What Quality Attributes Can We Find in Product Backlogs? A Machine Learning Perspective

Matthias Galster[1(✉)], Fabian Gilson[1], and François Georis[2]

[1] University of Canterbury, Christchurch, New Zealand
{matthias.galster,fabian.gilson}@canterbury.ac.nz
[2] University of Namur, Namur, Belgium
francois.georis@student.unamur.be

Abstract. Automatically identifying quality attributes (e.g., security, performance) in agile user stories could help architects reason about early architecture design decisions *before* analyzing a product backlog in detail (e.g., through a manual review of stories). For example, architects may already get the "bigger picture" of potential architectural key drivers and constraints. Applying a previously developed method to automatically identify quality attributes in user stories, in this paper we investigate (a) what quality attributes are potentially missed in an automatic analysis of a backlog, and (b) how the importance of quality attributes (based on the frequency of their occurrence in a backlog) differs to that of quality attributes identified in a manual review of a backlog. As in previous works, we analyzed the backlogs of 22 publicly available projects including 1,675 stories. For most backlogs, automatically identified quality attributes are a *subset* of quality attributes identified manually. On the other hand, the automatic identification would usually not find more (and therefore potentially irrelevant) quality attributes than a manual review. We also found that the ranking of quality attributes differs between the automatically and manually analyzed user stories, but the overall trend of rankings is consistent. Our findings indicate that automatically identifying quality attributes can reduce the effort of an initial backlog analysis, but still provide useful (even though high-level and therefore potentially incomplete) information about quality attributes.

Keywords: Agile software development · Quality attributes · Product backlog · User stories · Natural language processing

1 Introduction

A key principle of agile software development is to reduce potentially unnecessary upfront work. Nevertheless, it is important to understand the most significant architectural drivers early on to avoid architectural decisions that negatively impact modifiability or performance. If agile teams spend too little time thinking

T. Bures et al. (Eds.): ECSA 2019, LNCS 11681, pp. 88–96, 2019.
https://doi.org/10.1007/978-3-030-29983-5_6

about architecture design upfront, then there is an increased risk of failure [16]. Quality attributes such as performance, security or interoperability impact architecture design decisions, e.g., when selecting architectural patterns, tactics or reference architectures [1]. Therefore, identifying quality attributes early on is part of software architecture analysis [1]. Furthermore, in agile software development we need to balance near-term functional requirements and long-term quality goals [2]. Hence it is crucial to understand which quality attributes are relevant and which quality attributes might be more important than others. Prioritizing quality attributes is difficult in early development iterations and wrong decisions can result in hard-to-modify, unreliable, slow and insecure systems [8].

In agile software development, functional requirements are often specified as textual user stories. For example, for an online store one may define a story like "As a customer, I want to be able to create and edit a customer profile so that I can conveniently use all services of the e-shop." In our previous work [9] we showed that user stories do include information about quality attributes, and explored how to automatically identify user stories that include information about quality attributes. The goal was to better understand potential architectural key drivers and their "bigger picture" *before* analyzing a product backlog in detail (e.g., through a manual and potentially time-consuming review of the initial backlog).[1] As found by others, problems related to architecture are often found late in development projects [13]. Our previous work [9] also showed that we cannot rely on keywords when looking for quality attribute-related information in user stories. We therefore applied machine learning and natural language processing [9].

Machine learning and natural language processing are usually limited regarding precision and recall [7]. Therefore, in this paper we build on our previous work to investigate two exploratory questions: **Q1:** Does an automatic analysis of a backlog miss potentially relevant quality attributes? Answering this question could help understand whether automatic analysis of backlogs potentially misguides the architect's decision making process. **Q2:** How does the importance of quality attributes (based on the frequency of their occurrence in a backlog) differ between an automatic and a manual review of a backlog? Answering this question helps understand whether quality requirements can be reliably prioritized based on an automated analysis. We are interested in a more analytical and exploratory discussion of the implications of identifying quality attributes in user stories, rather than a detailed statistical and experimental evaluation as partially done in our previous work [9]. Thus, in this paper we present a challenge in software architecture research and promising results.

[1] We acknowledge that quality attributes are not the only factors with architectural significance; however, other factors are outside the scope of this work.

2 Related Work

In software architecture, there are already examples of using natural language processing, e.g., to extract design decisions from issue management systems [3, 14] or to identify architectural knowledge in developer communities [15].

In 2019, Binkhonain and Zhao conducted a review on machine-learning techniques that classify non-functional requirements [4]. However, most techniques use comprehensive requirement documents (rather than short user stories) written by requirement engineers and rely on keywords. On the other hand, we focus on user stories that are usually written by end users with less training in requirements engineering and writing requirements. Two techniques identified in the review [6,12] deal with mobile app reviews which share some commonalities with user stories (e.g., short and concise sentences), but for a different context. While most of the works discussed in [4] recover design decisions or architecture knowledge *post-hoc* or for reuse, our goal is to inform decisions of architects early on based on architectural drivers that arise from user requirements. Furthermore, our work is also related to prioritizing quality attributes (see e.g., Koziolek [11]). We aim at a lightweight yet useful analysis of architecture-relevant quality attributes in agile development.

3 Research Approach

Below we briefly discuss our approach to explore the two questions outlined in Sect. 1. The **corpus of backlogs and user stories** in our study was a set of 1,775 publicly available stories similar to that used by Dalpiaz *et al.* [5] and in our previous work [9], see Table 1. The number of user stories ranges from 50 to 114 with an average of 76 per backlog. The average length of stories is 24 words.

To **manually identify quality attributes**, two researchers independently labelled user stories to indicate up to two quality attributes per user story (part of our previous work). Then, we merged the labelling and discussed disagreements. We used quality attributes as described in the ISO/IEC 25010 standard [10] (for more details see our previous work [9]). For example, the story "As a repository administrator, I would like to be able to continue to provide access to the repository in the event that the server fails." was labelled as referring to *reliability* since it mentions continuous access to a system even in case of failures. In the following, we use abbreviations for quality attributes (C: compatibility, M: maintainability, PF: performance, PT: portability, R: reliability, S: security).

To **automatically identify quality attributes**, we relied on our previous work [9] which compared different natural language-based machine learning techniques and models using the spaCy library for natural language processing.[2] In this paper we used the best performing model to identify quality attributes (this model had an average precision of *0.65*, average recall of *0.67* and average f_1 score of *0.66* in a k-fold 10 validation) trained on manually labeled stories for all quality attributes.

[2] https://spacy.io/.

Table 1. Corpus of product backlogs and user stories.

Backlog	Description	Stories
FederalSpending	Web platform for sharing US government spending data	94
Loudoun	Land management system for Loudoun County, Virginia	57
Recycling	Online platform to support waste recycling	50
OpenSpending	Website to increase transparency of government expenses	53
FrictionLess	Platform for obtaining insights from data	66
ScrumAlliance	First version of the Scrum Alliance website	97
NSF	New version of the NSF website	72
CamperPlus	App for camp administrators and parents	53
PlanningPoker	First version of the PlanningPoker.com website	52
DataHub	Platform to find, share and publish data online	67
MIS	Management information system for Duke University	83
CASK	Toolbox to for fast and easy development with Hadoop	63
NeuroHub	Research data management portal	102
Alfred	Personal interactive assistant for active aging	134
BadCamp	Conference registration and management platform	69
RDA-DMP	Software for machine-actionable data management plans	82
ArchiveSpace	Web-based archiving system	55
UniBath	Institutional data repository for the University of Bath	53
DuraSpace	Repository for different types of digital content	99
RacDam	Software for archivists	100
CulRepo	Content management system for Cornell University	114
Zooniverse	Platform that allows anyone to help with research tasks	60

For each **user story** in each backlog (and following the manual and automatic identification procedures from above) we recorded whether or not it addresses a quality attribute and if so which one(s). For each **backlog**, we collected a ranked list (or sequence) of quality attributes based on (a) the absolute number of occurrences of a quality attribute in all stories of a backlog, (b) the relative occurrence of a quality attribute compared to the number of user stories in a backlog, and (c) the relative occurrence of a quality attribute based on all stories that reference a quality attribute over all backlogs. The rankings were the same using any of these three metrics. We collected this information for manually and automatically identified quality attributes separately. We do not consider other priorities of user stories (e.g., based on value): Priorities are often not known upfront as user stories are usually prioritized by different stakeholders during initial iteration planning and sometimes even re-prioritized later.

To **compare the sequences** of manually and automatically identified quality attributes, we used a simple metric based on the pairwise swaps required to transform one sequence into the other. In case two ranked sequences did not include the same number of quality attributes, the shorter sequence was filled up

with empty strings. Then, when transforming one sequence into the other, this empty string in one sequence was moved to the position of the missing quality attribute in the other sequence. For example, the sequence $s_m = \{PF, C, R\}$ from the manual identification and $s_a = \{PF, R\}$ from the automatic identification would lead to a comparison of sequences $s_m = \{PF, C, R\}$ and $s_a = \{PF, R, \epsilon\}$ (where ϵ denotes the empty string). We would require one swap between C and R in s_a to move C to the position of ϵ in s_m. The total number of swaps required in the example would then be 1. The larger the number of swaps, the more different the sequences.

4 Results

In Table 2 we provide the sequences of ranked quality attributes for each backlog (most frequently to least frequently occurring attribute). "Missed" indicates how many quality attributes appear in the sequence from the manual identification, but not in the sequence from the automatic identification. "Additional" indicates the number of quality attributes that appear in the sequence from the automatic identification, but not in the sequence from the manual identification.

Key Findings Regarding Q1 (Missing Quality Attributes): Table 2 (column "Difference") shows that for most backlogs, the automatic classification identified a *subset* of the manually labelled quality attributes. For only two backlogs, the automatic identification found quality attributes that were not identified through manual inspection (security for backlogs of *PlanningPoker* and *NSF*). This means that the amount of false positives on backlog level is rather small (we analyzed false positives at story level in [9]). On the other hand, the most frequently missed quality attributes across all backlogs were security, reliability and portability (eight times each). There are two backlogs for which no quality attribute were automatically identified. *CamperPlus* contained two stories related to security and *BadCamp* contained two stories related to security and one related to compatibility. Still, these attributes were indirectly related to the stories. For example, the story "As a parent, I want to be able to create an account, so that I can sign up my kids for camp online." from *CamperPlus* was annotated with security albeit no obvious reference to security, the manual annotations often being subject to human interpretation.

Key Findings Regarding Q2 (Importance of Quality Attributes): We found that the ranked sequences were quite different mostly because of the missing quality attributes in the automatic classification (column "Difference"). The number of swaps is rather small except for a few backlogs, e.g., *NSF* and *CASK* (see column "Swaps"). On the other hand, the sequences for *NeuroHub* (the only backlog where the quality attributes were the same in both rankings) showed quite a different order. Focusing on the top quality attributes, the differences are rather small (e.g., a quality attribute might be the first ranked in one sequence and the second ranked in another sequence). An exception is the backlog for *CASK*, where compatibility appears least frequently in the manual sequence,

Table 2. Sequences of ranked quality attributes.

Backlog	Sequences	Difference	Swaps
FederalSpending (manual) FederalSpending (automatic)	$\{M, C, PF, S\}$ $\{C\}$	Missed: 3 Additional: 0	1
Loudoun (manual) Londoun (automatic)	$\{C, S\}$ $\{C\}$	Missed: 1 Additional: 0	0
Recycling (manual) Recycling (automatic)	$\{C, S, M, PT\}$ $\{S, C\}$	Missed: 2 Additional: 0	1
OpenSpending (manual) OpenSpending (automatic)	$\{C, M, S, PT\}$ $\{C, S\}$	Missed: 2 Additional: 0	1
FrictionLess (manual) FrictionLess (automatic)	$\{C, PF, R, M\}$ $\{C, PF, S, M\}$	Missed: 1 Additional: 1	1
ScrumAlliance (manual) ScrumAlliance (automatic)	$\{S, C\}$ $\{S, C\}$	Missed: 0 Additional: 0	0
NSF (manual) NSF (automatic)	$\{C, M, PT\}$ $\{M, S, C\}$	Missed: 1 Additional: 1	3
CamperPlus (manual) CamperPlus (automatic)	$\{S\}$ None	Missed: 1 Additional: 0	0
PlanningPoker (manual) PlanningPoker (automatic)	$\{C, PF, S\}$ $\{S, C\}$	Missed: 1 Additional: 0	2
DataHub (manual) DataHub (automatic)	$\{C, R, PT, S, M\}$ $\{C\}$	Missed: 4 Additional: 0	0
MIS (manual) MIS (automatic)	$\{S, C, M, PT, R\}$ $\{C, S\}$	Missed: 3 Additional: 0	1
CASK (manual) CASK (automatic)	$\{M, R, PT, C\}$ $\{C, M\}$	Missed: 2 Additional: 0	3
NeuroHub (manual) NeuroHub (automatic)	$\{C, S, PT, M, R, PF\}$ $\{C, S, M, PT, PF, R\}$	Missed: 0 Additional: 0	2
Alfred (manual) Alfred (automatic)	$\{C, S, M, PT, PF\}$ $\{C, S, M, PT\}$	Missed: 1 Additional: 0	0
BadCamp (manual) BadCamp (automatically)	$\{S, C\}$ None	Missing: 2 Additional: 0	0
RDA-DMP (manual) RDA-DMP (automatic)	$\{S, PF, C, R\}$ $\{S, C\}$	Missed: 2 Additional: 0	1
ArchiveSpace (manual) ArchiveSpace (automatic)	$\{C, S, R, M\}$ $\{C, S\}$	Missed: 2 Additional: 0	0
UniBath (manual) UniBath (automatic)	$\{C, S, R, M, PF\}$ $\{C, S\}$	Missed: 3 Additional: 0	0
DuraSpace (manual) DuraSpace (automatic)	$\{S\}$ $\{S\}$	Missed: 0 Additional: 0	0
RacDam (manual) RacDam (automatic)	$\{S, C\}$ $\{S, C\}$	Missed: 0 Additional: 0	0
CulRepo (manual) CulRepo (automatic)	$\{C, S, R, PF, PT\}$ $\{C, S\}$	Missed: 3 Additional: 0	0
Zooniverse (manual) Zooniverse (automatic)	$\{C, S\}$ $\{C\}$	Missed: 1 Additional: 0	0

but most frequently in the automatic sequence. When looking at the number of occurrences of quality attributes in each sequence of *CASK*, we notice that the absolute numbers for compatibility are rather close, but the main difference is related to maintainability.

5 Discussion

Implications: Given our preliminary key findings above, we believe that even though automatically identifying quality attributes may not result in exactly the same quality attributes identified by a human analyst, the automatic approach still provides insights for an initial design space exploration. Considering the time required to manually review a backlog (magnitude of hours) compared to the time of conducting the automatic approach (magnitude of seconds or minutes), we believe that an automated backlog analysis could *complement* rather than *replace* human decision making during architecture design: the automated backlog analysis provides a starting point for problem and solution space exploration (e.g., the automated analysis could identify key architectural drivers).

Limitations: One limitation of our work is that we do not differentiate runtime quality attributes and design time quality attributes. Differentiating types of quality attributes would allow a more detailed analysis of the implications and perhaps the importance of quality attributes. Furthermore, we do not consider the business value of user stories when determining the ranking of quality attributes. Quality attributes that appear in more "valuable" user stories may receive a higher priority (in addition to considering how often these quality attributes appear in a backlog). Also, we do not consider changing and growing backlogs. Our assumption is that quality attributes can be analyzed continuously, but it is important to understand the "bigger picture" early on.

Treats to Validity: In terms of *external validity*, this research relies on a limited number of user stories and backlogs. It is unclear whether these backlogs are representative of industrial practices in general. However, since how user stories are specified in practice varies (e.g., phrasing patterns and writing guidelines) and does not always follow best practices, it may be hard to identify a truly representative set of stories. Also, we only consider stories but no acceptance criteria. Finally, our set of quality attributes is rather limited as it follows the structure of the ISO/IEC 25010 quality model. Future work includes considering more hierarchy levels of that quality model. On the other hand, this will require a much larger set of user stories to train a machine learning classifier, since the number of quality attributes in ISO/IEC 25010 is rather large. Regarding *internal validity*, there could be confounding variables which impact our results and in particular the manual labeling of stories, e.g., the labeling did not involve initial stakeholders. Regarding *conclusion validity*, when comparing quality attributes identified manually and automatically, we treated manually identified attributes as "ground truth". Thus, our findings depend on the quality of the manual classification and consistency across stories.

6 Conclusions

In this paper we presented insights about how automatically identified quality attributes in user stories can provide information for architectural decision making. We found that (a) even though the automatic classification does not identify all quality attributes considered relevant by human experts, at least it identifies a subset rather than a random list of quality attributes, and (b) the rankings of quality attributes identified manually and automatically vary, but trends in sequences are consistent. Future works include analyzing more backlogs and user stories and investigating the impact of distinguishing types of quality attributes on the identified quality attributes and their rankings.

References

1. Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice. Addison-Wesley, Boston (2012)
2. Bellomo, S., Gorton, I., Kazman, R.: Toward agile architecture: Insights from 15 years of ATAM. IEEE Softw. **32**(5), 38–45 (2015)
3. Bhat, M., Shumaiev, K., Biesdorf, A., Hohenstein, U., Matthes, F.: Automatic extraction of design decisions from issue management systems: a machine learning based approach. In: Lopes, A., de Lemos, R. (eds.) ECSA 2017. LNCS, vol. 10475, pp. 138–154. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-65831-5_10
4. Binkhonain, M., Zhao, L.: A review of machine learning algorithms for identification and classification of non-functional requirements. Expert Syst. Appl.: X **1**, 1–13 (2019)
5. Dalpiaz, F., van der Schalk, I., Brinkkemper, S., Aydemir, F.B., Lucassen, G.: Detecting terminological ambiguity in user stories: tool and experimentation. Inf. Softw. Technol. **110**, 3–16 (2019)
6. Deocadez, R., Harrison, R., Rodriguez, D.: Automatically classifying requirements from app stores: a preliminary study. In: Fourth International Workshop on Artificial Intelligence for Requirements Engineering (AIRE). IEEE (2017)
7. Domingos, P.: A few useful things to know about machine learning. Commun. ACM **55**(10), 78–87 (2012)
8. Galster, M., Angelov, S., Martínez-Fernández, S., Tofan, D.: Reference architectures in scrum: friends or foes? In: Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE), pp. 896–901. ACM (2017)
9. Gilson, F., Galster, M., Georis, F.: Extracting quality attributes from user stories for early architecture decision making. In: International Workshop on Decision Making in Software Architecture (MARCH), pp. 1–8. IEEE (2019)
10. ISO/IEC: ISO/IEC 25010 system and software quality models. Technical report, International Organization for Standardization/International Electrotechnical Commission (2010)
11. Koziolek, A.: Architecture-driven quality requirements prioritization. In: IEEE International Workshop on the Twin Peaks of Requirements and Architecture (TwinPeaks), pp. 1–5. IEEE (2012)

12. Lu, M., Liang, P.: Automatic classification of non-functional requirements from augmented app user reviews. In: International Conference on Evaluation and Assessment in Software Engineering (EASE), pp. 344–353. ACM (2017)
13. Martensson, T., Martini, A., Stahl, D., Bosch, J.: Continuous architecture: towards the goldilocks zone and away from vicious circles. In: International Conference on Software Architecture (ICSA), pp. 131–140. IEEE (2019)
14. Shahbazian, A., Lee, Y.K., Le, D., Brun, Y., Medvidovic, N.: Recovering architectural design decisions. In: International Conference on Software Architecture (ICSA), pp. 95–104. IEEE (2018)
15. Soliman, M., Galster, M., Riebisch, M.: Developing an ontology for architecture knowledge from developer communities. In: International Conference on Software Architecture (ICSA), pp. 89–92. IEEE (2017)
16. Waterman, M., Noble, J., Allan, G.: How much up-front? A grounded theory of agile architecture. In: International Conference on Software Engineering (ICSE), pp. 347–357. IEEE (2017)