Improving the Completeness of Acceptance Criteria

Astrid Rohmann

Institute for Computer Science, Heidelberg University, INF 205, 69120 Heidelberg, Germany

Abstract

[**Context and Motivation**] Acceptance criteria are a popular textual notation, especially in connection with user stories. Typically, they express refined requirements and are used to derive test cases, sometimes automatically. [**Question/Problem**] While acceptance criteria are very useful, they are not always documented or not always documented well. [**Principle ideas/results**] In order to find indications for more acceptance criteria (semi-) automatically for a given set of user stories and acceptance criteria we want to use information that is available in user stories and acceptance criteria or public knowledge. [**Contribution**] This proposal discusses the research problem of improving the completeness of acceptance criteria. It then introduces potential solutions to that problem that only use available information. It also discusses the used research method, plan and progress so far as well as related work to the research.

Keywords

Acceptance Criteria, User Stories, Natural Language Processing

1. Problem

Acceptance criteria (AC) are a common means in agile projects to check whether a user story (US) is fulfilled [23]. A US is a requirements notation that often follows the template '*As a <role>, I want < goal> [so that < reason>]'*, which is known as Cohn's template [5]. An example for a US with AC is the following:

As an employee I want to be able to book a half day of leave in the year in which the leave day is available so that I can use up my half days

- Checkbox hidden if no half leave day available.

- Start date must be equal to end date (in the respective year)

- Start date decides on "current" year

AC can be used to specify more details that need to be considered during implementation and they can be used to derive test cases. The derivation of test cases can already be done automatically, as shown by Fischbach et al. [10]. However, they also found out that in the two industry projects they used, only 31.1% and 50.1% of the US contained AC. We made similar observations in two datasets that we acquired from industry where only around 30% or less of the USs contained AC. The lack of AC is also mentioned by Hoda & Murugesan [11] as one of the challenges for agile teams because it lets software teams struggle to implement the correct software.

One of the quality criteria for requirements mentioned by Fabbrini et al. [8] is completeness. Similarly, if some or all AC of a US are missing, we define this as an *incomplete set* of AC for a US. In the example above none of the AC covers that the half day of leave will be booked, therefore, this set of AC is incomplete. Our goal is to find ways to support requirements engineers by (semi-)automatically improving the completeness of sets of AC in natural language for US. We only want to use information available in the US or AC or publicly available so that completeness can be judged

In: A. Ferrari, B. Penzenstadler, I. Hadar, S. Oyedeji, S. Abualhaija, A. Vogelsang, G. Deshpande, A. Rachmann, J. Gulden, A. Wohlgemuth, A. Hess, S. Fricker, R. Guizzardi, J. Horkoff, A. Perini, A. Susi, O. Karras, A. Moreira, F. Dalpiaz, P. Spoletini, D. Amyot. Joint Proceedings of REFSQ-2023 Workshops, Doctoral Symposium, Posters & Tools Track, and Journal Early Feedback Track. Co-located with REFSQ 2023. Barcelona, Catalunya, Spain, April 17, 2023.

EMAIL: rohmann@informatik.uni-heidelberg.de

without extra knowledge of project experts. This ensures that our approach can also be used by novice project members. We also do not want to use information from diagrams or other project documentation as it may vary from project to project which documentation and diagrams are used. Given a dataset with US and some AC we therefore only want to use the text of the US and the text of the AC and public information to find more AC for US in the dataset by using natural language processing (NLP) or machine learning (ML), especially deep learning (DL) algorithms (as defined in [25]).

The remainder of the paper is structured as follows: Section 2 describes the research method used to find solutions for our goal. Section 3 describes our proposed solution and our available datasets in more detail. Section 4 describes related work. Section 5 describes the research plan and current progress.

2. Research Method

We use the design science research method [24]. With this method we want to achieve the following **design goal:** *Design and evaluate an approach to improve the completeness of AC for US by using only available information.* The first step of the research method is to analyze the as-is-state in research and practice. That is the problem investigation. Then, solution ideas are implemented based on the as-is-state in a software prototype as treatment design. Finally, treatment validation is performed to get empirical evidence that the solution ideas and the prototype solve the problems.

Our **problem investigation** consists of three parts.

1. An interview study with practitioners from industry on the creation and usage of AC.
2. The analysis of datasets from industry with US and AC.
3. A systematic mapping study conducted by the guidelines of Kitchenham and Charters [13] to find suitable algorithms that could solve our problem.

We already performed the first step with the following research question:

RQ1: What is the state of practice regarding documentation and usage of acceptance criteria and are there improvement ideas?

For the second step, we used the following research question:

RQ2: What is the quality of acceptance criteria in practice and how are they documented?

After the first two steps of the problem investigation we could narrow down our problem and define our research goal as the goal described in the first section. We also did some first searches for literature that describes research on how to generate more AC for US or other requirements, but only found two approaches described in section 4 that don't use US as requirements notation. Therefore, we want to focus our systematic mapping study on possible solution ideas with the following question:

RQ3: What is the state of the art in extracting and comparing information from user stories and other requirements texts with NLP, ML and DL algorithms?

To answer this question we first will view the results of existing literature surveys [20], [25] and consider if they are relevant to answer the question. Then we will do forward snowballing on the relevant papers to see if there is more recent research that was not covered in the existing literature surveys. To be sure that there is nothing left out search terms can be defined from the found results and used in the typical search locations like ACM, IEEE, etc. A possible search term could be the following: ("requirement*" OR "user story" OR "user stories") AND ("NLP" OR "natural language processing" OR "machine learning" OR "deep learning") AND ("extract*" OR "compare" OR "similar*") AND ("information" OR "concept").

The first step of **treatment design** is to derive solution ideas from the as-is state. For this, the results of the systematic mapping study will be evaluated how they could be used to solve our problem. We will define criteria and select the approaches to implement. They will be implemented as algorithms in a prototype software. In the **treatment validation,** the algorithms will be evaluated on data sets with US and AC in natural language that we acquired from industry. Those data sets are described in the next section. We are also looking for suitable open source data, that can be used to evaluate our solution as those could be shared with the research community. The solution will be evaluated in the lab based on metrics like precision and recall. Our goal is to reach high recall (~0.8) with reasonable precision (~0.5).

The contribution of the thesis will be the interviews with practitioners on AC, coded datasets, a mapping study on algorithms for information extraction and comparison and the prototypical implementation of algorithms that can help to find more AC.

3. Proposed Solution

The general idea of our proposed solution is to apply NLP, ML or DL algorithms to available information in order to suggest indications for additional AC. The information that is available are the texts of the US, the texts of the AC and publicly available information (e.g. lexical databases like WordNet [18], pre-trained classifiers e.g. [4], [12], user forums etc.). More specifically for a US we can identify the role, goal and reason parts and we can use other parts of the text like single words, bigrams, concepts or the whole text. As AC in general do not have a structure, the available information are only the text or parts of the text like single words, concepts.

We want to use the information in the following ways to find indications for additional AC (information like lexical databases or pre-trained classifiers might be used in all approaches):

The first approach is to compare the information from the US with the information from its AC to find information in the US that is not covered by its AC. The uncovered information will be presented to the requirements engineer and can give hints for additional AC. In this approach the used information will be concepts, i.e. verbs, subjects and objects of the US and AC. For the US, we only want to use the concepts of the goal part as this part is the most important part for the described functionality of the software that should be covered in AC. The role is not important in AC, as AC focus on the software functionality, not the user. The reason part describes the rationale of the goal and does not give further insights into the needed functionality. This first approach is only able to find additional AC for information that is present in the US. Therefore, it can only find missing functional AC, but no AC that concern quality criteria or other constraints that are not mentioned in the US.

Typically, AC contain more details than the US, as they should refine the US and answer questions wrt functional behavior, quality characteristics, constraints, etc. [23] (see also the example giving in the beginning). Thus, we want to use information from other US and AC to get indications for these details. Furthermore, we assume that similar US have similar AC. Datasets we acquired from industry also indicate that similar US have similar AC, but this still needs to be shown empirically. Therefore, *the second approach* is to find similar US for one or more given US in a given set of US and show the requirements engineer the similar US together with their AC. These AC could either be used directly or they can give ideas for new AC for the given US. The definition for similarity we use in this approach is the following: Two US are similar if they either refer to the same feature or workspace (that means group of related functions and data) or if they have a similar action. But it might be necessary to change this definition for similar US if our empirical analysis of the similarity of their AC indicates that.

For the third approach, we want to explore if there are publicly available sources of information, other than the US or AC, that we can use to find indications for additional AC. Possible ideas are to utilize forums or platforms like stack overflow, but it is not yet clear, which information from these sources can be used for our goal and how they could be used.

We also need to consider if it is possible and helpful to combine the first approach with the other approaches. E.g. if a missing concept is identified it could be used to filter the results of the other approaches to find a fitting AC.

To find suitable algorithms for these approaches we will use the systematic mapping study. Therefore, the novelty of our solution is not necessarily in the algorithms we use but in the problem context that we apply them and the insights we get on the suitability of these algorithms in this context.

To be able to evaluate our approaches we acquired two datasets from industry that contain US and AC. The characteristics of the datasets are described in Table 1. As can be seen, only a fraction of the total requirements issues contained a US with AC that we could use. The datasets originally contained 308 issues for P1 and 2462 issues for P2. From those only 142 issues for P1 and 711 for P2 contained a US or AC. Of those, only 94 issues for P1 and 97 for P2 contained both AC and US. The US have 1-15 AC in P1 with an average of 4.03 AC. In P2 the US have 1-6 AC with an average of 2.21 AC. The final size of the datasets is the number of requirements after removing requirements of bad quality (e.g. when the US was divided by a bullet list or when the user's role mentioned in the US was another

software). For P1 this is 86 US and 358 AC, for P2 74 US and 157 AC. Both data sets contain requirements in German language that we automatically translated into English, which we checked and corrected if needed.

We are also looking for further data sets that are publicly available to be able to share our results with the research community. We did a first search in GitHub and found it difficult to identify projects which use US and AC in natural text, not in Gherkin notation, with the available search possibilities.

Table 1.
Characteristics of the data sets

Dataset	# original issues	# issues containing US and/or AC	# issues containing US and AC	#AC per US	Final #US, #AC
P1	308	142	94	1-15 (mean 4.03)	86 US and 358 AC
P2	2462	711	97	1-6 (mean 2.21)	74 US and 157 AC

4. Related Work

This section describes and discusses work related to our goal and to our solution ideas. There are approaches to generate AC automatically from requirements models via test models [2] or from controlled natural language [22] that is used to enhance the models to get more complete AC. However, those approaches use models which are not necessarily used in all projects. The second approach uses a notation for the requirements, which was developed for their approach. Both approaches generate AC in Gherkin language. This notation can be used for automatic tests, but is not very common in industry. We did not find any approaches without models which give ideas how to come up with a more complete set of AC.

We conducted first searches in literature to find algorithms for experiments with our solution ideas. In this first literature research, we found the following algorithms for concept extraction and comparison. This will be complemented by the mapping study. In [9] Ferrari et al. describe an approach to check the completeness of requirements by extracting and comparing the concepts of requirements and their input documents. We adapt this idea by extracting concepts from US and AC and comparing them. Robeer et al. [21] is an example to extract concepts specifically from US. As AC are usually more detailed than US and therefore of a lower abstraction level, Kof et al. [15] is interesting for us, who describe how concepts of different levels of abstraction can be compared by using WordNet [18].

In the first literature research, we found the following algorithms for similarity checks [1], [3], [6], [7], [14], [17], [19]. This will be complemented by the mapping study. The general idea is to transform the US into a format, e.g. vectors, that can be used to calculate the similarity between the US or the words of the US. For the calculation of the similarity a measure is needed. If the similarity was only calculated for the words of the US, it needs to be calculated for the whole US from these similarities. We looked in detail into the following two approaches.

Barbosa et al. [3] use and evaluate different similarity measures in order to find duplicate US. They evaluate the following measures: the Jaccard Similarity Index, the Vector Space Model in combination with Term Frequency – Inverse Document Frequency and Cosine Similarity, the WordNet [18] database together with the similarity measures designed for it, namely the WuPalmer similarity or Lin similarity and Lesk-A relationship. The similarity measures for WordNet measure only the similarity between pairs of words. To get the similarity on sentence level, they need to be aggregated in some way. Barbosa et al [3] do not describe how they did it.

Kochbati et al. [14] aim to group user stories of a complex software project according to similarity. In order to find similar user stories, first the word-level similarity is determined. They use word2vec, which is implemented as a pre-trained neural network and produces word embeddings. This means words are reflected by vectors. In the vector space, vectors representing similar meanings are close to each other. As a similarity measure of the word vectors, the cosine similarity is used. For the extension to requirement-level similarity, the Mihalcea scoring formula (presented by Mihalcea et al. [16]) is

applied. The inverse document frequency is also an ingredient of this formula. The scoring formula is applied to all user stories and the final output is an N x N similarity matrix, where N is the number of user stories.

There are different goals for the calculation of similarity, but, to the best of our knowledge, the goal to reuse AC has not been proposed yet.

5. Progress & Research Plan

Our research plan consists of the following steps

1. Conduct interviews with practitioners from industry
2. Analyze datasets with US and AC regarding quality criteria
3. Come up with solution ideas based on 1. and 2.
4. Conduct first experiments with algorithms for solution ideas
    a. Ground Truth creation
    b. Algorithm implementation
    c. Evaluation
5. Conduct systematic mapping study on NLP, ML and DL algorithms for solution idea
6. Choose algorithms from systematic mapping study to implement by defining and applying criteria
7. Implement chosen algorithms
8. Evaluate chosen algorithms.

We already did steps 1-4. In the first step we conducted interviews with 7 practitioners from industry. Those interviews were held online and lasted about 90 minutes each. In these interviews we asked questions on how they use and create AC, if there are any problems and if they have improvement ideas. The main results were that AC are a valuable tool to get a clear understanding of the software and that it is problematic if they are missing. One of the improvement ideas was to suggest the AC of similar US to be able to reuse them.

In the second step we analyzed the two datasets that we acquired from industry regarding quality criteria for US and AC. We found defects in all quality criteria, but as there are already tools that can find quality defects like understandability and consistency, we focused on the quality criteria completeness for further research. In the third step we came up with the solution ideas described in section 3.

Currently we are conducting experiments with algorithms for our solution ideas. First experiments are already done. For the first approach we used the concept extraction capability of Stanford NLP to extract the concepts similar to [9] and compared the concepts of the US with the concepts of the AC by utilizing WordNet [18] like in [15] as the concepts in the AC are often more detailed than in the US. For the second approach we implemented three different algorithms. The first algorithm uses the vector space model and cosine similarity like in [3]. The second algorithm uses WordNet with the Wu Palmer similarity on word level like in [3] which is extended to US level by using the Mihalcea scoring formula [16]. The third algorithm uses word2vec with cosine similarity on word level which is again extended to US level with the Mihalcea scoring formula [16] like in [14].

With our available datasets and our ground truth the algorithms for the second approach either have high precision with low recall or high recall with low precision, which is similar to some algorithms we found in literature. The algorithm for the first approach has low recall with medium to high precision, which is mostly due to the fact, that only few concepts could be extracted with the chosen algorithm. Therefore, we want to explore ways how those algorithms can be improved. On the one hand, we want to try other algorithms that we find in literature. On the other hand, we want to improve the combination of algorithms.

Our next experiment will be to utilize pre-trained models like USE [4] and BERT [12] to calculate the similarity between US. Additionally, we want to conduct the systematic mapping study in the next months to be able to refine our solution. To be able to choose fitting algorithms we already defined some sub questions to our research question RQ3:

- What are the prerequisites to use those algorithms?
- What are the steps of those algorithms? Are steps used in different algorithms?

- Are the steps from our already implemented algorithms used? How are they used?
- What are the goals of those algorithms?
- How are the algorithms evaluated? Which metrics are used? Which data set sizes are used?

After the systematic mapping study, we will refine our solution ideas and evaluate them with available data sets. It is planned to complete this until end 2024.

6. Acknowledgements

I thank my advisor Barbara Paech for her valuable feedback and support.

7. References

[1] M. Abbas, A. Ferrari, A. Shatnawi, E. Enoiu, M. Saadatmand, D. Sundmark: On the relationship between similar requirements and similar software: A case study in the railway domain. Requirements Engineering (2022). https://doi.org/10.1007/s00766-021-00370-4

[2] M. Alferez, F. Pastore, M. Sabetzadeh, L. C. Briand, J. R. Riccardi: Bridging the Gap between Requirements Modeling and Behavior-Driven Development. In: International Conference on Model Driven Engineering Languages and Systems, pp. 239–249, IEEE (2019). https://doi.org/10.1109/MODELS.2019.00008

[3] R. Barbosa, A. E. A. Silva, R. Moraes: Use of similarity measure to suggest the existence of duplicate user stories in the srum process. In: Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshop (DSN-W), pp. 2-5, Toulouse, France, IEEE (2016). https://doi.ieeecomputersociety.org/10.1109/DSN-W.2016.27

[4] D. Cer, Y. Yang, S. Kong, N. Hua, N. Limtiaco, R. S. John, N. Constant, M. Guajardo-Cespedes, S. Yuan, C. Tar, B. Strope, R. Kurzweil. Universal Sentence Encoder for English. In: Conference on Empirical Methods in Natural Language Processing: System Demonstrations, pp. 169–174, Brussels, Belgium. ACL (2018)

[5] M. Cohn: User stories applied: For agile software development. Addison-Wesley Educational, Boston, MA, USA (2004)

[6] F. Dalpiaz, I. van der Schalk, S. Brinkkemper, F. B. Aydemir, G. Lucassen: Detecting terminological ambiguity in user stories: Tool and experimentation. Information & Software Technology 110, 3–16 (2019). https://doi.org/10.1016/j.infsof.2018.12.007

[7] A. G. Duszkiewicz, J. G. Sørensen, N. Johansen, H. Edison, T. R. Silva: On identifying similar user stories to support agile estimation based on historical data. In: International Workshop on Agile Methods for Information Systems Engineering (Agil-ISE2022), pp. 21–26. CEUR-WS.org (2022).

[8] F. Fabbrini, M. Fusani, S. Gnesi, G. Lami: An Automatic Quality Evaluation for Natural Language Requirements. In: International Workshop on Requirements Engineering: Foundation for Software Quality, pp. 150–164 (2001).

[9] A. Ferrari, F. dell'Orletta, G. O. Spagnolo, S. Gnesi: Measuring and Improving the Completeness of Natural Language Requirements. In: Requirements Engineering: Foundation for Software Quality. REFSQ 2014. Lecture Notes in Computer Science, vol 8396. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-05843-6_3

[10] J. Fischbach, A. Vogelsang, D. Spies, A. Wehrle, M. Junker, D. Freudenstein: SPECMATE: Automated Creation of Test Cases from Acceptance Criteria. In: Int. Conf. on Software Testing, Validation and Verification, pp. 321–331, IEEE (2020). https://doi.org/10.1109/ICST46399.2020.00040

[11] R. Hoda, L. K. Murugesan: Multi-level agile project management challenges: A self-organizing team perspective, Journal of Systems and Software, Volume 117, pp. 245-257, (2016). https://doi.org/10.1016/j.jss.2016.02.049

[12] D. Jacob, C. Ming-Wei, L. Kenton, T. Kristina: BERT: Pre-training of deep bidirectional transformers for language understanding. In Conference of the North American Chapter of the

Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), pp. 4171–4186, ACL, Minneapolis, Minnesota (2019)
[13] B. A. Kitchenham, S. Charters: Guidelines for Performing Systematic Literature Reviews in Software Engineering (Version 2.3). Tech. rep. EBSE 2007-001. Keele, Staffs, UK; Durham, UK: Keele University and Durham University Joint Report, p. 65
[14] T. Kochbati, S. Li, S Gérard, C. Mraidha: From user stories to models: A machine learning empowered automation. In: International Conference on Model-Driven Engineering and Software Development, pp. 28-40. SCITEPRESS - Science and Technology Publications (2021). https://doi.org/10.5220/0010197800280040
[15] L. Kof, R. Gacitua, M. Rouncefield, P. Sawyer: Concept mapping as a means of requirements tracing. In: International Workshop on Managing Requirements Knowledge, pp. 22-31 (2010). https://doi.org/10.1109/MARK.2010.5623813
[16] R. Mihalcea, C. Corley, C. Strapparava: Corpus-Based and Knowledge-Based measures of text semantic similarity. In: National conference on Artificial intelligence (AAAI 06), pp. 775–780, AAAI Press, Boston (2006).
[17] F. A. Mihany, H. Moussa, A. Kamel, E. Ezzat, M. Ilyas: An automated system for measuring similarity between software requirements. In: Africa and Middle East Conference on Software Engineering - AMECSE '16, pp. 46-51, ACM Press, New York (2016). https://doi.org/10.1145/2944165.2944173
[18] G. A. Miller: WordNet: A lexical database for English. Communications of the ACM, 38 (11), 39–41 (1995). https://doi.org/10.1145/219717.219748
[19] G. Ninaus, F. Reinfrank, M. Stettinger, A. Felfernig: Content-based recommendation techniques for requirements engineering. In: International Workshop on Artificial Intelligence for Requirements Engineering (AIRE). IEEE (2014). https://doi.org/10.1109/AIRE.2014.6894853
[20] I. K. Raharjana, D. Siahaan, C. Fatichah: User Stories and Natural Language Processing: A Systematic Literature Review, IEEE Access, 9, pp. 53811–53826, 2021. https://doi.org/10.1109/ACCESS.2021.3070606
[21] M. Robeer, G. Lucassen, G., J. M. E. M. van der Werf, F. Dalpiaz, S. Brinkkemper: Automated Extraction of Conceptual Models from User Stories via NLP. In: International Requirements Engineering Conference (RE), pp. 196-205 (2016) https://doi.org/10.1109/RE.2016.40
[22] A. Veizaga, M. Alferez, D. Torre, M. Sabetzadeh, L. Briand, E. Pitskhelauri: Leveraging Natural-language Requirements for Deriving Better Acceptance Criteria from Models. In: International Conference on Model Driven Engineering Languages and Systems, pp. 218-228, ACM, New York (2020)
[23] K. Wiegers, J Beatty: Software Requirements, 3e, Microsoft Press, 2013, pp. 347-348.
[24] R. J. Wieringa: Design Science Methodology for Information Systems and Software Engineering. Springer Berlin Heidelberg, p. 332 (2014). https://doi.org/10.1007/978-3-662-43839-8
[25] L. Zhao, W. Alhoshan, A. Ferrari, K. J. Letsholo, M. A. Ajagbe, E. Chioasca, R. T. Batista-Navarro: Natural Language Processing for Requirements Engineering: A Systematic Mapping Study. ACM Comput. Surv. 54, 3, Article 55 (April 2022), 41 pages. https://doi.org/10.1145/3444689