Refinement of User Stories into Backlog Items: Linguistic Structure and Action Verbs

Research preview

Laurens Müter[1], Tejaswini Deoskar[2], Max Mathijssen[1], Sjaak Brinkkemper[1], and Fabiano Dalpiaz[1]

[1]RE-Lab, Dept. of Information and Computing Sciences, Utrecht University
{L.H.F.Muter, M.Mathijssen, S.Brinkkemper, F.Dalpiaz}@uu.nl
[2]Utrecht Institute of Linguistics, Department of Languages, Literature, and Communication, Utrecht University
T.Deoskar@uu.nl

Abstract. [**Context and motivation**] In agile system development methods, product backlog items (or tasks) play a prominent role in the refinement process of software requirements. Tasks are typically defined manually to operationalize how to implement a user story; tasks formulation often exhibits low quality, perhaps due to the tedious nature of decomposing user stories into tasks. [**Question/Problem**] We investigate the process through which user stories are refined into tasks. [**Principal ideas/results**] We study a large collection of backlog items (N=1,593), expressed as user stories and sprint tasks, looking for linguistic patterns that characterize the required feature of the user story requirement. Through a linguistic analysis of sentence structures and action verbs (the main verb in the sentence that indicates the task), we discover patterns of labeling refinements, and explore new ways for refinement process improvement. [**Contribution**] By identifying a set of 7 elementary action verbs and a template for task labels, we make first steps towards comprehending the refinement of user stories to backlog items.

Keywords: Requirements engineering, user stories, backlog items, natural language processing, sprint tasks.

1 Introduction

User stories (USs) have made their way into the development process of companies [1] and their adoption is evolving to higher levels [2,1]. USs are the starting point for specifying software that is developed, according to the agile development paradigm, through a series of sprints. The USs are distributed to the development teams that refine the USs into a number of (usually 3 to 6) so-called backlog items (but also called tasks) to break down a US into specific executable tasks for developers to carry out during the sprints.

Software specifications have been thoroughly studied from the viewpoint of their linguistic structure. Researchers have proposed approaches for finding ambiguity [3,4] and other types of defects [5] in natural language requirements, for generating conceptual models [6,7], and much more [8].

Previous work has conducted linguistic analyses of USs and defined guidelines for writing a *good* specification in agile development [1,9]. The template structure of a US "As a [Role] I want to [Action], so that [Benefit]" is often misused and many real-world USs are poorly written requirements [10]. However, there is no study on the requirements-related artifacts that stem from USs in agile development and Scrum, i.e., backlog items or *tasks*.

Table 1: Example US that has been refined into 3 tasks

US: As a webshop visitor I want to add shipping addresses so that I can send presents to my friends	
Task-1	Create ShippingAddresses records for visitors
Task-2	Update validity check for Addresses
Task-3	Add data-item for LastShippingAddress to visitor

Table 1 shows the refinement of a US into three tasks. By reading the table, one can see that tasks are the bridge between user-centered requirements (USs) and development artifacts like code and test cases. It is not surprising that the tasks are the basic constituents of sprint backlogs, i.e., they define what functionality will be included in the next release of the product.

The contribution of this paper is a linguistic analysis of a large industrial product backlog that includes 195 USs and 1,593 tasks. We study the linguistic structure of the task labels as well as the main verb that indicates what actions the developers are expected to carry out. Based on the analysis, we distill guidelines for writing tasks in a clear and consistent way.

After describing our research approach in Sec. 2, we present our linguistic analysis of the sentence structure (Sec. 3) and of the main verb in a task (Sec. 4). Finally, we present conclusions and outline future directions.

2 Research Approach

We considered a large product backlog provided to us by a multinational software development company, located in the Netherlands, and having circa fifty employees. The company's main product is a web-based platform to manage contract and tender processes of companies in the procurement industry.

The initial data consisted of 2,702 backlog items, each labeled as Epic, Feature, Task, or Bug. In this paper, we focus on the tasks (1,593, 59.04%). Each backlog item has an attribute that defines the development status in the product development: New (6.49%), To Do (3.74%), Approved (1.41%), Committed (1.33%), In Progress (1.26%), Done (85.29%), Removed (0.48%).

Our linguistic analysis started with running the Stanford Part-of-Speech (POS) tagger to determine the structure of the task labels; for example, "Define (VB) box (NN) type (NN) actions (NNS) and (CC) implement (VB) them

(PRP). (.)"[1] indicates that "define" is a verb, "box" is a singular noun, "actions" is a plural noun, "and" is a conjunction, and so on.

We experienced that the POS tagger accuracy was not perfect, presumably because task labels are hardly written as grammatically correct sentences. Two major problems we encountered were words that can be tagged as either verbs or nouns (e.g., "update") and spelling mistakes (e.g., "crate" instead of "create").

We then looked at the first-occurring verb in each task label, trying to identify recurring patterns. After tagging the unique verbs, we employed classes of VerbNet to cluster the identified verbs in families of related verbs.

Finally, we extracted regular expression patterns that fit most of the tasks and that can be used as a recommended template for task label writers.

3 Linguistic structure of task labels

The goal of this analysis is to identify the most common linguistic structures in the sentences that represent tasks labels. Because of the vast number of existing POS tags, we grouped the tags as shown in Table 2. For example, verbs tagged with different tenses (present/past) are grouped into the *verb* category.

Table 2: Grouping of POS tags employed in analysis

Group Tag	POS Tags	Occurrence	%	Unique first Words
verb	VB, VBD, VBG, VBP, VBZ	1,173	73.63	70
noun	NN, NNS, NNP, NNPS	322	20.21	65
adjective	JJ, JJR, JJS	27	1.69	13
adverb	RB, RBR, RBS	27	1.69	4
pronoun	PRP, PRP$	7	0.44	2
other		37	2.32	11
total		1,593	100	165

Despite the grouping, the Stanford POS tagger identified 968 different linguistic structures that represent the 1,593 tasks, thereby showing the various ways task labels are formulated by developers.

POS taggers are trained with long newswire text and not with short, sketched sentences like task labels, so to further improve the accuracy we performed a manual amendment of some tags (especially verb instead of noun). The ten most frequent structures are shown in Table 3. In the table, we use the following abbreviations: NN = noun, VB = verb, IN = conjunction, and JJ = adjective. The most frequent pattern is a verb followed by two nouns, for example: "Create tender-settings component" (VB, NN, NN). Several variations exist that add an adjective or a conjunction to the sequence of nouns. In the top-10 list, only two structures start with a noun, which usually indicates the architectural location of the task. Task labels starting with a noun will be analyzed in future work.

Given the variations in sentence structures as presented in Table 3, we distill a template that we propose as a guideline for writing task labels. The extended

[1] The individual tags refer to the Penn Treebank tagset [11].

Table 3: The ten most frequent structures of task labels

Structure	Freq.	%	Example
VB, NN(S), NN	130	8.17	Create tender-settings component
VB, NN(S), NN, NN(S)	67	4.18	Create Messages DB tables
NN, NN(S), NN(S)	25	1.57	Admin licenses breadcrumbs
VB, NN(S), IN, NN	21	1.32	Add filters for KO
VB, NN, NN(S), NN(S), NN	20	1.26	Implement TenderPlan actions business logic
VB, JJ, NN(S), NN	18	1.13	Create disqualified offers card
VB, NN	27	1.67	Create TenderProcessDefinitionLevelRule
VB, NN(S), IN, NN, NN	15	0.94	Bind rules per section item
VB, NN, NN, IN, NN, NN(S)	13	0.82	Create SQL Script for AcceptedById items
NN, NN(S)	10	0.62	Update actions

Baccus-Naur form (EBNF) grammar for the template (shown below) states that a task is expressed by a verb, followed by one or more `follow` elements, each being either a noun, a conjunction, an adjective, a "to", or a cardinal number.

```
task = verb, follow, {follow};
follow = noun | conjunction | adjective | "to" | cardinal number;
```

The pattern matches 42.4% of the tasks in the dataset (676 out of 1,593). Further research will reveal more detailed patterns in the label set in order to develop guidelines for task refinement.

4 On the choice of an action verb

Task labels describe an *action* for the developer to carry out in order to implement part of a software function, or to improve existing code. We have first analyzed the first *action verb* that occurs in a task label. To do so, we employed the Stanford POS tagger and extracted the action verbs from our 1,593 task labels. This resulted in 56 different verbs, which became 81 after some manual pre-processing of spelling errors and noun-verb conversion. The 20 most frequently occurring action verbs are shown in Table 4.

The most frequent action verb is *create*, which amounts to about one third of the entire task set. This figure is a strong indicator of the feature creep phenomenon [12]. On the other hand, a very related verb such as *delete* occurs only in 1.5%. However, while analyzing the results, we observed that quasi-synonyms exist; for instance, the *remove* verb is a synonym of *delete*.

The observed relatedness of some verbs and the quasi-synonyms motivate to obtain a smaller set of action verbs for use in task descriptions. We resorted to VerbNet [13], a taxonomy of verbs that groups similar verbs in so-called *verb classes*. For example, the class *create* (CREATE-26.4) includes alternative terms,

Table 4: Most frequent action verbs that occur in a task label

Rank	Action verb	Frequency	Rank	Action verb	Frequency
1	Create	578	11	Bind	11
2	Modify	125	12	Update	11
3	Add	85	13	Move	10
4	Implement	79	14	Show	10
5	Change	27	15	Delete	9
6	Extend	19	16	Get	9
7	Set	18	17	Redesign	9
8	Check	16	18	Setup	8
9	Load	14	19	Fix	8
10	Remove	13	20	Review	8

besides the namesake verb, the similar verbs *coin*, *fabricate*, *construct*, etc. We identified verb classes in VerbNet that could act as containers for multiple verbs; moreover, we performed some adjustments to cope with the domain-specific jargon of software development. The analysis of our dataset resulted in the seven families of action verbs listed in Table 5.

Table 5: Families of action verbs in task labels

Family	Members of the verb-family
Create	code, create, define, design, implement, insert, make
Update	add, adjust, change, edit, extend, fix, improve, insert, renew, replace, refactor, redesign
Merge	bind, export, insert, integrate, invite, link, list, offer
Delete	delete, remove
Validate	check, evaluate, research, test, verify
Control	accept, allow, apply, bind, cancel, check, configure, control, determine
Investigate	inquire, investigate, research, search

Our analysis of the data set leads us to distill the following recommendations regarding the use of elementary action verbs in task labels:

- Each task should start with an action verb.
- The family-verb defines the nature of the development action to be performed with the code.
- The starting action verb should be in the imperative mood.
- When a suitable member-verb exists in Table 5, that verb should be used.

When re-analyzing our data set using our guidelines, we found many well formed task labels but also several poorly defined lables. A poorly defined task would be “Box breadcrumb component”, which could be rewritten as “Create box breadcrumb component”. On the other hand, “Update validity check for Addresses” from Table 1 is a well defined task, for “update” is listed in Table 5.

Table 6: Elementary action verbs for task labeling

Verb	Explanation	Example
Create	add new features	Create new tender property.
Update	change existing functionality	Update all permissions screens.
Merge	combine existing functionalities	Integrate localization in datetime picker.
Delete	remove existing functionalities	Delete offer stored procedure.
Validate	test existing functionalities	Evaluate inserted event.
Control	manage existing functionality	Control of access to box content.
Investigate	study potential functionality	Research angular 2.0 validation and refactoring components.

5 Conclusions and directions

Our linguistic analysis of a large industrial product backlog resulted in preliminary guidelines for writing backlog items / tasks in a consistent manner, which also offers possibilities for the development of tools that assist analysts in the authoring of high-quality task descriptions.

Tasks play a key role in agile development, for they bridge the problem space (the requirements) and the solution space (the architecture and the code). The tasks refine the product requirements expressed as USs. A poorly formulated task is likely to lead to issues in the developed code and sprint velocity.

This research-in-progress paper simply paves the way for future work in the field. First and foremost, we have used a single dataset in our analysis. The guidelines are likely to need some amplification, and their impact on software development needs to be evaluated *in vivo*. In the long run, we hope this research will bring insights and theories to the "wild" world of agile development.

References

1. Lucassen, G., Dalpiaz, F., van der Werf, J.M.E.M., Brinkkemper, S.: The Use and Effectiveness of User Stories in Practice. In: Proc. of REFSQ. (2016) 205–222
2. Kassab, M.: The Changing Landscape of Requirements Engineering Practices over the Past Decade. In: Proc. of EmpiRE. (2015) 1–8
3. Berry, D.M., Kamsties, E., Krieger, M.M.: From contract drafting to software specification: Linguistic sources of ambiguity. Technical report, School of Computer Science, University of Waterloo, Canada (2001)
4. Bano, M.: Addressing the challenges of requirements ambiguity: A review of empirical literature. In: Proc. of EmpiRE. (2015) 21–24
5. Rosadini, B., Ferrari, A., Gori, G., Fantechi, A., Gnesi, S., Trotta, I., Bacherini, S.: Using NLP to detect requirements defects: An industrial experience in the railway domain. In: Proc. of REFSQ. (2017) 344–360
6. Lucassen, G., Robeer, M., Dalpiaz, F., van der Werf, J.M.E.M., Brinkkemper, S.: Extracting conceptual models from user stories with visual narrator. Requirements Engineering **22**(3) (2017) 339–358

7. Yue, T., Briand, L.C., Labiche, Y.: A systematic review of transformation approaches between user requirements and analysis models. Requirements Engineering **16**(2) (Jun 2011) 75–99
8. Bakar, N.H., Kasirun, Z.M., Salleh, N.: Feature extraction approaches from natural language requirements for reuse in software product lines: A systematic literature review. Journal of Systems and Software **106** (2015) 132–149
9. Wautelet, Y., Heng, S., Kolp, M., Mirbel, I.: Unifying and Extending User Story Models. In: Proc. of CAiSE. Volume 8484 of LNCS. (2014) 211–225
10. Lucassen, G., Dalpiaz, F., van der Werf, J.M.E.M., Brinkkemper, S.: Improving agile requirements: The Quality User Story framework and tool. Requirements Engineering **21**(3) (2016) 383–403
11. Marcus, M.P., Marcinkiewicz, M.A., Santorini, B.: Building a Large Annotated Corpus of English: The Penn Treebank. Computational Linguistics **19**(2) (1993) 313–330
12. Jones, C.: Strategies for managing requirements creep. Computer **29**(6) (1996) 92–94
13. Schuler, K.K.: Verbnet: A Broad-coverage, Comprehensive Verb Lexicon. PhD thesis, Philadelphia, PA, USA (2005) AAI3179808.