Requirements Communication in Issue Tracking Systems in Four Open-Source Projects

Thorsten Merten[1], Bastian Mager[1], Paul Hübner[2], Thomas Quirchmayr[2], Barbara Paech[2], and Simone Bürsner[1]

[1] Bonn-Rhein-Sieg University of Applied Sciences, Dept. of Computer Science, Sankt Augustin, Germany
{thorsten.merten,simone.buersner}@h-brs.de, bastian.mager.2010w@informatik.h-brs.de
[2] University of Heidelberg, Institute of Computer Science, Heidelberg, Germany
{huebner,quirchmayr,paech}@informatik.uni-heidelberg.de

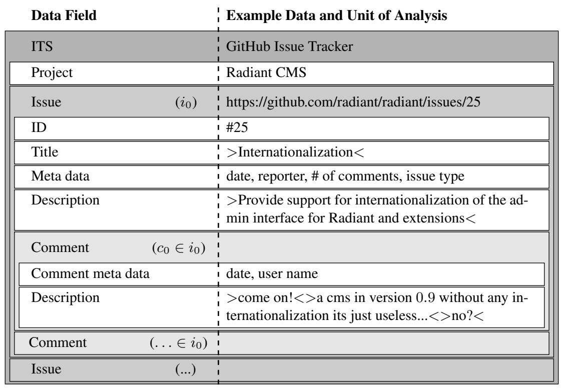
Abstract. [Context and motivation] Communication in distributed software development is usually supported by issue tracking systems. Within these systems, most of the communication is stored as unstructured natural language text. The natural language text, however, contains much information with respect to requirements management, e.g. discussion, clarification and prioritization of features, bugs, and refactorings. **[Question]** This paper investigates the information stored in the issue tracking systems of four different open-source projects. It categorizes the text and reports on the distribution of issue types and information types. **[Principal ideas/results]** A manual analysis of 80 issues, using a grounded approach, is conducted to derive a taxonomy of issue types and information types. Subsequently, the taxonomy is used as a codebook, to manually categorize and structure the text in another 120 issues. **[Contribution]** The first contribution of this paper is the taxonomy of issue and information types and the second contribution is an in-depth analysis of the natural language data and the communication. This analysis showed, for example, that information with respect to prioritization and scheduling can be found in natural language data, whether the ITS supports such tasks in a structured way or not.

Keywords: Issue Tracking Systems, Requirements Communication, Empirical Study, Grounded Method, Content Analysis, Issue Types, Information Types, Taxonomy

1 Introduction

Most software projects use an issue tracking system (ITS) to support software engineering (SE) work [11]. In ITS development, bug fixing or refactoring tasks get tracked and assigned. The information is stored in different data fields like title, descriptions and comments. Most of these data fields contain unrestricted natural language (NL) text accompanied with meta data, like user names or timestamps to complement the NL data (see Figure 1). In contrast to the meta data, the ITS NL data mixes any kind of information from feature requests or bug reports to rationales, implementation ideas or social interaction.

Fig. 1: Typical ITS Issue Structure and Example Issue



Data Field	Example Data and Unit of Analysis
ITS	GitHub Issue Tracker
Project	Radiant CMS
Issue (i_0)	https://github.com/radiant/radiant/issues/25
ID	#25
Title	>Internationalization<
Meta data	date, reporter, # of comments, issue type
Description	>Provide support for internationalization of the admin interface for Radiant and extensions<
Comment $(c_0 \in i_0)$	
Comment meta data	date, user name
Description	>come on!<>a cms in version 0.9 without any internationalization its just useless...<>no?<
Comment $(\ldots \in i_0)$	
Issue (...)	

If requirements engineering (RE) is practiced ad-hoc, software requirements documentation and plans may exist, but this information is typically out-of-date [2]. Therefore, ITS are often the sole means to document, which software features have been implemented or deployed in a certain software version or which feature still needs to be scheduled for implementation. However, ITS's are neither used nor designed as documentation systems, although they contain valuable information for the software development process and for retrospective analysis. ITS NL data is typically unstructured and most of the metadata does not add meaning to the NL. Additionally, the meta data, that adds semantics, like the common *feature vs. bug* categorization of issues, is often unreliable. E.g. Herzig et al. [4] found in their study, that about a third of this meta data is stored wrongly in open-source (OSS) projects.

This suggests that the NL data, that is used in issues, needs to be analyzed more thoroughly. This paper analyzes ITS NL data and focusses on the following questions a) what kinds of issues and what information can be found in ITS NL data, b) how are these information types related to software requirements and c) how do aspects of the communication differ in different OSS projects?

The next section describes study design, details on the research questions, the data collection procedures, and the data analysis procedures. Section 3 presents the results for each research question. The Sections 4 and 5 describe the threats to validity and similar studies. Finally, Section 6 concludes the paper.

2 Study Design

2.1 Research Questions

In our research questions we use the concepts of *information type* and *issue type*. An information type describes the information, that is carried by one or more sentences in ITS NL data (see the rightmost nodes in Figure 4). An *issue type* is considered a context or frame for *information types* (see the leftmost nodes in Figure 4). E.g. the information type *request* can be used in different *issue types*, such as *feature request*, *request for* *fix*ing a bug or *request for refactoring*. The main study goals are broken down into the following three research questions:

RQ1 What are the issue types and information types captured in ITS NL data?
RQ2 What is the distribution of different issue types and information types?
RQ3 Are issue types and information types used differently in different projects?

2.2 Case and Subject Selection

We sampled from the following four OSS projects, since information from OSS projects is easily available. Furthermore, there are large volumes of data and communication activities stored in ITSs, if people work in a distributed manner [3]:

– **c:geo**, an Android app to play the real-world treasure hunting game Geocaching
– **Lighttpd**, a HTTP web server for static and dynamic content delivery
– **Radiant**, an extensible content management system application
– **Redmine**, an extensible issue tracking system with web and REST API interfaces

To answer RQ3, we chose projects with very different characteristics (see Table 1). These characteristics have influence on the NL data. e.g. in lighttpd, issues are more detailed and technical than in c:geo. We guess that this is due to the different audiences and the technical nature of a server application.

Table 1: Project Characteristics

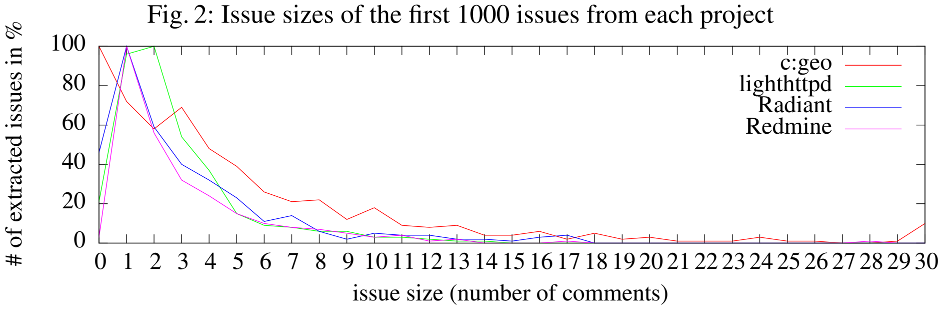
	c:geo	Lighttpd	Radiant	Redmine
Software Type	Android app	HTTP server	content management system	ITS
Audience	consumer	technician	consumer / developer	hoster / developer
ITS	GitHub	Redmine	GitHub	Redmine
ITS Usage	ad-hoc	structured	ad-hoc	very structured
ITS size (in # of issues)	$\sim$ 3300	$\sim$ 330	$\sim$ 3300	$\sim$ 13.000
Main programming lang.	Java	C	Ruby	Ruby
Project size (in LOC)	$\sim$ 115000	$\sim$ 41000	$\sim$ 33000	$\sim$ 150000

2.3 Data Collection Procedures

Figure 1 shows the typical issue structure of an ITS on the left and an example issue on the right. Each sentence of the ITS title, description, and comment fields, that contain NL, is marked with “>” and “<”. Those fields will be referred to as *ITS NL data* in the remainder of this paper. All ITS NL data and meta data, that can be retrieved from the respective ITS APIs, was extracted, converted into a unified naming schema, and stored in an XML-format. However, attachments were not extracted, since these a) are generally used for technical information like code snippets, log files, or stack traces and b) typically use various formats (e.g. jpeg images or Microsoft Word documents), which cannot be converted to plain text easily .

We extracted the first 1000 issues[3] from each project and divided the issues in three sets: I_s includes all issues with less than the median number of comments per project (on average $0 - 2$), I_m includes all issues with the median to the mean number of comments (on average $2-5$), and I_l includes all issues with more than the mean number of comments (on average $6- \sim 70$). Most issues fall in the classes I_s and I_m as shown in Figure 2.

Fig. 2: Issue sizes of the first 1000 issues from each project



To develop the taxonomy, 80 issues were randomly drawn from the I_m and I_s sets. For further analysis another 120 issues were drawn by two coders, equally over each project and each set, to make sure that the sample includes all issue sizes and all projects. Details of the research process are presented in the next section.

To process the data, the General Architecture for Text Engineering (GATE) [1] was used. GATE could be used a) for the grounded approach to calculate inter-rater agreements, b) to enforce the use of the taxonomy for deeper analysis, and c) for text analysis, e.g. to retrieve statistical data or to add additional meta data to the text (our own plugins were used for analysis and the Stanford Parser [7] for annotating the english language).

2.4 Analysis Procedures

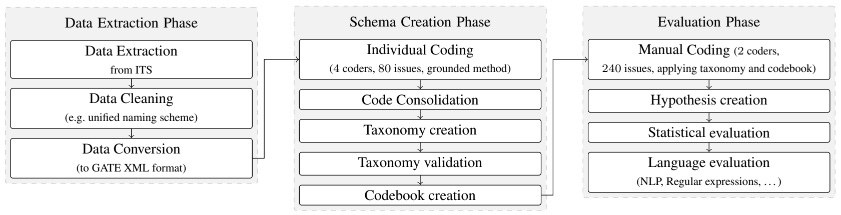
In this section we present the details of our analysis process. An overview of the process is presented in Figure 3.

A taxonomy of Information Types: One of the major problems in this study, was to identify the issue types and information types used in the ITS NL data. Although earlier studies analyzed ITS NL data, they focussed on specific information types, like discussions [3,6] or the categories of issue types [4]. In contrast, this study tries not to focus on a certain aspect of ITS NL data, but provides a broader taxonomy of the issue and information types.

Therefore, a grounded technique [8] was used to create a taxonomy of issue and information types. Four of the authors manually coded all sentences in 20 issues. During this process, each coder developed his own taxonomy. The only requirements for the taxonomy development were, that a) issue and information types should be distinguished, and b) each sentence should be coded. Intentionally, all coders used a two-phase schema. The first phase represents the *issue type* (e.g. feature-, bug-, or software development process-related information) and the second the *information type*

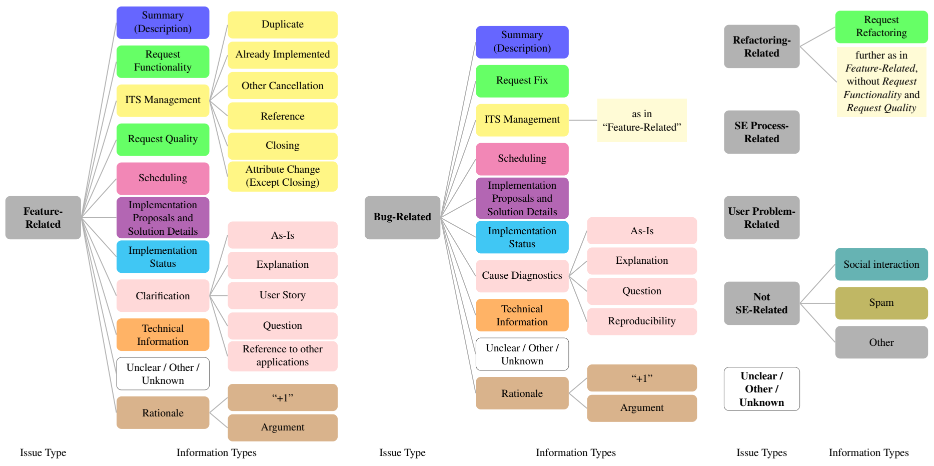
[3] Paech et al. suggest that most requirement-related information can be found in early issues [9]

Fig. 3: Research Process Overview



Data Extraction Phase
Data Extraction from ITS
Data Cleaning (e.g. unified naming scheme)
Data Conversion (to GATE XML format)
Schema Creation Phase
Individual Coding (4 coders, 80 issues, grounded method)
Code Consolidation
Taxonomy creation
Taxonomy validation
Codebook creation
Evaluation Phase
Manual Coding (2 coders, 240 issues, applying taxonomy and codebook)
Hypothesis creation
Statistical evaluation
Language evaluation (NLP, Regular expressions, …)

Fig. 4: A taxonomy of ITS Issue Types and Information Types



(e.g. functionality or quality request, clarification question, or *as-is* descriptions). To consolidate the schemas, we created a large schema out of all issue and information types from the 80 coded issues. This schema included 14 issue types from coders c_i ($5_{c1} + 3_{c2} + 3_{c3} + 3_{c4}$) and 127 information types ($45_{c1} + 25_{c2} + 36_{c3} + 21_{c4}$). To consolidate the schema, all synonyms were merged (e.g. one coder named a sentence *suggested solution*, another *solution*, and a third *potential solution*). The remaining information types were discussed and during discussions, the coders found that information types are sometimes bound to an issue type and sometimes neutral. An example is the *as-is* information type. It describes the current status of the software and can be used for feature-related issue types to describe the context of a new feature, or bug-related issue types to describe the problematic behavior as it is in a certain software version. These neutral information types were simply added to all issue types. So the *as-is* information type is present in feature- as well as bug-related issue types as shown in the final taxonomy in Figure 4. After the consolidation, a code book was created. The codebook was tested on another 4 issues from the I_{ml} sets by each coder with an inter-rater agreement of 0.9869 (Cohen's Kappa $= 0.4630$) for bug-related and 0.8152 (Cohen's Kappa $= 0.6786$) for feature-related codes. The results of this test run, especially differences, were discussed again. Finally, the descriptions of the codebook were updated, such that the coders had a common understanding of each issue and information type.

Further Data Analysis: The taxonomy and code book were implemented in GATE and two of the coders used the schema to code 60 issues, each. To maximize the variance in issue selection, each coder drew 5 issues from I_s, I_m and I_l for each project. Overall 120 issues and 3167 sentences, as summarized in Table 2 were analyzed.

Table 2: Population, Sample and Coding Sizes

Project	open/closed issues	extracted issues	analyzed issues	analyzed comments[1]	coded sentences
c:geo	$425/2829$	first 1000	30	322	662
lighttpd	$53/272$	all	30	204	748
Radiant	$425/2829$	first 1000	30	218	592
Redmine	$4400/9244$	first 1000	30	374	1160
Sum			120	1118	3167

[1] Each issue consists of one title, one description and multiple comments (C). Therefore, overall $C+60$ ITS data fields were analyzed.

3 Results

3.1 RQ1: Information Types and Issue Types in ITS Data

The taxonomy shown in Figure 4 includes 6 *issue types* and 28 *information types*. The information types *ITS Management, Clarification, and Rationale* were again split into subtypes. The white fields *Unclear/Other/Unknown* are for information, that could not be classified by the coders.

Issue Types The following issue types were discovered: 1) *Feature-Related* – information, related to a new software feature or software requirements, 2) *Bug-Related* – software failures and problems, 3) *Refactoring-Related* – software changes that neither affect the functionalities nor the qualities of the software (besides maintainability), 4) *SE Process-Related* – discussions about the general SE process, e.g. if a developer notices that tests should be run more frequently in the project or if documentation should be relocated, 5) *User Problem-Related* – problems that are not related to software development, e.g. a user does not understand a configuration file and asks for help – and 6) *Not SE-Related* – anything, that is not related to software engineering activities, such as social interaction between developers .

Information Types For the issue types *Feature-Related, Bug-Related and Refactoring-Related*, the taxonomy provides detailed information types. Information types that reoccur in feature-, bug- and refactoring-related issues are represented with the same color in Figure 4.

Issues generally start with a *summary* (dark blue) in the title. The summary describes (certain aspects of) the issue and does not form a whole sentence (e.g. “more flexible bandwidth limiting”). Some bug-related issues start with an *as-is clarification* (light rose) in the title to denote what needs to be fixed and some feature-related issues put the request in the title (e.g. "provide an infrastructure for content-filtering"). An obvious information type is the *request* (green) itself. A *request* can be found in all software engineering-related issue types and is sometimes accompanied with *rationale arguments* (brown), emphasizing why the feature should be implemented. Rationales, however, are generally given later in a comment of the issue, e.g. when a user notices, that more support is needed to get a feature requests implemented, or by other users, who express that the issue is important for them as well. We found *rationales* that gave *arguments* and also ones which simply tried to up-vote an issue ($+1$). Especially in features, question-/answer-pairs for *clarification* (old rose) often occur after the original feature request. If this happens, more elicitation is needed to understand and implement the feature request. For bug-related information, we named the clarification phase *cause diagnostics* (light green), since the bug descriptions generally did not need clarification, but the actual cause of the problem had to be found, e.g. by providing reproducibility information. The *as-is* status of the software is often used to describe the problem in a bug. Sometimes even small *user stories*, consisting of one or two sentences, were given to clarify or motivate a new feature. Besides understanding the request and performing the actual implementation, *implementation proposals or solution ideas* (purple) are discussed.

Another common information type is *ITS management* (yellow), which is used in bug-, feature- and refactoring-related issues. *ITS management* describes NL data wrt. the management of the current issue and is therefore divided in subtypes like referencing other information, closing the issue, mentioning duplicated issues or changing attributes of an issue. Interestingly, this information can be handled by the ITS itself and is therefore not really necessary in the NL data. Some users, however, prefer not to use the ITS mechanisms and express duplicates or references in the NL data fields. In contrast to ITS management, *SE process*-related information was defined as the NL that discusses the SE process as a whole.

We did not find much information wrt. to prioritization. Generally, most *scheduling* (magenta) is done in a very pragmatic way. E.g. developers comment: "I will look into this tomorrow" or "We should delay this feature". On the other hand, information regarding the *implementation status* (light blue) is often communicated, e.g. "this was fixed in update 10" or "I already implemented part X of this issue".

Since we were interested in RE aspects of ITS usage, we did not look into the issue types *SE Process-Related, User Problem-Related and Not SE-Related* in such detail.

3.2 RQ2: Distribution of Issue Types and Information Types

Reporting in detail on each information type would go beyond the scope of this paper since many information types did not show patterns in their distribution. We did, however, create matrices of all issue and information types. These matrices include, how issue and information types are distributed in the NL ITS data fields (title, description and comment $c_1 \dots c_n$), combinations of the types, and relations between information and issue types. This data is available for download[4]. The following paragraphs report

[4] http://www2.inf.fh-bonn-rhein-sieg.de/∼tmerte2m

on some of the findings from these matrices and focus on feature-related information, since this is most relevant for RE.

Issue Types: Table 3 shows how issue types are distributed in each project. The maximum numbers are printed in bold font. Qualitatively, as expected, most issues include only *bug-related*,*feature-related*, or *refactoring-related* information. However, in some issues, aspects of different issue types are discussed. 4 issues include, bug- and feature-related information. An example is c:geo issue #365. It first describes the *as-is* situation of a bug. In this particular case, a certain color marking (similar to an icon) is missing in the application. Then, *cause diagnostics* of the bug are performed and during this discussion, *implementation ideas* for new features (e.g. user configurable color markings and priority handling for markings) are proposed. Although the *feature requests* are related to the original bug description, they go beyond the original problem and form new *feature requests*. We found this combination only in longer issues ($1 \subset I_m$ and $3 \subset I_l$), since the discussion starts about one topic and takes some time to drift into other topics. In 4 issues the *SE process* is discussed as a digression of a *feature-related* discussion. This combination occurred, when conditions of a certain issue have enough generality to discuss the overall *SE process*. However, we did not find a single issue, that explicitly discusses the *SE process* only. Another interesting combination are *user problems* and *feature-related* information. It occurred in two ways: Firstly, when a user has a very specific problem, that actually does not affect software changes and then ideas for new features pop up. Secondly, if users suggest a feature and it is already implemented. Then the *feature requests* turns into a *user problem*, e.g. how to find a certain checkbox (e.g. Redmine issue 638). In practice these combinations of issue types in a single issue suggest that ITSs should offer better refactoring possibilities, e.g. the extraction of a related issue from one or many comments.

Surprisingly, *Not SE-related* information occurs most often and is at the same time very short (# of sentences). Mostly, because of small acts of politeness between the stakeholders. For example, users often thank for “the great project” or for a “fast reaction” on feature requests or bugs. Although we did not explicitly analyze the sentiment in ITS NL data, no coder found any maleficent social interaction in the ITS.

Information Types: Table 4a shows the 20 issue types occurring most often and their combinations. Bug-, feature- and refactoring-related *overview*s are used only in the title. However, in bugs sometimes the *as-is* situation is used in the title (16) to describe the bug, and in features and refactorings, sometimes a request (e.g. “please add functionality . . . ”) is used instead of a short overview (10 and 4).

Out of the 51 issues with feature-related information, 18 include *clarification questions* and 16 *clarification explanations*. These information types are used to detail the feature or according solution ideas. For about 50% of the issues, no further clarification was needed. In only 6, we found *implementation or solution-related* information. Solution ideas were mentioned mostly before any clarification information. One explanation for this is, that the *solution* helped to start a discussion. In contrast, almost all bugs (58) contained *cause diagnostics* and 28 times *technical information*, like stack traces or log files, is added. For 28 bugs, explicit *reproducibility* information is added.

Table 3: Distribution of Annotated Issue Types

	Overall		c:geo		lighttpd		Radiant		Redmine	
Issue type	occurrences	sentences	occ.	sent.	occ.	sent.	occ.	sent.	occ.	sent.
Bug-Related	59	**1532**	13	**300**	**22**	**614**	13	**246**	11	372
Feature-Related	51	1145	**17**	267	6	58	7	141	20	**674**
Not SE-Related	**72**	222	14	29	19	53	**16**	58	**23**	82
Refactoring-Related	17	192	6	48	1	1	9	132	1	11
SE Process-Related	11	19	6	10	2	2	3	7	-	-
User Problem	9	11	1	1	2	20	1	8	4	11
Unclear or Unknown	3	46	2	7	-	-	-	-	2	10
Sum	222	3167	59	662	52	748	49	592	61	1150

Table 4: Issue and Information Types

(a) Top Combinations of Issue Types (≥ 2 occurrences)

Issue Types	Occurences
BR only	46
FR only	38
RR only	11
FR, BR	4
BR, SE process-rel.	4
user problem	3
FR, user-problem	3
BR, refactoring-rel.	2
SE process rel., refactoring rel.	2

(b) TF-IDF Scores

Keyword	BR	FR
function	0.04	0.96
implement	0.04	0.91
feature	0.06	0.91
problem	0.87	0.06
exception	0.14	0
cause	0	0.89

(c) Top 20 Information Types

Issue Types	Occurrences	Sentences
BR → Technical Information	28	642
BR → Cause Diagnostics → Explanation	31	218
FR → Solution Implementation	28	188
Not SE → Social Interaction	62	167
BR → Cause Diagnostics → As-Is	42	149
BR → Cause Diagnostics → Reproducibility	28	141
FR → Implementation Status	32	110
FR → Rationale → +1	17	105
FR → Rationale → Argument	25	99
FR → Request Functionality	40	91
FR → Scheduling	22	71
BR → Cause-Diagnostics → Question	19	68
FR → Clarification → Explanation	16	66
BR → Implementation-Status	29	63
FR → ITS-Management → Reference	25	63
BR → Solution-Implementation	19	59
FR → Technical Information	6	58
User Problem	9	46
FR → Clarification As-Is	17	44
FR → Clarification Question	18	42

Other Observations: Some hypotheses evolved during coding, which could partially be confirmed: Firstly, that *bug-related* issues contained much more *technical information* (e.g. source code, stack traces, and log files) than feature-related issues ($H1$). Secondly, that *technical information* in *feature-related* issues is posted later than in bug-related issues ($H2$). Whereas $H1$ seems to be true (642 sentences of technical information in bugs vs. 58 in features), $H2$ only partially holds. Although 20 technical sentences could be found in feature descriptions, none in comments c_1 and c_2, and most in comments c_3 to c_11 up to c_22, the situation for bugs was similar: 305 technical sentences in the description and another 337 in comments c_1 to c_12. This implies that early and much technical information may be used as an indication to identify bug-related issues.

In terms of issue lengths, *feature-related* and *bug-related* issues are roughly of the same size (up to over 30 comments). Another hypothesis was that *bug-related* issues are shorter, since they need to be resolved quickly and they generally do not involve so many users ($H3$). Although more bug-related issues ($39\% \in I_s$) than feature-related issues ($25\% \in I_s$) were very short, the same medium and long issues were found ($25\% \in I_{ml}$). However, late comments in feature-related issues are often longer and include more discussions. *Refactoring-related* issues are mostly short (no more than 9 comments). We assume that these issues are mostly used as a reminder for the developers and no further discussion is necessary.

Herzig et al. [4] found that issue types are often classified wrongly. Besides the issues, that contain multiple issue types, as mentioned above, we found only 1 wrongly classified issue in the projects using the Redmine ITS. It seems that the number of correctly labeled issues varies between different projects, since Herzig et al. researched other projects than this paper. Furthermore, Redmine itself is an ITS, so the users may have more discipline. In the GitHub based projects, most issues are not categorized at all, since an issue does not need to be marked as bug or feature. A tag can be assigned manually, which simply is not done most of the times. So, besides the project, the ITS's architecture seems to have influence. In practice, the ITS should be chosen and customized according to the needed meta data, since optional meta data (e.g. tags) are often omitted. Furthermore, defaults for meta data fields should be chosen wisely (e.g. if an issue is categorized as bug per default this may never be changed. We recommend a neutral category such as "undecided" as default to prevent such problems).

We also analyzed the ITS NL data for keywords (using e.g. the popular Term Frequency-Inverse Document frequency, TF-IDF, metric) to check, if issues can be easily categorized. An excerpt of promising keywords is shown in Table 4b. For issue types, some obvious keywords, e.g. "bug", "problem" or "feature" can give a strong hint on the correct issue type, but there is still a chance of false positives. E.g. in Redmine the keyword "bug" was found in two feature and only one bug-related issue. Furthermore, the keywords only occur in a minority of the issues so that the recall is also low. For information types, no keywords could be identified.

3.3 RQ3: Project Differences

As shown in Table 3, the 30 analyzed issues for each of the projects c:geo, lighttpd, and radiant contain roughly the same amount of sentences ($592 - 748$). The Redmine issues are significantly larger with 1160 sentences. The Redmine project is also older than the other projects, so one possible explanation is, that features and bugs in Redmine are harder to describe due to the project size. Furthermore, some issues in Redmine were significantly older than in the other projects, therefore another explanation is that old issues (and especially features, see Table 3) get reactivated after some time and need to be discussed again (An example can be found in issue #285 comment #27: "Holy cow, this issue is [. . .] over six years old, and we're still asking what the feature means?".

Besides different issue sizes, some information types were also used differently. In the lighttpd project, 51% of all sentences are *bug-related technical information*. Redmine and Radiant have around 15% technical information. c:geo on the contrary, includes only 2% technical sentences, even though in this project, the maximum amount of sentences (25%) was composed of *bug-related cause diagnostics* and *reproducibility* information. Our hypothesis is, that this is due to the audience and project type. Lighttpd is a server application and bug- as well as feature issues often include configuration snippets. Bugs are also reported by technicians who run a server and therefore stack traces and log files are often included. c:geo on the other hand, is mostly for ordinary users who want to play the geo caching game. They seem to report bugs as well as feature requests on a higher level of abstraction and do not want to deal with technical details. In practice this implies that the content of a (good) feature or bug report largely depends on the project type and audience.

Also, scheduling activities, such as prioritization, differ in the projects. In Radiant and c:geo, there is more talk about scheduling ($\sim$ 30 sentences) than in lighttpd (7). Redmine (normalized over all issues) is about in between. We think that the high amount of explicit scheduling mentioned in the ITS NL data is due to the fact that the GitHub issue tracker does not provide the same flexible mechanisms for scheduling as the Redmine ITS does. E.g. in the Redmine and lighttpd projects, the *Milestone-* and the *Roadmap-Feature* of the Redmine ITS are extensively used and issues get prioritized and scheduled by assigning them to a certain milestone or software version. In the GitHub based projects, this needs to be communicated in the NL. Still, it can be observed that in all projects scheduling is mentioned in the NL, although the Redmine ITS offers extensive scheduling features.

4 Threats to Validity

Construct Validity: Overall, we used the guidelines of Runeson [10] for the analysis. The taxonomy was created by using a grounded technique [8]. To ensure the validity of the taxonomy, all coders created and discussed a codebook that was used during the rest of the study. However, the taxonomy is not very fine-grained and may therefore not be appropriate for other research without modification or addition. The content analysis was done by two coders without redundancy and inter-rater agreement. We tried to minimize this threat with a) test issues (with relatively high inter-rater agreement, considering that every possible NL sentence was coded), b) extensive discussions on coding and c) a codebook. Furthermore, the coders worked in the same room, so that they could ask each other if a sentence was unclear. *External Validity:* We sampled data from four different OSS projects. These projects represent some characteristics of software development projects as discussed in Section 2.2. The results can be transferred to similar project settings. However, we are well aware that we researched only 30 issues per project due to limited resources. We do not claim that our results are statistically significant. We think that project factors influencing the use of ITSs and ITS NL data need to be investigated in depth, to make results transferable between different projects.

5 Related Work

One of the first studies, that includes information types in SE was provided by Kitchenham et al. [5]. They defined an ontology that includes multiple SE aspects, such as product and process information of software maintenance. The ontology is defined on a document level and does not dig deeper into the content of these documents. Herzig et al. present different categories for *bug-related* issue types [4]. Their study does, however, not include issues with multiple issue types and they did not analyze the issues on a sentence level. Ko et al. analyzed discussions in bug reports [6] in depth. They provide detailed categories for the discussion elements. No study, however, analyzed all ITS NL data on a per sentence basis.

6 Conclusion and Future Work

This study presented a taxonomy of issue and information types in ITS. It also analyzed the ITS NL data of 120 issues in depth and presents insight into this analysis. In about 50% of the feature requests, that were analyzed, no further clarification of the request was needed and also in only about 50% a solution was described. This implies that often a single sentence or a small description can be sufficient to implement a feature. The study also found that the information types in ITS NL data are influenced at least by the project type, audience, and even the technical opportunities of the used ITS. With the analyzed data of 120 issues, it was almost impossible to identify keywords that can be used to find information types. Also no clear communication patterns could be found.

In future we work on methods to (partially) categorize ITS NL data automatically, e.g. using NL processing methods. Data categorization is a precondition to support SE development tasks and retrospective analysis of ITS data.

Acknowledgments

This work is partly funded by the Bonn-Rhein-Sieg University of Applied Sciences Graduate Institute.

References

1. Cunningham, H., Maynard, D., Bontcheva, K.: Text Processing with GATE (Version 6). University of Sheffield Department (2011)
2. Ernst, N.A., Murphy, G.C.: Case studies in just-in-time requirements analysis. In: 2012 2nd IEEE Intl. Workshop on Empirical RE (EmpiRE). pp. 25–32. IEEE (Sep 2012)
3. Fitzgerald, C.E.B.: Structured Discussion and Early Failure Prediction in Feature Requests. Phd, University College London (2012)
4. Herzig, K., Just, S., Zeller, A.: It's Not a Bug, It's a Feature: How Misclassification Impacts Bug Prediction. In: Proceedings of the 2013 Intl. Conference on Software Engineering (ISCE). pp. 392–401. IEEE Press (2013)
5. Kitchenham, B.A., Travassos, G.H., Mayrhauser, A.V.O.N.: Towards an Ontology of Software Maintenance. J. of SW Maintenance: Research and Practice 389(May), 365–389 (1999)
6. Ko, A.J., Chilana, P.K.: Design, discussion, and dissent in open bug reports. In: Proceedings of the 2011 iConference. pp. 106–113. ACM Press, New York, New York, USA (2011)
7. Marneffe, M.D., MacCartney, B., Manning, C.: Generating typed dependency parses from phrase structure parses. In: Proceedings of the Sixth Intl. Conference on Language Resources and Evaluation, LREC 2004. Lisbon, Portugal (2006)
8. Neuendorf, K.A.: The Content Analysis Guidebook. SAGE Publications (2002)
9. Paech, B., Hubner, P., Merten, T.: What are the Features of this Software? In: ICSEA 2014, The Ninth Intl. Conference on Software Engineering Advances. pp. 97–106 (2014)
10. Runeson, P., Höst, M., Rainer, A., Regnell, B.: Case Study Research in Software Engineering. Guidelines and Examples. Wiley, Hoboken, NJ, USA, 1st edn. (2012)
11. Skerrett, I., The Eclipse Foundation: The Eclipse Community Survey 2011 (2011)