GuideGen – A Tool for Keeping Requirements and Acceptance Tests Aligned

Hotomski, Sofija ; Glinz, Martin

DOI: https://doi.org/10.1145/3183440.3183484

GuideGen – A Tool for Keeping Requirements and Acceptance Tests Aligned

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

ABSTRACT

When changes in requirements occur, their associated tests must be adapted accordingly in order to maintain the quality of the evolving system. In practice, inconsistencies in requirements and acceptance tests—together with poor communication of changes—lead to software quality problems, unintended costs and project delays. We are developing GuideGen, a tool that helps requirements engineers, testers and other involved parties keep requirements and acceptance tests aligned. When requirements change, GuideGen analyzes the changes, automatically generates guidance on how to adapt the affected acceptance tests, and sends this information to subscribed parties. GuideGen also flags all non-aligned acceptance tests, thus keeping stakeholders aware of mismatches between requirements and acceptance tests. We evaluated GuideGen with data from three companies. For 262 non-trivial changes of requirements, the suggestions generated by GuideGen were correct in more than 80 percent of the cases for agile requirements and about 67 percent for traditional ones.
Demo video: https://vimeo.com/254865530

CCS CONCEPTS

• **Software and its engineering** → *Software notations and tools*; *Requirements analysis*; *Acceptance testing*; *Software evolution*;

ACM Reference Format:
Sofija Hotomski and Martin Glinz. 2018. GuideGen – A Tool for Keeping Requirements and Acceptance Tests Aligned. In *ICSE '18 Companion: 40th International Conference on Software Engineering , May 27-June 3, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3183440.3183484

1 INTRODUCTION

Weak alignment of requirements-related activities with validation and verification tasks leads to software quality problems, unintended costs, wasted effort and delays. In particular, when requirements documents are not aligned with their corresponding acceptance tests, features might be incorrectly verified or not verified at all [4]. Maintaining the alignment is a difficult task due to many factors, such as poor communication of changes among team members and manual tracing between requirements and acceptance tests. For instance, when a requirements engineer changes a requirement, but does not timely communicate this to developers and testers, testers will report bugs for newly added or removed features. Time and effort is wasted for understanding the causes of such problems and making the documentation consistent again [9]. Furthermore, communicating changes explicitly is time-consuming and prone to misunderstandings. Problems also stem from using different tools for managing requirements and acceptance tests where traces between artifacts must be established and maintained manually. In some companies, requirements and tests are still captured in spreadsheets or text documents, having barely or no traces between documents.

We are developing GuideGen, a tool that keeps requirements and acceptance tests closely together and aligned when requirements evolve. The core idea of our tool, which distinguishes it from other document management tools, is that it automatically generates guidance on how to adapt the impacted acceptance tests according to changes in requirements. In such a way, GuideGen supports requirements engineers in easily communicating requirements changes to testers and test engineers in properly adapting acceptance tests when their related requirements change. In addition, GuideGen automatically creates the traces between requirements and acceptance tests as soon as a new acceptance test is added for a requirement.

In this paper, we focus on the main features of the tool and how they are used. The technical details as well as the evaluation of the GuideGen approach are described in [10] and [11]. We describe the main features of the tool in Section 2 and summarize the evaluation in Section 3. Section 4 briefly discusses related work.

2 THE GUIDEGEN TOOL

GuideGen is a web application, written in Java using Servlet and JSP technology [8]. It is deployed on Apache Tomcat [19]. On the one hand, GuideGen supports requirements engineers in maintaining the requirements of a system and in communicating all changes of requirements to testers, developers and other interested parties on-time and with almost no effort. On the other hand, GuideGen supports testers, who maintain acceptance test documents, by providing them with guidance on how to modify impacted tests so that they stay aligned with the modified requirements. In addition, by flagging all non-aligned acceptance tests, any stakeholder can easily see which acceptance tests are currently mis-aligned with their corresponding requirement—be it that tests do not exist yet or that they have not been updated after changes in the requirements.

The current version of GuideGen is limited to one acceptance test per requirement. This is no severe limitation: in a previous study [9] we found that a one-to one correspondence between requirements and acceptance tests frequently occurs in practice.

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

ACM ISBN 978-1-4503-5663-3/18/05.
https://doi.org/10.1145/3183440.3183484

2.1 Using GuideGen

Upon starting the application, a user can (1) upload a collection of existing requirements and acceptance tests into GuideGen, (2) view the current list of requirements and acceptance tests, or (3) add a new requirement. The resulting list is shown in Figure 1.

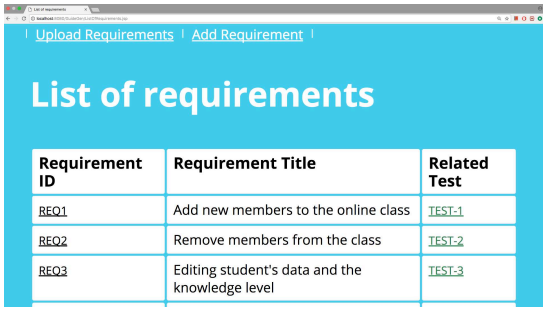

Figure 1: List of requirements and their acceptance tests.

The acceptance tests shown in Figure 1 are all aligned with their requirements—they are green and no warning flags are shown. When a user clicks on the identifier of a requirement or acceptance test, GuideGen shows the selected requirement or test in detail and the user can edit it. Subsequently, we describe three typical scenarios for working with GuideGen.

Scenario 1: Changing a requirement. Let's assume that a requirements engineer selects and edits the requirement REQ3:

A student can edit his personal data ~~and the knowledge level.~~ Only teachers can edit students' knowledge level.
- The knowledge level can be: – beginner, – intermediate, – advanced, – proficient. This change is due to users' feedback.

Added words are green and underlined, removed ones are red and struck through, while unchanged text is shown in black. As soon as the requirements engineer saves the changes, GuideGen identifies the relevant change patterns and, within about three seconds, generates guidance on how to adapt the associated acceptance test. Figure 2 shows the guidance generated for the previously presented changes.

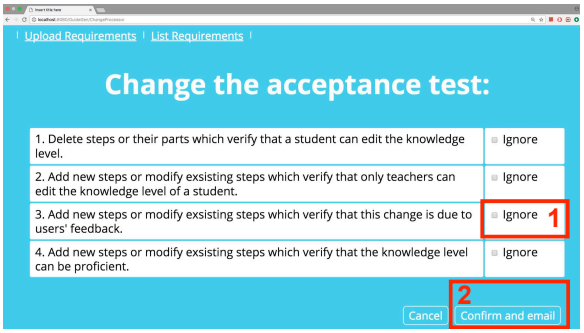

Figure 2: List of generated suggestions. A user may ignore suggestions (1) and e-mail the other ones to interested parties (2).

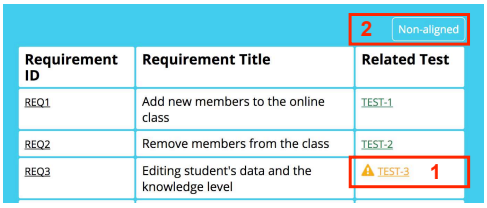

Figure 3: The warning flag showing a mismatch (1) and the button for displaying the non-aligned documents only (2).

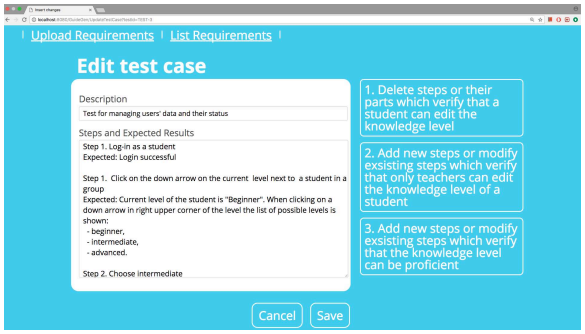

Figure 4: The form for editing TEST-3.

The guidance generated by GuideGen consists of a suggestion per added, deleted or modified sentence. GuideGen lets the requirements engineer review the suggestions and allows her to mark those that she considers irrelevant or wrong (Fig. 2). As soon as she confirms, GuideGen automatically creates an e-mail and sends it to all GuideGen users who have subscribed for receiving change alerts. This e-mail consists of three parts: (1) the test case to be modified, (2) the generated suggestions, and (3) the original and the changed requirement. GuideGen also flags non-aligned tests in the list of requirements and acceptance tests with yellow color and a warning sign, as shown in Figure 3.

Scenario 2: Adapting an acceptance test. Let's assume that a tester receives an e-mail about the change in requirement REQ3 as described above, reads the message, and decides to adapt the impacted acceptance test. After launching GuideGen, the tester clicks TEST-3 which is flagged as non-aligned by GuideGen (see Figure 3). GuideGen then displays a form for editing the test case (see Figure 4). In the edit field on the left, the current version of the test is displayed. On the right side, the suggestions generated by GuideGen about how to change the test case are displayed. The tester can now edit the test according to the given suggestions, without having to analyze the changed requirement manually. When he saves his changes, GuideGen will remove the warning in the requirements list and display the test-ID in green.

Scenario 3: Overview of non-aligned tests. A stakeholder can get an overview of all acceptance tests which are currently not aligned by opening the list of requirements and acceptance tests in GuideGen (Figure 3) and filtering it for non-aligned entries by clicking the "Non-aligned" button.

2.2 The Algorithms Used in GuideGen

In this subsection, we briefly describe how our approach works. GuideGen first identifies relevant change patterns and then generates the text of the suggestions. For more details see [11].

Identifying relevant change patterns. Relevant change patterns characterize those changes in requirements that require the associated acceptance test to be adapted. We classify the changes into relevant and irrelevant ones by analyzing changes both on the sentence level and on the level of individual words.

In the analysis on the *sentence level*, we identify added, deleted and modified sentences. First, we split the old version of the changed requirement (oldReq in further text) and the new version (newReq) into sentences using an implementation of the Stanford sentence splitting algorithm [13]. We then compare the sentences from oldReq with the sentences from newReq by using a semantic similarity toolkit [17]. If the similarity between a sentence in oldReq and one in newReq is equal to one, that sentence is unchanged. For distinguishing between deleted and modified sentences, we use an experimentally determined similarity threshold of 0.6. If for a sentence s in oldReq there is no sentence in newReq with a similarity score above the similarity threshold, we consider s to be deleted. Otherwise, s has been modified. We identify the corresponding sentence s' in newReq by choosing the pair (s, s') with the highest similarity score among all candidate pairs.

When we remove best matches, unchanged sentences and already identified deleted sentences, the leftovers in newReq are added sentences and the leftovers in oldReq are deleted sentences.

For modified sentences, we then analyze the changes on the *word level*. First, we split each modified sentence into a list of words. Then our diff engine determines whether the words have been added, deleted, modified or are unchanged. We adapted the algorithm implemented in Text_Diff [7], so that we get the changes at a word level, instead of a phrase level as in the original implementation. We then identify a word class (e.g. noun, verb) and grammatical function (e.g. subject, object) for each word. For identifying word classes we use SyntaxNet [2]. SyntaxNet also identifies dependencies between words, presented as dependency numbers, which is used later when we generate suggestions. An example of the output of SyntaxNet is shown in Figure 5.

ID	TEXT	WORD CLASS	DEPENDENCY NUMBER	GRAMMATICAL FUNCTION
1	A	DET	2	det
2	student	NOUN	4	nsubj
3	can	VERB	4	aux
4	edit	VERB	0	ROOT
5	personal	ADJ	6	amod
6	data	NOUN	4	dobj
7	and	CONJ	6	cc
8	the	DET	10	det
9	knowledge	NOUN	10	nn
10	level	NOUN	6	conj
11	.	PUNCT	0	ROOT

Figure 5: The SyntaxNet output for the old version of the first sentence in the example shown above.

Finally, we classify the changes into relevant and irrelevant ones. As acceptance tests contain a list of actions to be performed, and actions are generally expressed using verbs in English sentences, we consider a change in a requirement 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, objects). Changes of other word classes, such as determiners, relative pronouns or prepositions are irrelevant, since we assume that they do not influence any actions.

Generating guidance. For all relevant changes, we now formulate a list of suggestions on how to adapt the affected test cases. We identify static parts first, then we identify dynamic parts and finally we combine them to formulate a suggestion.

The static parts are identified by analyzing the change patterns. For instance, when a whole sentence has been added, the static part is "Add new steps or modify existing steps to verify that". When a sentence has been deleted, the static part is "Delete steps or their parts 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 or number has been added, deleted or modified. For instance, if a subject has been added, the static parts of the suggestion are "Make sure that now *+{dynamic part}*" and "Add the steps which verify this activity."

We then identify *the dynamic parts*. When a whole sentence has been added or deleted, the dynamic part contains that sentence. When a sentence has been modified, we formulate the dynamic parts by finding and sorting the related words of the changed word. For instance, when "the knowledge level" has been deleted, we identify that the object "level" refers directly to the noun "data" by a conjunction (see Figure 5). In this case we find verbs and subjects that are related to the main object ("data"). The verb "edit" with its auxiliary verb "can" is directly related to the noun "data" and the subject "student" with its determiner "a" directly refers to the verb "edit". The noun "knowledge" with its determiner "the" directly refers to the object "level". We sort the words by their index and formulate the dynamic part of the suggestion: "a student can edit the knowledge level". Since the change pattern is "an object is deleted", the static part is "Delete steps or their parts which verify that". Finally, when we combine the static and the dynamic part, the following suggestion is generated: *"Delete steps or their parts which verify that a student can edit the knowledge level."*

GuideGen is applicable for requirements and acceptance tests written in (free-form or structured) natural language. We have not explored generating suggestions for how to change the test code when using automated, executable tests.

3 EVALUATION

In this section, we summarize the results of a first evaluation of the GuideGen approach with real-world data. The details are described in [11].

Study design. We obtained 262 requirements changes from three companies. Companies C1 and C2 work with user stories, while company C3 writes traditional textual requirements. For every change, we used GuideGen to generate guidance about how to change the associated acceptance test. We then asked experts from the three companies to evaluate the generated suggestions with respect to seven questions: (1) correct with respect to suggested actions? (2) grammatically correct? (3) complete? (4) understandable? (5) self-explanatory?, (6) unnecessary?, and (7) any missed changes?

Results. For companies C1 and C2, the experts found more than 80% of the suggestions to be correct in terms of actions. For C3, there was some disagreement among the experts resulting in 66.7% of the suggestions being considered correct on average. Grammatical correctness ranged from 67.5% (C2) to over 80% (C1 and C3). Over 93% of all suggestions were considered to be complete and understandable by all experts. Over 70% were considered to be self-explanatory. With respect to necessity, results were good for C1 and C2: the experts found only 7 to 10% of the suggestions to be unnecessary. In C3, however, two experts considered about 30% of the suggestions unnecessary and one expert even found 50% to be unnecessary. The rate of missed changes (i.e., no suggestion was generated although it would have been appropriate to create one) was only 3 to 6%. When comparing our results with a gold standard of 100% correct suggestions, GuideGen achieves a very high recall (94-97%) with rather high accuracy.

In follow-up interviews, all experts deemed GuideGen to be useful for communicating changes on time and with less effort, and helpful for test engineers to update acceptance tests. They also preferred the explicit guidance produced by GuideGen over just being notified about changes in requirements.

4 RELATED WORK

In order to keep software documentation consistent when a system evolves, researchers propose information retrieval [6], [12] or natural language processing [3] methods to automatically identify which documents are related to each other and which of them are impacted by a change. However, there is little research about how to actually update impacted documents, although it would be beneficial to have guidance about how to handle the changes and what actions to perform [15]. Our tool provides concrete suggestions for testers on how to handle changes.

Many researchers studied the role of communication in the success of software projects and its challenges [18], [5], [1]. Some suggest that face-to-face communication should be the main source of knowledge sharing, since writing e-mails or documents is time-consuming [14]. However, having only verbal communication introduces the risk of misunderstandings and miscommunicated changes due to many factors, such as geographic, temporal, cultural, and linguistic distance [16]. GuideGen reduces the effort needed for communicating changes by automatically providing e-mails with information regarding the change.

5 CONCLUSION

Summary. We presented GuideGen, a tool that (1) automatically generates guidance on how to align acceptance tests with evolving requirements, (2) provides e-mail notifications with that guidance and a summary of changes, and (3) sets warning flags that make stakeholders aware of mismatches between requirements and associated acceptance tests. The evaluation of our approach shows promising results with respect to the correctness, completeness and understandability of the generated guidance.

Limitations. Currently, GuideGen is a standalone tool, which limits its applicability in projects that use existing tool chains for managing requirements and tests. This is due to the fact that our tool is part of ongoing research where the focus is on principles rather than on features. As mentioned above, the current version is also limited to one acceptance test per requirement. Further, GuideGen does not check whether the generated suggestions are actually followed when an acceptance test is updated.

Future Work. We are currently evaluating the GuideGen approach with respect to its usefulness and applicability in industrial practice. We also plan to address the current limitations of the GuideGen tool.

ACKNOWLEDGEMENTS

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

REFERENCES

[1] Gojko Adzic. 2009. *Bridging the communication gap: specification by example and agile acceptance testing.* Neuri Limited.
[2] Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins. 2016. Globally normalized transition-based neural networks. *arXiv preprint arXiv:1603.06042* (2016).
[3] Chetan Arora, Mehrdad Sabetzadeh, Arda Goknil, Lionel C Briand, and Frank Zimmer. 2015. Change impact analysis for natural language requirements: An NLP approach. In *23rd IEEE International Requirements Engineering Conference (RE'15)*. 6–15.
[4] Elizabeth Bjarnason, Per Runeson, Markus Borg, Michael Unterkalmsteiner, Emelie Engström, Björn Regnell, Giedre Sabaliauskaite, Annabella Loconsole, Tony Gorschek, and Robert Feldt. 2014. Challenges and practices in aligning requirements with verification and validation: a case study of six companies. *Empirical Software Engineering* 19, 6 (2014), 1809–1855.
[5] Elizabeth Bjarnason and Helen Sharp. 2017. The role of distances in requirements communication: a case study. *Requirements Engineering* 22, 1 (2017), 1–26.
[6] Jane Cleland-Huang, Brian Berenbach, Stephen Clark, Raffaella Settimi, and Eli Romanova. 2007. Best practices for automated traceability. *IEEE Computer* 40, 6 (2007), 27–35.
[7] Chuck Hagenbuch and Jan Schneider. 2017. Text_Diff-Engine for performing and rendering text diffs. (2017). https://pear.horde.org/
[8] Marty Hall. 2003. *Core servelet and JavaServer Pages, online version.* Sun Microsystems Press available at: http://www. coreservlets. com.
[9] Sofija Hotomski, Eya Ben Charrada, and Martin Glinz. 2016. An Exploratory Study on Handling Requirements and Acceptance Test Documentation in Industry. In *24th IEEE International Requirements Engineering Conference (RE'16')*. 116–126.
[10] Sofija Hotomski, Eya Ben Charrada, and Martin Glinz. 2017. Aligning Requirements and Acceptance Tests via Automatically Generated Guidance. In *4th Workshop on Requirements Engineering and Testing (RET)*.
[11] Sofija Hotomski, Eya Ben Charrada, and Martin Glinz. 2018. Keeping Evolving Requirements and Acceptance Tests aligned with Automatically Generated Guidance. In *International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ 2018)*.
[12] Andrea Lucia, Andrian Marcus, Rocco Oliveto, and Denys Poshyvanyk. 2012. Information Retrieval Methods for Automated Traceability Recovery. *Software and Systems Traceability* (2012), 71–98.
[13] Christopher D Manning, Mihai Surdeanu, John Bauer, Jenny Rose Finkel, Steven Bethard, and David McClosky. 2014. The Stanford coreNLP natural language processing toolkit. In *ACL (System Demonstrations)*. 55–60.
[14] Grigori Melnik and Frank Maurer. 2004. Direct verbal communication as a catalyst of agile knowledge sharing. In *Agile Development Conference*. 21–31.
[15] Sunil Nair, Jose Luis de la Vara, and Sagar Sen. 2013. A Review of Traceability Research at the Requirements Engineering Conference (RE@21). In *21st IEEE International Requirements Engineering Conference (RE'13)*. 222–229.
[16] John Noll, Sarah Beecham, and Ita Richardson. 2011. Global Software Development and Collaboration: Barriers and Solutions. *ACM Inroads* 1, 3 (2011), 66–78.
[17] Vasile Rus, Mihai C Lintean, Rajendra Banjade, Nobal B Niraula, and Dan Stefanescu. 2013. SEMILAR: The Semantic Similarity Toolkit. In *ACL (Conference System Demonstrations)*. 163–168.
[18] Vibha Sinha, Bikram Sengupta, and Satish Chandra. 2006. Enabling collaboration in distributed requirements management. *IEEE Software* 23, 5 (2006), 52–61.
[19] Chanoch Wiggers, Ben Galbraith, Vivek Chopra, Sing Li, Debashish Bhattacharjee, Amit Bakore, Romin Irani, Sandip Bhattacharya, and Chad Fowler. 2004. *Professional Apache Tomcat.* John Wiley & Sons.