Requirements Information in Backlog Items: Content Analysis

Ashley T. van Can [0009−0001−1190−8327] and
Fabiano Dalpiaz [0000−0003−4480−3887]

Utrecht University, Heidelberglaan 8, 3584 CS Utrecht, The Netherlands
{a.t.vancan, f.dalpiaz}@uu.nl

Abstract. [**Context and motivation**] With the advent of agile development, requirements are increasingly stored and managed within issue tracking systems (ITSs). These systems provide a single point of access to the product and sprint backlogs, bugs, ideas, and also tasks for the development team to complete. [**Question/problem**] ITSs combine two perspectives: representing requirements knowledge and allocating work items to team members. We tackle a *knowledge problem*, addressing questions such as: How are requirements formulated in ITSs? Which types of requirements are represented? At which granularity level? We also explore whether a distinction exists between open source projects and proprietary ones. [**Principal ideas/results**] Through quantitative content analysis, we analyze 1,636 product backlog items sampled from fourteen projects. Among the main findings, we learned that the labeling of backlog items is largely inconsistent, and that user-oriented functional requirements are the prevalent category. We also find that a single backlog item can contain multiple requirements with different levels of granularity. [**Contribution**] We reveal knowledge and patterns about requirements documentation in ITSs. These outcomes can be used to gain a better empirical understanding of Agile RE, and as a basis for the development of automated tools that identify and analyze requirements in product and sprint backlogs.

Keywords: Agile Requirements Engineering · User Stories · Backlog Items · Issue Tracking Systems · Content Analysis.

1 Introduction

Agile software development emerged as a means for many software companies to stay competitive by improving market responsiveness, gaining the ability to continuously and quickly define and re-prioritize software requirements based on the ever-changing stakeholder needs [11].

In agile software development, requirements are defined in an iterative and incremental fashion. Documenting and discussing requirements in agile practices is commonly carried out by writing and discussing user stories [4], which are key elements of the product and sprint backlogs [9].

While researchers have explored and reviewed the benefits and challenges of agile RE [11] and the nature of the product backlog [23], there is limited research that analyzes how requirements are documented in real-world product and sprint backlogs, which are often stored in an issue tracking system like JIRA.

In this paper, we conduct a preliminary analysis of 1,636 issues from fourteen JIRA repositories (seven open source, seven proprietary), aiming to address our main research question **MRQ**: *How are requirements documented within backlog items?*. Since we focus on requirements that are stored in product and sprint backlogs, we examine only those JIRA repositories that we know are used or that are likely to serve as a representation of these backlogs[1]. Methodologically, we employ content analysis [14, 27] as the lens through which we analyze a sample of the issues in these repositories, leading to a bottom-up understanding that is rooted in empirical data.

Studies that adopt a similar research method to analyze data from product or sprint backlogs include the investigation of architectural knowledge in JIRA issues [25], the analysis of emotions in backlog items [20], and linking JIRA issues to software life-cycle activities [18].

Given the informal nature of issue tracking systems as documentation tools, in this paper we use the term *requirement* to denote a variety of textual fragments, without limiting ourselves to the use of specific templates such as the 'shall' format, user stories, or the like. This is in line with the CPRE Glossary by IREB [7], which states that a requirement is "A documented representation of a need, capability or property".

This paper makes two contributions to the state-of-the-art:

- Through content analysis of a large sample of backlog items from fourteen projects, we present first insights on how requirements are documented in agile RE via issue tracking systems;
- As a byproduct of the analysis, we share a coding scheme that can be used for the analysis of additional datasets.

The rest of the paper is structured as follows. Sect. 2 refines the MRQ into five research questions. Sect. 3 presents our research method, including the sampled dataset and the coding scheme. Sect. 4 shows our results. Sect. 5 contrasts our work with related work. Sec. 6 discusses the results in terms of the research questions. Finally, Sect. 7 draws conclusions and sketches future directions.

2 Research Questions

In order to address the main research question stated in the introduction, we put forward five more specific research questions: RQ1 through RQ5.

RQ1. To what extent do the backlog item labels chosen by practitioners reflect the requirements expressed in the items?

[1] We do not make a distinction as to whether the items belong to the product backlog or to the sprint backlog; in the remainder of this paper, we therefore use the term 'backlog items' to refer to the components of either backlog.

RQ1 focuses on whether the practitioners use suitable issue labels (e.g., 'Story' or 'Epic' in JIRA) to distinguish requirements from issues that represent other aspects, such as bugs or tasks. In particular, we define one hypothesis that emerges when separating issues with labels about requirements from issues with labels that concern the execution of tasks:

H1. Backlog items with requirement-related labels contain requirements more often than backlog items with task-related labels.

In addition to analyzing backlog items based on the appropriateness of the labels, we dig deeper into the understanding of the requirements categories that are represented. A full explanation of the categories we employ is presented later in the paper (see Table 1); for now, an example is the classic distinction between functional and non-functional requirements [3]. This leads to RQ2:

RQ2. What categories of requirements information are more commonly used?

One of the key properties of requirements, when stored in a requirements management system, is for them to be uniquely identifiable [10]. We want to study whether this is the case also with issue tracking systems, or if multiple requirements co-occur in the same backlog item (in issue tracking systems, backlog items are the smallest identifiable piece of information). This entails RQ3:

RQ3. How often does a single backlog item include multiple requirements?

We can further refine RQ3 based on the question regarding categories (RQ2), leading to studying whether – when multiple requirements appear in the same backlog item – certain combinations of different categories are prevalent:

RQ3.1. What different requirements categories do co-occur more often in a backlog item?

Given the importance of justifying the *why* in requirements engineering [28], we put forward RQ4 that explores whether the requirements in backlog items also include information concerning why they are needed.

RQ4. To what extent are requirements complemented by a motivation for their existence?

Finally, we aim to conduct a preliminary analysis on whether open source software (OSS) projects, which are easier to retrieve and access, are representative of how requirements are represented in proprietary projects, which are subject to confidentiality rules. To this extent, we introduce RQ5:

RQ5. Can we identify differences w.r.t. RQ1–RQ4 when comparing proprietary projects and open-source projects?

3 Research Method

We apply content analysis to gain insight into the requirements information present in issue tracking systems. Content analysis is a "research technique for making replicable and valid inferences from texts (or other meaningful matter) to the contexts of their use" [14]. In our case, in line with the distinction by White and Marsh [27], we make use of quantitative content analysis: after defining our research questions and hypotheses when suitable, we collect data, determine the collection unit (issues organized into projects), define the coding scheme, tag the data, analyze, and write up the results.

One important deviation from the classic approach is that, while we started from literature knowledge to define basic codes (we performed *inductive coding*), we have used a subset of the data to refine the coding scheme by introducing specific sub-categories (*deductive coding*). Our process is visualized in Fig. 1.

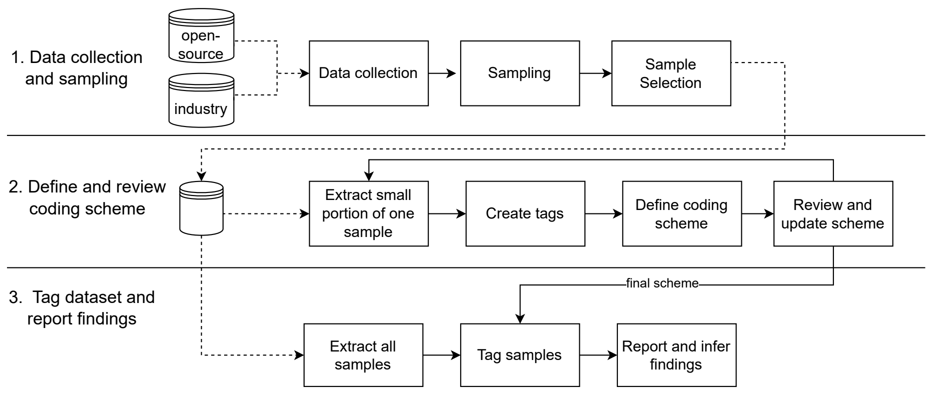
1. Data collection and sampling
open-source
industry
Data collection
Sampling
Sample Selection
2. Define and review coding scheme
Extract small portion of one sample
Create tags
Define coding scheme
Review and update scheme
3. Tag dataset and report findings
final scheme
Extract all samples
Tag samples
Report and infer findings

Fig. 1: The research method describing how we 1) collected and sampled the data, 2) defined and reviewed the coding scheme and 3) tagged the data.

3.1 Data collection and sampling

The topmost row of Fig. 1 illustrates how we collected data and we sampled the backlog items to analyze.

Data collection. We analyze data extracted from issue tracking systems of several projects, aiming to identify backlog items. We also collected seven internal development projects from a collaborating company: the low-code development platform provider Mendix. The other projects were sampled from two large public datasets of issue tracking systems for OSS projects [26, 18].

From a quick scan, we could determine that many of the issue tracking systems for the open-source projects are employed mainly for reporting bugs rather than for documenting requirements. We automatically excluded projects with $\geq$80% issues labeled as 'bug' and $<$10% labeled as 'story'. This screening allowed us to filter out the projects whose dataset is likely not to serve as sprint or product backlog, retaining 88 projects.

In addition, since user stories (arguably the main requirements artifact in Agile RE [4, 13, 16]) are user-oriented requirements, we decided to focus on projects with a clear user interaction component, eliminating projects without a clear user interface. The first author assessed the 88 retained projects based on their UI component and excluded irrelevant projects.

This led to 16 candidate open-source projects. Since we knew that the JIRA issues from all seven industry projects were used to represent the sprint backlog, we did not exclude any industry projects.

Sampling. Since the projects varied in size (from 57 to 5,750 issues), we selected a sample from each project. We defined an initial sample size of 100 items, as we found that a sample selection based on a time period would result in a large variation in sample sizes (as each project varies greatly in how often items are uploaded). We aimed to select a sample that represents well the backlog items that the development teams worked upon in a given time period. Therefore, we randomly selected one issue and included the 99 subsequent issues (not counting bugs) in order of creation time, excluding the last 100 issues from the random selection phase. In other words, considering the dataset as a set of sliding windows of size 100 (and sliding interval of 1), we randomly selected one sliding window per project.

Links between issues provide additional context regarding how various backlog items relate and depend on one another [17]. Thus, we decided to include this information when extracting samples of the projects. We considered each issue as part of a *cluster* of linked issues. After selecting a sample S of 100 issues, we added all issues that were directly and transitively linked to a particular issue in S, excluding those created later than the last issue in S.

Sample selection. Before the different OSS projects[2] could be utilized further, the two authors of this paper independently evaluated the first 12 issues of each sample in order to verify if the initial selection contained a sufficiently high volume of relevant requirements information and were not merely exploited for bug reports and task lists. Each of the 12 issues per project was classified as to whether it contained information related to requirements. After an independent tagging, the authors reviewed the classification together and subsequently discussed any differences. Finally, the projects with over 50% (of the 12 issues) marked as requirements relevant were shortlisted (details in our online appendix[3]). This resulted in 7 open-source projects and 7 private projects.

[2] Thanks to our collaboration with Mendix, we knew those projects were using the issue tracking system to represent the backlog items the teams would implement in the various sprints.

[3] Online appendix: https://doi.org/10.5281/zenodo.10643450

3.2 Coding scheme construction

The coding scheme was constructed through the analysis of two of the 14 project samples, one OSS (QT Design Studio) and one proprietary (Portfolio). We defined the coding scheme iteratively and performed the tagging using the software Nvivo. In each iteration, one tagger examined a small additional section of data to identify a variety of information in the backlog items, focusing specifically on content and writing patterns. After creating the codes, the first tagger grouped related codes or adjusted codes to construct a coding scheme. When the first tagger was unsettled about certain scenarios, the first and second taggers discussed the situation, adjusting the scheme accordingly. The rest of the process was repeated each time adding new data to adjust the scheme until the scheme no longer required adjustment.

After the two projects were all tagged based on the scheme, the second tagger checked the scheme and the tagged dataset, resulting in some minor final adjustments, leading to the coding scheme that is available in our online appendix and that is summarized in Sect. 3.3.

The first tagger applied the scheme to 50% of the remaining projects. The second tagger independently tagged a random 20% of each project. We subsequently compared the tagged items and discussed any conflicts. We consider tagging difference as a conflict when a specific text is tagged with a different granularity level or type. This resulted in a percent agreement rate of 65%, after which we made a few minor adjustments to 6 projects.

Next, the first tagger completed the remainder of the projects, of which the second tagger independently tagged 20%. After comparing the tags, we agreed on 71% of the items, which is an improvement over the first 6 projects. Based on these minor mismatches, we adjusted the dataset.

3.3 Coding scheme

The coding scheme (Table 1) distinguishes two characteristics on which a requirement can be classified: a) the requirement *type* and b) the *granularity* level. The requirement type indicates whether it is a functional or non-functional requirement. For functional requirements, we have defined two possible subcategories: a) user-oriented (indicating that the user directly experiences the added functionality, and b) system-oriented (the added functionality is not directly experienced by the user but is necessary for the system to function as desired). The granularity level denotes the level of refinement of the requirement, where we distinguish between low-level (e.g., acceptance criteria), medium-level (e.g., user stories), and high-level requirements (e.g., epics). Recognizing the importance of the reason for the requirement, we also tag whether there exists a *motivation* for the requirements in that backlog item. The complete tagging guidelines are available in our online appendix.

Table 1: Overview of our coding scheme.

Characteristic	Category	Description
Requirement type	User-oriented functional	Functionality directly experienced by the user.
	System-oriented functional	Functionality that the system will implement but that is not directly experienced by the user.
	Non-functional	Requirement that constrains or sets some quality attributes upon functional requirements [5].
Granularity level	Low	Requirement that is directly verifiable (e.g., acceptance criterion).
	Medium	Requirement that refers to one specific functional or non-functional aspect of the system (e.g., user story).
	High	Requirement that encompasses multiple aspects or functionalities of the systems (e.g., epic or theme).

Table 2: The projects selected in this study, showing the total size, sample size, number of items with requirements labels (*E*: Epic, *F*: Feature, *US*: User Story, *SU*: Suggestion), number of task-labelled issues (*T*: Task, *ST*: Sub-task, *TT*: Technical task, *ST*: Support ticket), and other issues.

Project	Size		Req-labeled				Task-labeled				Other
	Total	Sample	E	F	US	SU	T	ST	TT	ST	
Control	738	120	15	0	90	0	0	14	0	1	0
Service	173	100	6	0	57	0	37	0	0	0	0
Store	634	109	12	0	69	0	0	0	0	21	7
Company	29	29	0	0	29	0	0	0	0	0	0
Portfolio	97	97	4	0	84	0	0	9	0	0	0
Data	57	57	8	0	27	0	20	2	0	0	0
Learn	994	143	15	0	116	0	0	5	0	7	0
Cost Management	2,038	179	15	8	99	0	28	29	0	0	0
Jira Performance Testing Tools	777	105	2	0	26	57	12	8	0	0	0
Lyrasis Dura Cloud	1,125	113	0	0	105	0	7	0	0	0	1
Network Observability	137	102	2	0	99	0	1	0	0	0	0
OpenShift UX Product Design	369	130	3	0	113	0	3	11	0	0	0
Qt Design Studio	4,983	180	5	0	51	6	11	21	86	0	0
Red Hat Developer Website	5,750	172	21	0	151	0	0	0	0	0	0
Total	17,901	1,636	108	8	1,116	63	119	99	86	29	8

3.4 Selected projects

Table 2 shows the different projects included in this study, indicating the number of items in the original dataset, the sample size and the labels used. Each backlog item consists of a label, which is specified by one of the team members and should reflect the content of the item. In addition to the label, we examine the description, summary (i.e., title) and cluster to which each item belongs.

4 Results

We present the findings of our content analysis for RQ1–RQ5. We address RQ1–RQ4 in four sections, each of which ends with a reflection on RQ5: we split the projects population between proprietary and OSS projects in order to assess if differences exist. Given the small sample size for RQ5 (7 projects per group), we do not run statistical significance tests as their reliability would be low. Due to space limitations, the raw data are available in our online appendix.

4.1 Do practitioners choose accurate backlog item types (RQ1)?

RQ1 examines whether practitioners' labeling accurately reflects the content of backlog items and to identify any inconsistencies. We expect backlog items of types 'Epic', 'Feature', 'Story' and 'Suggestion' to contain requirements information, while items with types 'Task', 'Technical task', 'Sub-task' and 'Support-tickets' not to. We exclude 8 issues with a rarely occurring type that are hard to relate to requirements or tasks. Our hypothesis H1 is that the the first kind of backlog items will more frequently contain requirements-related tags.

Table 3 shows, for each project, the ratio of items that (i) are labeled as tasks and include at least one requirement, and that (ii) are labeled as requirements and include at least one requirement. The columns on the left focus on proprietary projects, the ones on the right on OSS projects.

Table 3: Ratio of items task-labeled and requirements-labeled items that include at least one requirement according to our tagging.

Project (proprietary)	Task	Req	Project (OSS)	Task	Req
Control	0.2	0.62	Cost Management	0.28	0.58
Service	0.14	0.92	Jira Performance Testing Tools	0.15	0.64
Store	0	0.49	Lyrasis Dura Cloud	0.29	0.75
Company	0	0.66	Network Observability	0	0.42
Portfolio	0.22	0.51	OpenShift UX Product Design	0.79	0.58
Data	0.73	1	Qt Design Studio	0.30	0.52
Learn	0.25	0.45	Red Hat Developer Website	0	0.44
Macro-average	0.22	0.66		0.26	0.56
Std- dev	0.25	0.22		0.27	0.11
Macro-average (all)	0.24	0.61			
Std-dev (all)	0.25	0.17			

To confirm H1 statistically, given the limited sample size of n=14, we choose a robust non-parametric test: Mann-Whitney's U, verifying whether the ratio of requirements in the requirements-labeled items is greater than that in the task-labeled items (H1), or alternatively if they can be considered equal (H0). The Mann-Whitney U test results in a test statistic of 173 with a p-value of 0.0001.

At a significance level of $\alpha > 0.05$, we can reject the claim H0 that the two rates are equal. The effect size is *large*, as $d_{cohen} = 1.716$.

When comparing proprietary and OSS projects, task-labeled items exhibit similar results: for proprietary projects we obtain an average $\overline{x} = 0.22$ ($\sigma = 0.25$), and for OSS projects an average $\overline{x} = 0.26$ ($\sigma = 0.27$). Similarly, when comparing the items labeled as requirements, the proprietary projects yield an average $\overline{x} = 0.66$ ($\sigma = 0.22$) while the OSS projects result in $\overline{x} = 0.56$ ($\sigma = 0.11$). The results indicate a slight difference, with the proprietary projects having on average more items correctly labeled as requirements in comparison to OSS projects, although more investigations are necessary to draw robust conclusions.

4.2 What are the most commonly used categories (RQ2)?

RQ2 aims to reveal what type of requirements are frequently present in backlog items and with what degree of granularity they are formulated. Fig. 2 shows, for each combination of type and granularity (see Table 1), the occurrence across the 14 projects, distinguishing between proprietary from OSS projects.

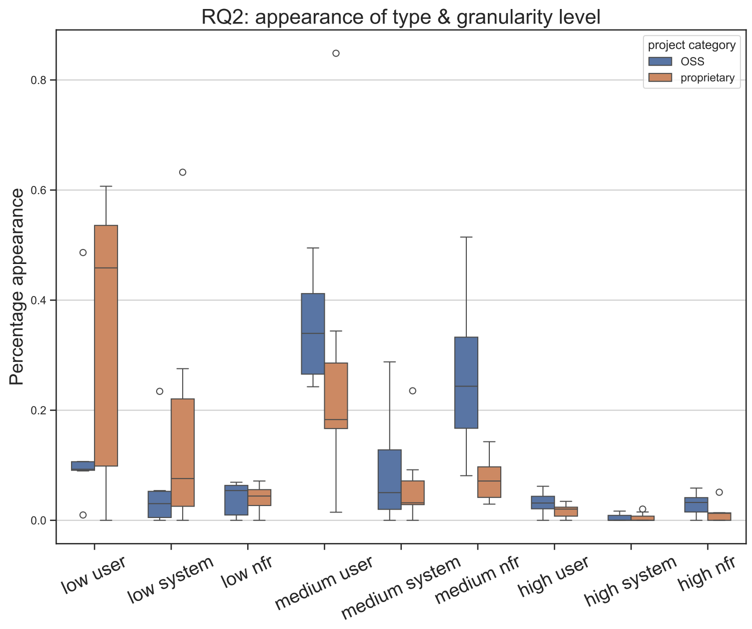

Fig. 2: Percentage of occurrence of the type-granularity combinations.

When examining the medians per combination, medium-level user-oriented requirements appear to be most prevalent in all projects with an overall median of 0.266 ($\overline{x} = 0.313$, $\sigma = 0.199$). The second more frequent ones are low-level user-oriented requirements with a median of 0.106 ($\overline{x} = 0.237$, $\sigma = 0.228$). As can be seen by the relatively high standard deviation (compared to the $\overline{x}$) and the gap between the median and $\overline{x}$, low-level user-oriented requirements also exhibit the largest fluctuations across projects.

Fig. 2 shows that certain combinations exhibit a considerable difference between the OSS and proprietary projects. In particular, for low-level user-oriented requirements, the large variation for proprietary projects contrasts with a nearly nonexistent one for OSS projects. The results also show the higher percentage of medium-level requirements for OSS projects compared to proprietary projects. For high-level requirements and low-level non-functional requirements, only a slight variation exists between project types.

4.3 Do backlog items include multiple requirements (RQ3)?

Since requirements are expected to be uniquely identifiable [10], RQ3 examines whether backlog items comply with this property. We explore how many of the backlog items with requirement-related information have more than one requirement (RQ3). Then, we examine which combinations of tags (type and granularity level) are most prevalent when 2+ requirements per item are identified (RQ3.1).

Table 4 shows the percentage of requirements-related backlog items consisting of multiple requirements. The results show that the projects in our sample does not only comprise requirements that are uniquely identifiable. Nonetheless, the standard deviation indicates large per-project variations. For example, among the items containing requirements in Jira Performance Testing Tools, only 14% contain items with multiple requirements, while the project Company has almost 95% items containing multiple requirements.

Table 4: Presence of multiple requirements in a single issue.

Project (proprietary)	Multiple (%)	Project (OSS)	Multiple (%)
Control	0.632	Cost Management	0.540
Service	0.048	Jira Performance Testing Tools	0.140
Store	0.550	Lyrasis Dura Cloud	0.235
Company	0.947	Network Observability	0.286
Portfolio	0.745	OpenShift UX Product Design	0.538
Data	0.549	Qt Design Studio	0.164
Learn	0.419	Red Hat Developer Website	0.434
Macro-average (propr)	0.556	Macro-average (OSS)	0.334
Std-dev (propr)	0.280	Std-dev (OSS)	0.170
Macro-average (all)	0.445	Std-dev (all)	0.251

Table 5 shows the most common combinations of different tags that co-occur in an issue (RQ3.1) having at least 10 total occurrences. The most frequent combination is having a medium requirement to be refined into one or more low-level requirements of the same type. In addition to its frequency, this combination appears in 12 of our 14 projects. The second most common combination is two medium-level requirements: one non-functional and one functional user-oriented: this occurs in 13 projects. The third row is complementary to the first one and it shows that in several projects, medium-level system-oriented functional requirements are refined into low-level requirements of the same type.

Table 5: Most frequent combinations of different tags in the same issue, showing both the total and the per-project counts.

Combinations	Total	Control	Service	Store	Company	Portfolio	Data	Learn	Cost mgmt	JIRA Perf	Lyrasis	Network Obs	OpenShift	QT Design	RH Developer
low user, medium user	96	16	0	1	1	17	17	17	2	1	0	6	1	2	15
medium nfr, medium user	61	2	3	3	0	0	2	0	6	2	15	1	23	2	2
low system, medium system	34	3	0	4	12	6	0	0	8	0	0	0	0	0	1
low user, medium user, low system	16	7	0	2	0	3	0	3	0	0	0	0	0	0	1
low user, low nfr, medium user	13	4	0	1	0	1	2	1	1	0	0	0	1	0	2
low user, medium nfr	13	2	0	2	0	3	3	0	0	0	0	0	1	1	1
low nfr, medium nfr	11	1	0	2	0	2	0	0	0	0	0	0	4	0	2
low user, medium nfr, medium user	10	0	0	0	0	0	4	0	2	0	0	0	1	1	2

Table 4 also compares the two types of projects (RQ5) in terms of the presence of multiple requirements. On average, the proprietary projects hold more items with multiple requirements ($\overline{x} = 0.556$, $\sigma = 0.280$) than the OSS projects ($\overline{x} = 0.334$, $\sigma = 0.170$). The standard deviations likewise show larger fluctuations in these percentages among proprietary projects compared to OSS projects.

4.4 Are requirements complemented by a motivation (RQ4)?

RQ4 examines whether a backlog item containing requirements includes an associated motivation. Table 6 shows the percentage of backlog items where we identified at least one requirement that contain at least one justification. The overall macro-average shows that the motivation behind many requirements is not present in the backlog. The standard deviation of 0.169 also indicates that the percentages fluctuate only slightly across the projects. In ten of the fourteen projects, less than 50% of the requirements-containing items have an associated justification. The other four projects have less than 60% of their backlog items with motivations.

Table 6 compares the type of projects (RQ5) in terms of the presence of motivation. The macro average reveals only a small difference, with OSS projects having slightly more frequent motivations for their backlog items.

Table 6: Backlog items with at least one requirement that also have a motivation.

Project (proprietary)	Yes (%)	Project (OSS)	Yes (%)
Control	0.088	Cost Management	0.540
Service	0.524	Jira Performance Testing Tools	0.649
Store	0.525	Lyrasis Dura Cloud	0.383
Company	0.368	Network Observability	0.238
Data	0.529	OpenShift UX Product Design	0.487
Portfolio	0.319	Qt Design Studio	0.269
Learn	0.194	Red Hat Developer Website	0.171
Macro-average (propr)	0.364	Macro-average (OSS)	0.391
Std-dev (propr)	0.176	Std-dev (OSS)	0.175
Macro-average (all)	0.377	Std-dev (all)	0.169

4.5 Threats to validity

We discuss threats to validity, ranging from internal to external factors, and discuss how we mitigated these to preserve the credibility of the study.

Regarding project selection, only one tagger eliminated irrelevant projects. Since this evaluation was conducted by a single tagger, it is possible that some interesting projects in the OSS datasets were excluded. Moreover, we did not tag all issues in the projects, but only a subset. We reduce this vulnerability by randomly selecting a subset of a representative size.

In addition, a single tagger created the initial coding scheme. To eliminate bias, the second tagger reviewed all the data on which the initial coding scheme was built. The feedback from the second tagger was used to adjust the coding scheme. Additionally, only one tagger coded the full sets of remaining projects for the final tagging of the projects. We mitigated these biases by including a second tagger who randomly tagged 20% of the issues in the sample.

For some open source projects, we could not ascertain whether the analyzed issues are part of a sprint or product backlog, due to the absence of such details in the datasets. We employed filtering mechanisms to only retain projects whose issue tracking systems are likely to be used to support sprint backlogs or to serve as a product backlog; however, we cannot be certain.

Furthermore, we have excluded issues labeled as “bug”. Although it is possible that bug issues exist that contain requirements information, based on an initial exploration phase, we have determined that this situation is improbable.

For the proprietary projects, we examine projects from one company. Selecting only projects from one specific source could lead to a number of threats. We reduced this vulnerability by using projects from different teams and including a wide variety of OSS projects from different companies, but we reckon that future work needs to use data from multiple companies.

5 Related work

Lüders and colleagues conducted research on the visualization and automated categorization [17] of links between issues. Our approaches are complementary. While they are concerned with the relationships (including dependencies and hierarchies) between requirements and other issue types, we offer an in-depth analysis of the *contents* of the issues.

Rath *et al.* [22] explored the effectiveness of automated traceability by assessing the ability of machine learning in recovering trace links between code commits and JIRA issues. Similar studies have been conducted, in the context of model-driven engineering, by van Oosten *et al.* [19]. Although these studies also analyzed issues in JIRA repositories, their focus is on repairing trace links, while we examine the requirements information in the issues.

Interview studies have been often employed to learn about the practices of documenting requirements in agile development. For example, Behutiye *et al.* [2] conducted fifteen interviews with practitioners from four companies using agile software development and they studied how quality requirements are documented. Their findings showed that in certain cases issues and epics are used to represent quality requirements, but also that prototypes and face-to-face communication are very important. A similar analysis was conducted by Alsaqaf *et al.* [1] in the context of large-scale, distributed settings. Their exploratory study reveals fifteen challenges, several of which are related to the minimal documentation principle in agile development. The study by Franch and colleagues [6] on requirements specification shows that, in agile contexts, project management tools are commonly used to document requirements. These studies are based on interviewing practitioners, while we focus on obtaining insights through the content analysis of backlog items.

Some research groups have collected and shared collections of issues extracted from publicly available repositories. The two largest and renowned datasets are the TAWOS dataset by Tawosi *et al.* [26] and the 'alternative' one by Montgomery and colleagues [18]. We make use of six projects from Montgomery's dataset and one from TAWOS, and we contrast these with seven proprietary projects.

Some studies applied content analysis to backlog items. For instance, Soliman *et al.* [25] investigated where architectural knowledge is located in JIRA issues, Ortu and colleagues [20] have studied the emotions that are included in the issues, and Montgomery [18] linked issues to software life-cycle activities. We conduct a more in-depth analysis of requirements within backlog items.

Content analysis has been used extensively in software engineering for the analysis of communication within instant messaging systems as well as chat rooms. For example, Parra *et al.* [21] compare the contents present in Slack and Gitter in terms of Bin's categorization [15]: do the messages fulfill a developer's personal needs, team-wide purposes, or community support? Silva *et al.* [24] conducted thematic on a large number of Slack and Gitter chatrooms to identify what developers talk about. In this paper, we also apply content analysis but we focus on backlog items rather than messaging systems.

6 Discussion

We address each research question on the basis of the findings reported in Sect. 4. While doing so, we highlight remarkable results and provide additional likely explanations for certain phenomena.

RQ1: To what extent do the backlog item labels chosen by practitioners reflect the requirements expressed in the items? The hypothesis H1 formulated for this research question tested whether the percentage of requirements in requirements-labeled items is higher than in task-related items. The test results reported in Sect. 4.1 confirm the hypothesis, indicating it is more likely to find requirements in items labeled as such than in items labeled as tasks. This result provides empirical evidence in support of a straightforward conjecture. The average percentages shown in Table 3 are, instead, more surprising. On average, over 20% of the items labeled as tasks do in fact contain requirements. In addition, on average, more than 30% of the items labeled as requirements contain no requirements at all. These results show that practitioners do inconsistently label the items; therefore, in order to locate requirements within backlog items, it is not sufficient to simply display the issues that are labeled as requirement (here: epics, features, user stories, and suggestions).

RQ2: What categories of requirements information are more commonly used? Fig. 2 visualizes the occurrence of different types of requirements with specific granularity levels in backlog items. The figure shows a high fluctuation between certain categories, especially for low and medium-level user-oriented functional requirements and medium-level non-functional requirements. This reflects the different usage patterns of teams managing their product or sprint backlogs. For example, the combination low-level functional requirements ('low user' in Fig. 2) for proprietary projects shows high variability; this happens because some teams include acceptance criteria in the same issue where a user story is written, while others do not specify them, or store them in a different environment.

RQ3: How often does a single backlog item include multiple requirements? For each project, Table 4 shows how many of the items containing at least one requirement also contain multiple requirements. In general, the results show that many of these backlog items contain multiple requirements, making them no longer uniquely identifiable. In addition, Table 5 indicates which different categories of requirements often occur together, revealing that low-level requirements most often occur in combination with medium-level requirements. This is a generalization of the refinement pattern where a user story is refined into acceptance criteria. In addition, the results show that non-functional requirements often co-occur with functional requirements. This could be interpreted in terms of the refinement of non-functional requirements into functional requirements that are closer to system design [8]. More frequently and surprisingly, however, we found functional requirements that also specified a non-functional aspect of the system. An example from the Cost Management project is 'As a user, I want to quickly filter my tags based on tag key or value, [. . .]', which points to the functionality of a filtering option and also to the quality of performing it quickly.

RQ4: To what extent are requirements complemented by a motivation for their existence? We found that more than half of the backlog items that contain at least one requirement did not include any justifications for those requirements. These results show the lack of recognition of the importance [28] and recommendations [10] for expressing the 'why' behind requirements. This may be due to agile software development practices, where requirements are formulated in a concise manner, as they are intended to support and foster the conversation within the team [12], rather than acting as a precise and complete specification. In addition, this table considers the presence of justifications for all categories of requirements, while justifications are not equally essential for all categories, particularly when we look at low-level requirements. Nevertheless, the project Service, for example, contains no low-level requirements and still only includes justifications for roughly 40% of the required items.

RQ5: Can we identify differences w.r.t. RQ1–RQ4 when comparing proprietary projects and open-source projects? For each of the research questions above, we distinguish the open source projects from the proprietary projects. One of the most notable discrepancies between the open source projects and the proprietary dataset is the difference in labeling the backlog items, with the open source projects showing more inconsistencies (see Table 3). This lack of consistency in open source projects may be due to the low level of oversight or the varying experience level of contributors. In contrast, industrial projects in our sample show large variations between teams in their structuring of the project backlog (e.g., what type of requirements they include), and more often document multiple requirements within a single backlog item (Table 4).

7 Conclusion and future work

We performed content analysis on collections of backlog items to better understand the occurrence of requirements-related information in backlogs. For this purpose, we collected, tagged, and analyzed fourteen samples of open-source and proprietary projects, summing up to a total of 1,636 items.

Our results show that backlog item labeling is applied inconsistently and in a misleading manner by practitioners. In addition, teams may use one backlog item to document multiple requirements, making requirements within backlogs not uniquely identifiable. Both aspects pose challenges for those who need to retrieve requirements information. Furthermore, the most common are medium- and low-level user-oriented requirements, which also occur together in one item and mirror the refinement pattern of user stories in acceptance criteria.

In general, we find that item labels chosen by practitioners do not fully represent the content of requirements, especially when a backlog item contains multiple requirements, possibly of different types. Our most immediate future work aims to build a prototype tool to help practitioners automatically extract and classify requirements from collections of backlogs items.

Although backlog items contain a significant amount of requirements, they may not represent all the requirements. Especially in agile development, scenarios may arise where developers discover the need for a new feature during system development without specifying the implementation in the backlogs. Therefore, future work could focus on examining the completeness of sprint and product backlogs as a requirements specification artifact, or whether other documents (e.g., user journeys and vision documents, as indicated by our industrial partner Mendix) should be considered to obtain a fuller picture.

Acknowledgements This research is partially funded by the Dutch Research Council (NWO) through the Open Technology Programme 2021-II TTW, project AUTOLINK (19521). We would like to thank Mendix, and especially Toine Hurkmans, for providing us with the proprietary datasets used in this study.

References

1. Alsaqaf, W., Daneva, M., Wieringa, R.: Quality requirements challenges in the context of large-scale distributed agile: An empirical study. Information and Software Technology **110**, 39–55 (2019)
2. Behutiye, W., Seppänen, P., Rodríguez, P., Oivo, M.: Documentation of quality requirements in agile software development. In: Proc. of EASE. pp. 250–259 (2020)
3. Cleland-Huang, J., Settimi, R., Zou, X., Solc, P.: Automated classification of non-functional requirements. Requirements engineering **12**, 103–120 (2007)
4. Cohn, M.: User stories applied: For agile software development. Addison-Wesley Professional (2004)
5. Cysneiros, L.M., do Prado Leite, J.C.S., de Melo Sabat Neto, J.: A framework for integrating non-functional requirements into conceptual models. Requirements Engineering **6**, 97–115 (2001)
6. Franch, X., Palomares, C., Quer, C., Chatzipetrou, P., Gorschek, T.: The state-of-practice in requirements specification: an extended interview study at 12 companies. Requirements Engineering pp. 1–33 (2023)
7. Glinz, M.: A glossary of requirements engineering terminology. Standard Glossary of the Certified Professional for Requirements Engineering (CPRE) Studies and Exam, Version **2.0.1** (2022)
8. Gross, D., Yu, E.: From non-functional requirements to design through patterns. Requirements Engineering **6**, 18–36 (2001)
9. Hess, A., Diebold, P., Seyff, N.: Understanding information needs of agile teams to improve requirements communication. Journal of Industrial Information Integration **14**, 3–15 (2019)
10. IEEE: Systems and software engineering – life cycle processes –requirements engineering. ISO/IEC/IEEE 29148:2018(E) (2018)
11. Inayat, I., Salim, S.S., Marczak, S., Daneva, M., Shamshirband, S.: A systematic literature review on agile requirements engineering practices and challenges. Computers in Human Behavior **51**, 915–929 (2015)
12. Jeffries, R.E., Anderson, A., Hendrickson, C.: Extreme Programming Installed. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (2000)

13. Kassab, M.: The changing landscape of requirements engineering practices over the past decade. In: Proc. of EmpiRE. pp. 1–8. IEEE (2015)
14. Krippendorff, K.: Content analysis: An introduction to its methodology. Sage publications (2018)
15. Lin, B., Zagalsky, A., Storey, M.A., Serebrenik, A.: Why developers are slacking off: Understanding how software teams use slack. In: Proc. of CSCW companion. p. 333–336 (2016)
16. Lucassen, G., Dalpiaz, F., van der Werf, J.M., Brinkkemper, S.: The use and effectiveness of user stories in practice. In: Proc. of REFSQ. LNCS, vol. 9619, pp. 205–222 (2016)
17. Lüders, C.M., Pietz, T., Maalej, W.: On understanding and predicting issue links. Requirements Engineering pp. 1–25 (2023)
18. Montgomery, L., Lüders, C., Maalej, W.: An alternative issue tracking dataset of public jira repositories. In: Proc. of MSR. pp. 73–77 (2022)
19. van Oosten, W., Rasiman, R., Dalpiaz, F., Hurkmans, T.: On the effectiveness of automated tracing from model changes to project issues. Information and Software Technology **160**, 107226 (2023)
20. Ortu, M., Murgia, A., Destefanis, G., Tourani, P., Tonelli, R., Marchesi, M., Adams, B.: The emotional side of software developers in JIRA. In: Proc. of MSR. pp. 480–483 (2016)
21. Parra, E., Alahmadi, M., Ellis, A., Haiduc, S.: A comparative study and analysis of developer communications on Slack and Gitter. Empirical Software Engineering **27**(2), 1–33 (2022)
22. Rath, M., Rendall, J., Guo, J.L., Cleland-Huang, J., Mäder, P.: Traceability in the wild: automatically augmenting incomplete trace links. In: Proc. of ICSE. pp. 834–845 (2018)
23. Sedano, T., Ralph, P., Péraire, C.: The product backlog. In: Proc. of ICSE. pp. 200–211. IEEE (2019)
24. Silva, C.C., Galster, M., Gilson, F.: A qualitative analysis of themes in instant messaging communication of software developers. Journal of Systems and Software **192**, 1–15 (2022)
25. Soliman, M., Galster, M., Avgeriou, P.: An exploratory study on architectural knowledge in issue tracking systems. In: Proc. of ECSA. pp. 117–133. Springer (2021)
26. Tawosi, V., Al-Subaihin, A., Moussa, R., Sarro, F.: A versatile dataset of agile open source software projects. In: Proc. of MSR. pp. 707–711 (2022)
27. White, M.D., Marsh, E.E.: Content analysis: A flexible methodology. Library trends **55**(1), 22–45 (2006)
28. Yu, E.S., Mylopoulos, J.: Understanding "why" in software process modelling, analysis, and design. In: Proc. of ICSE. pp. 159–168. IEEE (1994)