Aligning Requirements and Acceptance Tests via Automatically Generated Guidance

Sofija Hotomski, Eya Ben Charrada, Martin Glinz
Department of Informatics, University of Zurich, Switzerland
Email: {hotomski, charrada, glinz}@ifi.uzh.ch

***Abstract*—Keeping requirements and acceptance test documents aligned and up-to-date plays an important role in the success of software projects. In practice, these documents are not always aligned with each other, nor with the actual system behaviour. A previous study showed that even when requirements are updated, acceptance tests might stay outdated, which often leads to quality problems and unintended costs. In order to keep the requirements and test documents in a consistent state, we are developing an approach that automatically generates guidance on how to change impacted acceptance tests when changes in requirements occur. In this paper, we briefly present our approach and a prototype tool that implements it. A preliminary evaluation of our approach yielded encouraging results.**

I. Introduction

Requirements and acceptance tests are related software artifacts that define and verify the features and behaviors of a system. In practice, requirements constantly evolve and, in most of the cases, changes in requirements impact acceptance tests. However, changes are not always propagated from requirements to the acceptance tests and acceptance test documents are not kept up-to-date. Having non-aligned documents increases the risk of the late discovery of a mismatch between stakeholders' expectation and the actual software behaviour, which is one of the main reasons for the failure of many software projects [14]. In an earlier study, we found that requirements and acceptance tests are not aligned mostly due to (1) the manual effort needed for keeping the documents in a consistent state, (2) preferable verbal communication between requirements and test engineers and (3) the separation of the requirements and testing activities [9].

Having weak communication between requirements engineers and test engineers often leads to confusing features and bugs [5]. For instance, if a requirement is extended by adding a new feature, but this change is not communicated to testers nor documented in the acceptance test, testers will report the actual feature to be a bug, regardless of the fact that the requirements document has changed [9]. Testers perform the steps specified in the acceptance tests and when the system does not behave the way it is specified in these tests, they report bugs without looking at the requirements documentation. Stakeholders then need a lot of time to identify and manage the issues caused by poor communication and outdated documents. This results in further software quality problems and project delays.

This work was partially funded by the Swiss National Science Foundation under grant 200021-157004/1.

In order to keep software documentation aligned and updated when a system evolves, many researchers focus their attention on improving change management practices. Much of the current research is focused on identifying which documents are related to each other and which of them are impacted by a change [6], [11], [13], while there is still research missing on how to manage this change. Nevertheless, the importance of change management is recognized among researchers. For instance, Nair et al. stated: "Practitioners will benefit on more guidance about how to deal with changes and what actions to perform, beyond only being aware of the artefacts potentially affected by a change" [13]. However, to the best of our knowledge, there is no concrete work done in this area. Therefore, in order to make a first step towards better requirements change management, we are developing an approach that automatically generates guidance on how to modify impacted acceptance tests, based on changes in requirements. By providing such guidance, we aim at supporting test engineers in making decisions on how to change the acceptance tests in order to keep them consistent with the rest of the system.

By guidance we refer to a list of suggestions. The suggestions are generated as soon as the changes in the requirements document are saved and they can be easily communicated to all interested parties via email. In this way we support requirements and test engineers in communicating changes on time and with less effort.

The paper is organized as follows. In the next section we present our approach. Then we describe the current state of our work in Section III. We discuss related work in Section IV. Section V concludes the paper with a summary and outlook.

II. Approach overview

The goal of the approach is to generate guidance on how to change the acceptance tests whenever the requirements are changed. This guidance will serve two main purposes. First, it will provide test engineers with concrete suggestions on how tests need to be changed in order to be aligned with the changed requirement. Second, it will be used to notify the test engineers about the relevant changes that occurred in the requirements and which should be taken into consideration when testing the updated software system.

Our approach has three main steps:

1. Identifying relevant change patterns: during this step, we compare the new requirement to the old one and identify the

relevant word classes (e.g. nouns, verbs, etc.) and their change types.

2. Generating guidance: in this step, we formulate suggestions in natural language on how to treat the changes.

3. Notifying subscribed parties: finally, the generated guidance are communicated to the relevant parties via email.

In the remainder of this section, we present each of these steps in more detail.

1) ***Identifying relevant change patterns****:* The goal of this step is to identify relevant change patterns that are applied to a requirement. A change pattern is characterized by the change type (add/delete/modify) that is applied to the sentence as well as the word classes (e.g. verb, noun) of the changed elements. Relevant change patterns are the ones whose changes require the acceptance tests to be adapted. In particular, relevant change patterns in our approach are the ones that directly or indirectly cause the change of some action, since acceptance tests contain a list of actions that should be performed.

Actions are generally expressed using verbs in English sentences. Therefore, we consider verbs as the principal element of analysis in our approach. More concretely, we consider a change in the requirements to be relevant if it involves an addition, deletion or modification of a verb or of another word class that relates to a verb such as:

- Nouns (subjects or objects) related to at least one *verb* or, recursively, to another noun that is related to at least one *verb*.
- Adjectives which refer to a noun that is related to at least one *verb*.

Nouns and adjectives can indirectly cause a change of an action that is being tested. For instance, if a sentence contains a verb and an object and if a new object is added, then the output of performing the action on the new object might differ from the output of performing that action on the existing object. In such a way, the change of a noun may cause a change in the acceptance test. Changes of other elements in a sentence, such as prepositions or articles, are not taken into consideration, since we assume that they do not influence any actions and, therefore, do not have an impact on acceptance tests.

Overall, we identified a set of 22 change patterns that can be applied to the requirements, e.g. "a verb is added", "a subject is modified", etc. An initial set of the patterns was identified by looking at real examples of changes that were applied to requirements. We then complemented these patterns with additional ones that we thought are plausible in a real context. The soundness of the patterns was then verified during the evaluation of our approach (see Section III).

For identifying a change type we adapted the algorithm implemented in a text-based diff engine called Text_diff [7]. For identifying a word class and for the dependency parsing we use Google's implementation of a globally normalized transition-based neural network model, called SyntaxNet [2].

For example, if the sentence: "User can see the button." is modified to "User can see and click on the button.", Text_diff will detect that the words "and", "click" and "on" are added. SyntaxNet will identify the word classes: conjuction ("and"), verb ("click") and preposition ("on"). Based on these word classes we then filter out the words other than verbs, nouns and adjectives (in this case "and" and "on"). Therefore, we take into consideration only the addition of the word "click".

2) ***Generating Guidance****:* The goal of this step is to generate suggestions about how to change the acceptance tests so that they stay aligned with the changed requirements. Every suggestion contains static and dynamic parts.

The static parts of a suggestion differ according to the change patterns identified in the previous step. For instance, if a whole sentence has been added to a requirement, the static part of the suggestion is "Add the steps which verify that". Accordingly, if a whole sentence has been deleted, the static part of the guidance is "Delete the steps which verify that". If a sentence has been modified, the static parts are formulated according to the modification type: whether a verb/subject/object/adjective is added/deleted/modified or a noun is changed from singular to plural, etc. For instance, if an object is modified, the static parts of the guidance are "Modify the step which verifies that ... Replace ... with ... in order to test that".

The dynamic parts of a suggestion fill in the gaps between the static parts. The dynamic parts contain words that have been changed as well as their related words. The related words belong to one of the relevant word classes: verbs, nouns (subjects and objects), adjectives, prepositions and articles. In order to find the related words we use the word classes, grammatical functions and dependencies between words.

In the example where "User can see the button" is modified to "User can see and click on the button.", our approach generates the guidance: "ADD THE STEPS WHICH VERIFY THAT *user can click on the button*". The static part (in capital letters) is derived from the fact that the relevant change, detected by our approach, is the addition of a verb ("click"). The dynamic part is constructed by analyzing the elements that relate to the newly added verb. For instance, by analyzing the sentence dependencies which we obtain from SyntaxNet, we can infer that the relevant elements related to "click" are *"user"* (subject), *"can"* (auxiliary verb), *"on"* (preposition), *"the"* (article) and *"button"* (object).We then order the related elements based on their word index that is generated by SyntaxNet, and we obtain the complete dynamic part, in this case *"user can click on the button"*.

Due to space limitation, we only present five more patterns and the guidance in Table I. We also include a concrete example for each of these patterns with the suggestions generated by our tool. Other patterns that are not covered in the table include, for example, addition/deletion/modification of conjunction verbs and changes of nouns from singular to plural and vice versa. In Table I, the static parts are written in capital letters and the dynamic parts are written in italic.

The presented examples contain at least one subject and object. However, we also take into consideration cases when no subject and/or object exist. For instance, if there is no subject the static part will contain "how to" instead of "that". E.g., if the sentence "Acceptance criteria: - add users" is added,

TABLE I
EXAMPLES OF CHANGE PATTERNS WITH CORRESPONDING GUIDANCE

	Change Pattern	Guidance
1	***An object is added to the requirement** Example: Old requirement: User can choose a country. New requirement: User can choose a country **and its region**.	***Add new steps to verify the action(s) over the new object** ADD NEW STEPS TO VERIFY THAT *user can choose a country region*.
2	***A subject is deleted from the requirement** Example: Old requirement: Admin **and user** can modify personal data of that user. New requirement: Admin can modify personal data of a user.	***Delete the steps which verify that this subject can perform some existing actions** DELETE ALL THE STEPS WHICH VERIFY THAT *user can modify personal data,* SINCE *user* CANNOT PERFORM THIS ANYMORE.
3	***An object is modified in the requirement** Example: Old requirement: Admin can add new **members**. New requirement: Admin can add new **users.**	***Replace an old object with the new one in the impacted acceptance test** WORD *"members"* IS MODIFIED TO: *"users"*. MODIFY THE STEP WHICH VERIFIES THAT *admin can add new members.* REPLACE *"members"* WITH *"users"* IN ORDER TO TEST THAT *admin can add new users.*
4	***An auxiliary verb is added** Example: Old requirement: I see the icon in upper, right corner. New requirement: I **don't** see the icon in upper, right corner.	***Update the acceptance test by taking the meaning of the auxiliary verb** MAKE SURE THAT NOW *I don't see the icon in upper, right corner.*
5	***A sentence which contains at least one verb is added** Example: Old requirement: I don't see the icon in upper, right corner. New requirement: I don't see the icon in upper, right corner. **The icon appears after I click to add new user.**	***Add steps to verify new action(s)** ADD NEW STEPS TO VERIFY THAT *the icon appears after I click to add new user.*

the generated guidance is: "Add steps which verify how to add users".

*3) **Notifying subscribed parties**:* We found in our previous study that testers are often not informed about changes in requirements [9]. Therefore, when a test fails, testers need a significant amount of time to realize that the cause is not a bug in the source code, but a change in the requirements. In order to support faster communication, we implemented a notification system that allows requirements engineers to send an email to test engineers with relevant changes and guidance generated for that change as soon as the change is saved. The explanation of what has been changed and the guidance on how the acceptance tests need to be updated should support the test engineers in performing the update with minimal effort and verbal communication. It will also contribute to a faster and more efficient communication between requirements and test engineers. Every guidance refers to one change in the requirement. If the change is only refactoring that has no influence on the related acceptance test, the requirements engineer can ignore the generated guidance and the email will not be sent.

III. CURRENT STATE OF WORK

Besides the conceptual solution, we implemented a *tool prototype* as a dynamic web application. The tool provides users with an option to upload the list of requirements, to make changes to each of them, to notify the subscribed test engineers about the changes and to send them guidance about how to modify the impacted acceptance tests. As soon as the changes in the requirements are made and saved, a guidance is automatically generated per each change and it could be sent with one click to the subscribed users.

We performed a *preliminary evaluation* of our approach by applying it to real-world data and obtained encouraging results. The data includes 28 changes made in 20 user stories that we obtained from a medium size industrial company in Switzerland. For each user story, we have both an old and a new version as well as the corresponding acceptance tests. We used our tool prototype to generate guidance for each change. Then we asked a requirements engineer and a test engineer from the company to assess the correctness and clarity of the result. For 25 of the 28 changes, they rated our guidance as correct in terms of the actions to be performed. With regard to clarity, they found all of the generated guidance to be understandable although five messages had some grammar issues. Finally, the engineers reported that for 21 cases they would be able to perform the update based on our guidance without needing any further clarification.

IV. RELATED WORK

Both researchers and practitioners are interested in binding requirements and testing activities more closely together [17], [5], [16], [10]. However, these activities, especially the documentation management, are still separated and performed by different people [9]. In order to keep the software documents aligned, many suggested traceability for change impact analysis using Information Retrieval (IR) and Natural language Processing (NLP) techniques. However,

these methods are mostly applied between source code artifacts and textual documents [12], [11], [8], while there is no research on using them in order to align requirements and acceptance tests. Arora et al. [3] proposed an approach based on NLP for analyzing the impact of changes in natural language requirements. They present a method for tracing impacted requirements when an existing requirement changes. All these methods focus on only identifying impacted documents, while we try to go a step further and to provide guidance on how impacted documents (acceptance tests) should be modified according to the change in another document (a requirements document).

Another problem that draws the attention of both, researchers and practitioners, is bridging the communication gap between requirements engineers and other parties involved in the software evolution process. Sinha et al. [15] defined and explained the communication problems when managing requirements in distributed environment. Bjarnason and Sharp [4] and Adzic [1] clearly emphasize the communication problems between requirements engineers, test engineers (managers), developers and testers in agile projects and propose guidance on how to bridge this communication gap. With an automated generation of guidance in natural language that can be sent on one click to the interested parties, our approach supports easier and "on-time" communication between requirements and test engineers.

V. Conclusion

In this paper, we presented an automated approach and a tool for generating guidance on how to change the impacted acceptance tests when requirements change. The guidance will provide support for test engineers when making decisions on what should be modified in acceptance tests according to changes in the requirements. Furthermore, our approach provides a notification system that enables timely and easily traceable communication of the requirements changes between requirements engineers and test engineers.

In our future work, we will refine our approach and conduct a thorough industrial evaluation to assess the correctness, completeness and usefulness of our approach. For this evaluation, we are currently collecting further data from other companies.

Acknowledgements

We thank the aforementioned company for providing us their data and supporting the evaluation of our approach.

References

[1] G. Adzic. *Bridging the communication gap: specification by example and agile acceptance testing*. Neuri Limited, 2009.

[2] D. Andor et al. "Globally normalized transition-based neural networks". In: *arXiv preprint arXiv:1603.06042* (2016).

[3] C. Arora et al. "Change impact analysis for natural language requirements: An NLP approach". In: *23rd Int. Requirements Engineering Conference (RE)*. IEEE. 2015, pp. 6–15.

[4] E. Bjarnason and H. Sharp. "The role of distances in requirements communication: a case study". In: *Requirements Engineering* 22.1 (2017), pp. 1–26.

[5] E. Bjarnason et al. "Challenges and practices in aligning requirements with verification and validation: a case study of six companies". In: *Empirical Software Engineering* 19.6 (2014), pp. 1809–1855.

[6] M. Borg, O. CZ Gotel, and K. Wnuk. "Enabling traceability reuse for impact analyses: A feasibility study in a safety context". In: *7th Int. Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE)*. IEEE. 2013, pp. 72–78.

[7] J. Schneider C. Hagenbuch. *Text_Diff - Engine for performing and rendering text diffs*. URL: https://pear.horde.org/ (visited on 06/20/2017).

[8] E. Ben Charrada, A. Koziolek, and M. Glinz. "Identifying outdated requirements based on source code changes". In: *20th Int. Requirements Engineering Conference (RE)*. IEEE. 2012, pp. 61–70.

[9] S. Hotomski, E. Ben Charrada, and M. Glinz. "An Exploratory Study on Handling Requirements and Acceptance Test Documentation in Industry". In: *24th Int. Requirements Engineering Conference (RE)*. IEEE. 2016, pp. 116–125.

[10] J. Larsson and M. Borg. "Revisiting the challenges in aligning RE and V&V: Experiences from the public sector". In: *1st Int. Workshop on Requirements Engineering and Testing (RET)*. IEEE. 2014, pp. 4–11.

[11] A. Lucia et al. "Information Retrieval Methods for Automated Traceability Recovery". In: *Software and Systems Traceability* (2012), pp. 71–98.

[12] A. Marcus, J. I Maletic, and A. Sergeyev. "Recovery of traceability links between software documentation and source code". In: *Int. Journal of Software Engineering and Knowledge Engineering* 15.05 (2005), pp. 811–836.

[13] S. Nair, J. L. de la Vara, and S. Sen. "A review of traceability research at the requirements engineering conference RE@ 21". In: *21st IEEE Int. Requirements Engineering Conference (RE)*. IEEE. 2013, pp. 222–229.

[14] F. Ricca et al. "Using acceptance tests as a support for clarifying requirements: A series of experiments". In: *Information and Software Technology* 51.2 (2009), pp. 270–283.

[15] V. Sinha, B. Sengupta, and S. Chandra. "Enabling collaboration in distributed requirements management". In: *IEEE Software* 23.5 (2006), pp. 52–61.

[16] M. Unterkalmsteiner et al. "Assessing requirements engineering and software test alignment. Five case studies". In: *Journal of Systems and Software* 109 (2015), pp. 62–77.

[17] E. J Uusitalo et al. "Linking requirements and testing in practice". In: *16th IEEE Int. Requirements Engineering Conference*. IEEE. 2008, pp. 265–270.