poster: A Systematic Literature Review to Support the Selection of User Acceptance Testing Techniques

Ernani César Dos Santos
Federal University of Santa Catarina
Florianópolis, Santa Catarina, Brazil
ernani.santos@posgrad.ufsc.br

Patrícia Vilain
Federal University of Santa Catarina
Florianópolis, Santa Catarina, Brazil
patricia.vilain@ufsc.br

Douglas Hiura Longo
Federal University of Santa Catarina
Florianópolis, Santa Catarina, Brazil
douglashiura@gmail.com

ABSTRACT

User Acceptance Testing (UAT) aims to determine whether or not a software satisfies users acceptance criteria. Although some studies have used acceptance tests as software requirements, no previous study has collected information about available UAT techniques and established a comparison of them, to support an organization in the selection of one over another. This work presents a Systematic Literature Review on UAT to find out available techniques and compare their main features. We selected 80 studies and found out 21 UAT techniques. As result, we created a comparative table summarizing these techniques and their features.

CCS CONCEPTS

• **Software and its engineering → Software creation and management → Software verification and validation → Process validation → Acceptance testing**

KEYWORDS

User acceptance testing, techniques, classification, features.

1 INTRODUCTION

Some agile methodologists consider that acceptance tests are more precise and accurate sources of information about the customer's needs than descriptions in natural language [1]. Then, they have proposed the use of User Acceptance Testing (UAT) techniques to specify software requirements (SR).

A UAT technique is a form of carrying out acceptance testing, which includes procedures, tools and a notation. The notation is a set of components, such as diagrams or tables, used to represent acceptance tests. As each UAT technique has a specific notation and method, so one could be more adherent to an organization or project rather than another.

ICSE '18 Companion, May 27-June 3, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5663-3/18/05.
https://doi.org/10.1145/3183440.3195036

Several studies have addressed the use of UAT as software requirement artifacts (SRA) [1-3]. Therefore, they focus on one UAT technique and do not establish a comparison between different ones. We address this problem performing a Systematic Literature Review (SLR), which goal is to find out UAT techniques and comparing their features. This study is a secondary literature review and it has been undertaken based on the guidelines proposed by Kitchenham in [4].

The research questions addressed in this study are: (RQ1) Which are the techniques used to specify acceptance test scenarios? (RQ2) Which are the notations used by the UAT techniques as means for specifying SR?

The purpose of the RQ1 is enumerating the UAT techniques based on their notation. Whereas, the RQ2 points out the UAT techniques which the selected studies have used to specify SR instead of traditional artifacts such as user stories or use cases.

2 SEARCH PROCESS

The search key was defined considering the PICOC (population, intervention, comparison, outcomes, and context) criteria as suggested in [4], resulting in: "(acceptance test OR user test OR customer test OR functional test OR user scenarios OR user stories) AND (software validation OR software testing OR requirements engineering) AND (technique OR tool OR notation OR format OR requirements)"

We have performed our searching in the titles, keywords, and abstracts of the studies on ACM Digital Library, IEEEXplore, Scopus, ScienceDirect, and Springer. The searching was limited to studies that belong to the Computer Science area and that are written in English.

The search process was conducted as follows. Firstly, we performed the search key on chosen databases and exported the results to a spreadsheet. Then, we excluded all duplicated entries and obtained 1418 studies as a partial result. Next, we excluded several studies analyzing their titles and abstracts. Finally, we read all remaining studies and selected 80 studies.

3 RESULTS AND DISCUSSION

Addressing RQ1, we have found out 21 different UAT techniques that are identified from T1 to T21 in Table 1. Fit tables (T2) is the most highlighted technique, it was referenced by 18 studies, which is in agreement with [5] that remarks Fit tables as the most popular UAT technique in the industry context.

As show in the fifth column of Table 1, we classified the UAT techniques according to their notation, which is composed by 7

classes: Table-driven Test, Record/Replay, Natural Languages, Formal Languages, Formal Diagrams, Behavior-driven, and Programming Language [6, 7].

The column SPEC (see Table 1) addresses RQ2 identifying 10 different UAT techniques that were used by 37 studies as SRA. These techniques are: T2, T5, T6, T7, T9, T10, T11, T14, T15, and T21. Although some studies use T9 and T11 as SRA, we found that these techniques cannot been see as a solution to specify SR because many pre-requisites are required to use them.

Techniques classified as Table-driven Test and Behavior-driven were widely used by the selected studies as SRA. Some studies show that the use of T2 and T10 reduce misinterpretation and ambiguity of SR. However, studies remark that specification in table format is weak in details. On the other hand, other studies propose the use of T15 and T5 to specify SR. Besides support tables, these techniques also use a declarative prose based on DSLs that offers a more detailed description for SR than T2 and T10.

Also, other studies have pointed out the use of T6, T7 and T14, which are techniques based on natural language, to specify acceptance test scenarios as SR. However, the problems of these techniques are the same found in specifying SR using natural language. Another UAT technique used to specify SR is T21, which expresses acceptance test scenarios using diagrams. Previous studies show that non-technical individuals can express user requirements through this technique.

In addition, the column "Tool" in Table 1 identify techniques that have tools to support them, whereas, the column "Goal" shows for which purposes the UAT techniques were used in the studies, as follows: software validation (VAL), requirements clarification (CLA), requirements specification (SPEC), regression testing (RT), and improvement of quality in requirements gathering (QA).

4 CONCLUSIONS AND FUTURE WORKS

Besides software validation and requirements communication, the selected studies also show that UAT could support the execution of regression testing and the requirements gathering preventing ambiguous, incomplete, inconsistent, unexpressed, unusable or verbose requirements.

Although the data collected from the studies allowed us to show a list of UAT techniques and their features, it is not possible to define "what is the best one" empirically. As future work, we intend to use some of these UAT techniques to analyze their capability of communicating SR.

Table 1: A comparative table of UAT techniques classification and features

Id	Technique name	Nos of Studies	Artifact	Classification	Tool	Goal				
						VAL	*CLA*	*SPEC*	*RT*	*QA*
T1	Declarative Domain System Test Language.	1	Text	Behavior-driven	-	●	●	-	-	-
T2	FIT Tables.	18	Table	Table-driven Test	●	●	●	●	●	●
T3	Formal diagrams (Finite State Machine, Petri net and/or State Machine).	1	Diagram	Formal Diagrams	-	●	-	-	-	-
T4	Formal language.	5	Script	Formal Languages	●	●	-	-	-	-
T5	Gherkin language.	8	Text	Behavior-driven	●	●	●	●	-	●
T6	Natural Language with annotation	4	Text	Natural Languages	●	●	●	●	-	●
T7	Natural language.	1	Text	Natural Languages	-	●	●	●	-	-
T8	Record and Replay testing scenarios based on an executable system implementation.	3	Script	Record / Replay	●	●	-	-	-	-
T9	Record and Replay testing scenarios based on mockups.	1	Script	Record / Replay	●	●	-	●	-	-
T10	Spreadsheets.	2	Table	Table-driven Test	●	●	●	●	-	●
T11	State diagram supported by page links.	1	Diagram	Formal Diagrams	●	●	●	●	-	-
T12	Test Programming Language.	6	Script	Programming Language	●	●	-	-	●	-
T13	Test scenarios from a set of UML diagrams.	1	Diagram	Formal diagrams	-	●	-	-	-	-
T14	Test scenarios from Semi-Structured Natural Language.	4	Text	Natural languages	●	●	●	●	-	●
T15	Test scenarios from Structured (semi-formal) Natural Language.	13	Text	Behavior-driven	●	●	●	●	-	●
T16	Test scenarios from UML Collaboration Diagram.	1	Diagram	Formal Diagrams	-	●	-	-	-	-
T17	Test scenarios from UML State Diagram.	2	Diagram	Formal Diagrams	●	●	-	-	-	-
T18	Test scenarios from UML extended Activity Diagram.	3	Diagram	Formal Diagrams	●	●	●	-	-	●
T19	Test scenarios from Use Cases.	3	Diagram	Formal Diagrams	●	●	●	-	-	-
T20	Test Scenarios are written in Domain Specific Language (DSL) and generated from detailed specifications.	1	Text	Behavior-driven	●	●	-	-	-	-
T21	User Scenarios through User Interaction Diagram (US-UID).	2	Diagram	Formal Diagrams	●	●	●	●	-	●

REFERENCES

[1] G. Melnik and F. Maurer. 2005. The Practice of Specifying Requirements Using Executable Acceptance Tests in Computer Science Courses. In Companion to the 20th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications, 365–370.

[2] F. Ricca, M. T., M. Di Penta, M. Ceccato, and P. Tonella. 2009. Using acceptance tests as a support for clarifying requirements: A series of experiments. Information and Software Technology 51, 2: 270–283.

[3] D.H. Longo and P. Vilain. 2015. Creating user scenarios through user interaction diagrams by non-technical customers. In Proceedings of the International Conference on Software Engineering and Knowledge Engineering, SEKE, 330–335.

[4] B.A. Kitchenham. 2007. Guidelines for Performing Systematic Literature Reviews in Software Engineering Version 2.3. Technical Report. Software Engineering Group, Keele University and University of Durham.

[5] S. S. Park and F. Maurer. 2008. The Benefits and Challenges of Executable Acceptance Testing. In Proceedings of the 2008 International Workshop on Scrutinizing Agile Practices or Shoot-out at the Agile Corral, 19–22.

[6] J. Andrea. 2007. Envisioning Next-Generation Functional Testing Tools. IEEE Software 24, 3: 58–66.

[7] J. P. Sauvé and O. L. Abath Neto. 2008. Teaching Software Development with ATDD and Easyaccept. In SIGCSE Bull, 542–546.