Identifying Content Types of Messages Related to Open Source Software Projects

Ioannis Korkontzelos, Paul Thompson and Sophia Ananiadou
National Centre for Text Mining, School of Computer Science, University of Manchester, UK
{ioannis.korkontzelos, paul.thompson, sophia.ananiadou}@manchester.ac.uk

Abstract

Assessing the suitability of an Open Source Software project for adoption requires not only an analysis of aspects related to the code, such as code quality, frequency of updates and new version releases, but also an evaluation of the quality of support offered in related online forums and issue trackers. Understanding the content types of forum messages and issue trackers can provide information about the extent to which requests are being addressed and issues are being resolved, the percentage of issues that are not being fixed, the cases where the user acknowledged that the issue was successfully resolved, etc. These indicators can provide potential adopters of the OSS with estimates about the level of available support. We present a detailed hierarchy of content types of online forum messages and issue tracker comments and a corpus of messages annotated accordingly. We discuss our experiments to classify forum messages and issue tracker comments into content-related classes, i.e. to assign them to nodes of the hierarchy. The results are very encouraging.

Keywords: manually annotated corpus, document classification, open source software

1. Introduction

Reusing freely available Open Source Software (OSS) can provide cost effective software solutions to all types of potential users, ranging from individuals to large organisations. Although reusing OSS can be beneficial in software development, it is not entirely risk-free, because OSS quality can vary widely (Spinellis and Szyperski, 2004). The quality of an OSS project also depends on maintenance and can vary substantially over time, depending on the interest of the community of its developers and users. Therefore, deciding whether to adopt an OSS project requires an assessment of its quality at the time of adoption and possibly an estimate of the perceived enthusiasm of the community towards it in the future.

Assessing OSS software quality has traditionally focused on analysing the source code behind the software to calculate quality indicators and metrics. However, complimentary information about OSS quality can be extracted by analysing messages posted to communication channels (newsgroups, forums, mailing lists), and issue trackers supporting OSS projects. For example, analysing online communication related to an OSS project can provide information about the speed at which the community responds to user requests, the rate that bugs are fixed or the satisfaction of users about the responses they receive to their requests.

In our previous work, we investigated the task of identifying request and responses among online messages about OSS (Korkontzelos and Ananiadou, 2014). This classification allowed us to compute a preliminary level of metrics that indicate the quality of support offered online. For example, we computed metrics such as the number of unanswered threads or issues in a communication channel or the average time taken to respond to a request. In this paper, we classify messages according to a more fine-grained set of content types. A more informative set of content types will allow the design of more fine-grained quality-indicating metrics. For example, the broad class of responses (Korkontzelos and Ananiadou, 2014) can be split into more informative types, so as to distinguish between:

- messages sent by the author of the initial request vs. messages sent by other users
- solutions vs. suggestions
- different kinds of redirections to other discussions or documents that solve the problem
- messages sent by the author of the initial request reinforcing it vs. messages sent by other users facing the same problem
- notifications that a bug was fixed in the source code vs. notifications that a bug has been addressed in a patch

In section 3., we present a novel hierarchy of content types of online communication messages, which is much more detailed than just requests and replies. Based on message content semantics, the hierarchy captures types of messages frequently encountered in different online communication channels, such as forums and newsgroups, or in issue trackers related to OSS projects. In section 4., we present the OSSMETER Threaded Corpus, a manually corpus which contains $1,165$ randomly selected Bugzilla and forum threads related to *eclipse* projects. The corpus has been annotated manually by 4 experts, who assigned each message to one or more content types in the hierarchy. We present statistics on the corpus, on the annotations and on inter-annotator agreement. During the process of annotating the corpus, annotators' feedback and inter-annotator agreement scores were used to iteratively identify and improve shortcomings of the hierarchy.

In section 5., we present our experiments to train machine learners able to assign one or more content types to previously unseen messages. We considered the full multi-label classification task and a single-label approximation of it. The best accuracy achieved for the multi-label setting is 62.4%, while the best performance for the single-label setting is 70.0%. The most frequent class baseline is 28.3% while the inter-annotator agreement score on annotating the OSSMETER Threaded Corpus is approximately 75%, setting an upper bound to the maximum accuracy that

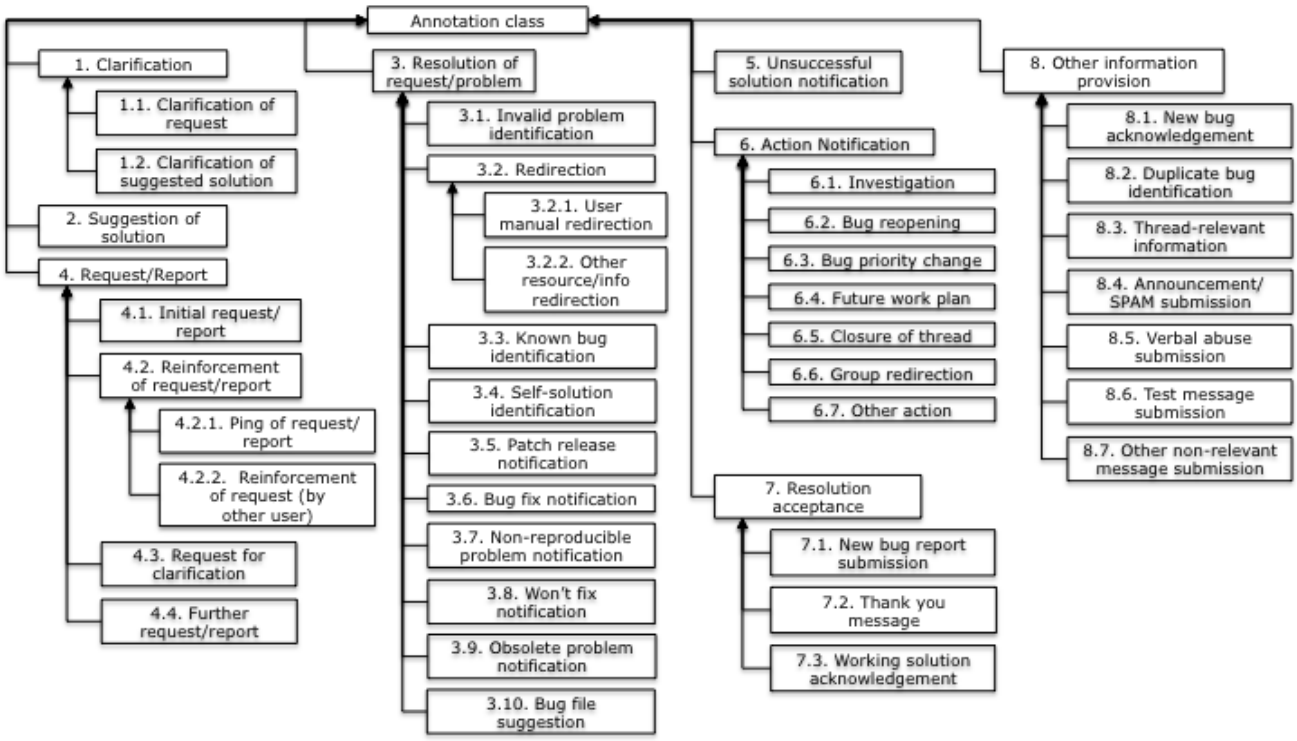

Figure 1: Diagram of the revised hierarchy of types

can be achieved by a machine learner. The results are encouraging, given the complexity of the task and the size of the hierarchy. The best performing classifier has been integrated into the platform of the OSSMETER project (`ossmeter.org`), which aims to aid decision makers in assessing the quality of OSS projects.

2. Related work

The idea of identifying requests and responses within online communication has been applied to email (Shrestha and McKeown, 2004; Carvalho and Cohen, 2005; Lampert et al., 2008), online forums (Ding et al., 2008; Cong et al., 2008) and online forums and issue trackers related to OSS (Korkontzelos and Ananiadou, 2014). More detailed than the coarse-grained classes of requests and responses, dialogue acts have been employed extensively to identify the structure of student discussion threads (Kim et al., 2006) and forum threads (Lin et al., 2009; Kim et al., 2010; Wang et al., 2011). In contrast to these works, the present work does not pursue the dialogue structure of threads, but instead proposes a detailed set of types and assigns them to messages.

Conversation disentanglement is another aspect of discussion analysis. Based on the idea that multiple conversations may occur simultaneously, the task is to identify to which conversation each message corresponds (Elsner and Charniak, 2008; Wang et al., 2011). In the present work, discussions are already threaded and we hypothesise that there is a single discussion in each thread. Conversation focus detection is the task of identifying the message in a thread that contains the most important information (Feng et al., 2006). This task is also similar to our work, since it is also based on analysing the content of threaded messages.

The most similar work to the present one focuses on classifying messages according to their purpose in a discussion thread (Bhatia et al., 2012). A small set of message types are used: asking a question, repeating a question, asking for clarification of a request, providing more details about a request, suggesting a solution and providing positive or negative solution feedback. This set of message types was originally proposed in the second task of the Mailing Lists and Forums (MLAF) Track at the Forum for Information Retrieval Evaluation (FIRE) 2011 `www.isical.ac.in/~fire/2011`. The task aimed at classifying mailing list and forum messages into one or more of these types. Since the set of types is very relevant to the topic of this paper, it was taken into account when developing our novel hierarchy. However, our novel hierarchy of content types is more detailed and contains content types that specifically describe the content of online messages related to OSS.

A similar but distinct area of research focuses on assessing the quality of messages in online forums, adopting a 5-star rating scale (Weimer et al., 2007). Assessing the quality of discussion threads has also been attempted (Kim and Beal, 2006). However, very simple metrics, such as the number of messages, the length of messages and the number of responses, have been employed.

3. Message Content Hierarchy

We develop a hierarchy of types describing the content of messages posted to communication channels about OSS, such as mailing lists, discussion forums and issue trackers. Labelling each message with relevant types allows the measurement of statistics to evaluate the quality of support provided in a communication channel about an OSS project.

ID	Label	Final	Description
1	Clarification	✗	A message that provides additional information, in response to a request to clarify the details of a previous post.
1.1	Clarification of request	✓	A message that clarifies or expands upon the details of a previously submitted request or problem report, in response to a "Request for Clarification".
1.2	Clarification of the proposed solution	✓	A message sent to ask for some clarification of what is described in a previous message that proposes a solution.
2	Suggestion of solution	✓	A message that proposes a solution to a previously submitted request or problem report, which may include background or explanatory contextual information. The suggestion may or may not solve the previously reported problem.
3	Resolution of request/problem	✗	A message that aims to provide a resolution to a previously submitted request or problem report. The message may include additional information that directly supports or elaborates upon the resolution given.
3.1	Invalid problem identification	✓	A message that determines that a previously submitted request/problem will not be solved, because it is not considered to be a genuine software issue (e.g., as determined by a more experienced developer).
3.2	Redirection	✗	A message that redirects the problem reporter to information contained externally from the current thread.
3.2.1	User manual redirection	✓	A message that points the problem reporter to information in a software manual that will help them to solve the problem.
3.2.2	Other resource/info redirection	✓	A message that points the problem reporter to an information source (other than a user manual) that will help them to solve the problem, e.g. another bug report, newsgroup thread or another online resource. Normally includes either a link or a bug report/thread ID.
3.3	Known bug identification	✓	A message that identifies a reported problem as a bug that is already known.
3.4	Self-solution identification	✓	A message from the problem reporter, stating or explaining that they found their own solution to the problem.
3.5	Patch release notification	✓	A message that provides notification that a software patch that solves a previously reported problem has been released.
3.6	Bug fix notification	✓	A message that provides notification that a previously reported bug has been fixed in the source code/binary version. May include details about what has been fixed, and how.
3.7	Non-reproducible problem notification	✓	A message in which a developer provides notification that they were unable to reproduce a problem previously reported by another user.
3.8	Won't fix notification	✓	A message that provides notification that a previously reported problem cannot or will not be fixed.
3.9	Obsolete problem notification	✓	A message that provides notification that a previously reported problem will not be solved, due to software updates that have rendered the reported problem obsolete.
3.10	Bug file suggestion	✓	A message proving a suggestion that the problem reporter should file a bug in some bug tracking system.
4	Request/Report	✗	A message that requests help with software, reports a software problem or requests/suggests a software update.
4.1	Initial request/report	✓	A message in which a user opens a new thread, by bringing to the attention of other developers or newsgroup users either: a) A specific problem or issue with which the requesting user needs help, b) A software problem or bug that needs attention or c) A suggestion for a software improvement.
4.2	Reinforcement of request/report	✗	A message that provides a reinforcement, emphasis or restatement of an initial request/report that was introduced earlier in the same thread.
4.2.1	Ping of request/report	✓	A message submitted by the request initiator to reinforce, emphasise or restate their initial request. This may occur, for example, when another user has tried to close the thread, to convince them of its importance.
4.2.2	Reinforcement of request (by other user)	✓	A message submitted by another user/developer to reinforce or emphasise importance/significance of the initiator's request (e.g., by stating that they have the same problem or issue).

Table 1: Message content types and descriptions (continued on table 2)

For example:

- How many threads are resolved as irrelevant, known bugs, self-solutions, and redirections?
- How often do users reinforce their requests before they are resolved?
- How often do users who requested help receive suggestions that did not work?
- How many requests lead to modifications and improvements to the OSS code?

In addition, message types can be combined with message metadata, such as the author name and the timestamp, to provide extra evaluation statistics, such as the average time taken to resolve a thread.

Despite the multitude of communication means related to

ID	Label	Final	Description
4.3	Request for clarification	✓	A message that requests or implies that further information is required about a previously submitted post in the thread.
4.4	Further request/report	✓	A message that introduces a new request for help or problem report, but which occurs in the middle of a the message thread.
5	Unsuccessful solution notification	✓	A message in which the problem reporter provides notification that a particular suggested solution did not solve their problem.
6	Action Notification	✗	A message that provides notification of some type of action that is planned, ongoing or has already been undertaken, as a step towards resolving a request/problem report.
6.1	Investigation	✓	A message that expresses an intention to investigate a previously reported problem, provides notification that such an investigation is ongoing, or directs other developers to investigate the problem.
6.2	Bug reopening	✓	A message that expresses an intention to reopen a bug, reports that the bug already has been reopened, or provides a directive to reopen the bug, based on new information that has been provided/discovered since the bug was closed.
6.3	Bug priority change	✓	A message that expresses an intention to change the priority of an existing bug, reports that the priority has been changed, or provides a directive for the priority to be changed.
6.4	Future work plan	✓	A message that expresses an intention to address/resolve the bug at some time in the future.
6.5	Closure of thread	✓	A message that expresses an intention to close the current thread, or implies/states that it has already been closed.
6.6	Group redirection	✓	A message that expresses an intention to redirect or move the request to another developer group, or reports that such a move has been carried out.
6.7	Other action	✓	A message that expresses an intention to act in some other way in response to a request, or explains other ongoing or completed action that does not fit into the types of action in this section.
7	Resolution acceptance	✗	A message that indicates that the problem reporter accepts (one of) the resolution(s) that has been proposed by other developers or users.
7.1	New bug report submission	✓	A message from the problem reporter stating that they have submitted a new bug report in a bug tracking system.
7.2	Thank you message	✓	A message from the problem reporter providing explicit thanks for the help provided.
7.3	Working solution acknowledgement	✓	A message from the problem reporter acknowledging that (one of) the suggested solutions provided by other users works for them to solve their reported problem.
8	Other information provision	✗	A message that contains information that does not fit into one of the other class descriptions.
8.1	New bug acknowledgement	✓	A message in which a developer responds to problem report submitted, by acknowledging the problem as a previously unencountered bug.
8.2	Duplicate bug identification	✓	A message that identifies the request/issue as a duplicate of another bug, which may or may not have been previously solved. Often, a link or ID of the identical bug is provided.
8.3	Thread-relevant information	✓	A message that provides information that is relevant to, supports or elaborates upon a previously introduced request/problem report or its solution, but which does not fit into any of the other class descriptions. Includes illustrative examples of code, error messages, etc, statements of software versions, information about testing, etc.
8.4	Announcement/ SPAM submission	✓	A message considered as SPAM within the context of a bug tracking system or newsgroup whose aim is to solve software problems. Such SPAM messages include announcements of information targeted at the community of users, such as events, conferences, job opening, general software release messages, etc.
8.5	Verbal abuse submission	✓	A message containing strong language, which is not directly related to requesting help or suggesting solutions.
8.6	Test message submission	✓	A message that tests whether the communication channel is working (e.g., a duplicate of a previously submitted message or a request for confirmation that a previously submitted message has been received).
8.7	Other non-relevant message submission	✓	Any other message that is not directly relevant to the discussion in the thread, including those that are incomplete or truncated.

Table 2: Message content types and descriptions (continued from table 1)

OSS, we develop a single hierarchy of types, based on the observation that OSS users use any of these means to express their questions, concerns, expectations, bugs and suggestions for enhancements. For example, a message describing a problem during the installation of an OSS project may be submitted to a forum and an issue tracker. Focusing on types rather than on the characteristics of means ensures that the hierarchy will be applicable to any new means of OSS-related communication that may arise in the future.

A first version of the hierarchy was developed by a group of 20 computer science researchers and professionals actively involved in OSS development. A large number

thread length	# Bugzilla threads	# Newsgroup threads
1	0 (0.0 %)	398 (47.8 %)
2	17 (4.9 %)	180 (21.6 %)
3	178 (50.9 %)	95 (11.4 %)
4	75 (21.4 %)	38 (4.6 %)
5	28 (8.0 %)	39 (4.7 %)
>5	47 (13.4 %)	70 (8.4 %)
>10	5 (1.4 %)	13 (1.6 %)

Table 3: Distribution of threads according to length in the Bugzilla and newsgroup part of the corpus

of random messages from eclipse forums (`eclipse.org/forums`) and the respective Bugzilla server (`bugs.eclipse.org`) were inspected. New types were introduced based on the requirement that one should be able to intuitively assign each message to one or more types, but an excessive number of types should be avoided.
Subsequently, 4 annotators were provided with annotation guidelines and annotated a corpus of 3.5K messages (section 4.). Inter-annotator agreement and feedback from the annotators was used to evaluate the coverage and quality of the hierarchy. High levels of agreement would indicate sufficient coverage, while low scores would denote that some messages are not covered or some types overlap semantically. Analysis showed that the hierarchy had to be amended slightly and that the coverage of some classes should be clarified. Accordingly, the hierarchy (figure 1), the type descriptions (Tables 1 and 2) and the guidelines were revised[1]. Extra types were added, while some existing ones were merged to increase clarity and coverage. The corpus was annotated according to the revised hierarchy. Inter-annotator agreement (section 4.) verified that the shortcomings of the first version were resolved. Only leaf nodes, i.e. final types, can be assigned to messages. Non-final classes exist as semantic super-classes of descendants. A message can be assigned to more than one type. e.g. a "Thank you message" (7.2) can also mention that the thread is closing (6.5).

4. Corpus of Message Threads

Since the hierarchy of types was designed to cover messages posted in newsgroups, forums and issue trackers, the OSSMETER Threaded Corpus contains representative messages from these sources. Specifically, the first part of the corpus consists of 345 Bugzilla threads, randomly chosen from the Bugzilla server for eclipse (`bugs.eclipse.org`), and contains 1369 messages. The newsgroup and forum part contains 820 threads, randomly chosen from the eclipse forums (`news.eclipse.org`), that consist of 2004 messages. Threads were chosen randomly, irrespective of the Bugzilla product or component, or newsgroup they belong to, i.e. the probabilities of all Bugzilla or newsgroup threads being selected are equal. Table 3 shows the length distribution of threads in each part of the corpus. Annotations were performed by 4 researchers or professionals actively involved with OSS development and related communication means. Standard inter-annotator agreement metrics which require that category assignments are mutually exclusive (Hripcsak and Rothschild, 2005), such as Cohen's kappa, are not applicable to the current task, because a message can be assigned two or more hierarchical content types. Following Thompson et al. (2009), we compute the average F-measure (F) between annotator pairs, as the geometric mean of Precision (P) and Recall (R). Let A_T, B_T be the set of unique message Ids assigned by two annotators to a final type, T. P, R and F can be computed as follows:

$$P_T = \frac{|A_T \cap B_T|}{|A_T}, \; R_T = \frac{|A_T \cap B_T|}{|B_T|}$$
$$F_T = 2\left(P_T^{-1} + R_T^{-1}\right)^{-1}$$

For non-final types, we could average the F-measure scores associated with its descendants. However, this would not take into account the actual number of annotations of each descendant class. Let N be a non-final type with D descendant types. We employ weighted versions of P and R:

$$P_N = \frac{\sum_{j \in D} |A_j \cap B_j|}{\sum_{j \in D} |A_j|}, \; R_N = \frac{\sum_{j \in D} |A_j \cap B_j|}{\sum_{j \in D} |B_j|}$$
$$F_N = 2\left(P_N^{-1} + R_N^{-1}\right)^{-1}$$

After revising the hierarchy, inter-annotator agreement scores did not deteriorate for any content type while they improved for some types. Table 4 presents inter-annotator agreement scores, accompanied by the average number of messages that were assigned to each type.
Before carrying out classification experiments, we determined which of the annotations to keep as gold-standard. This step filtered out content types that were assigned to a message by a minority of annotators due to misunderstandings or annotation errors. A frequency threshold was used, accepting only annotations that were cross-verified by more than one annotator. Since 4 annotators participated, we can choose a threshold of either 2 or 3. A threshold of 3 means that in the final version of the corpus each message only retains types that were assigned to it by 3 or 4 annotators. The greater the threshold, the more messages have no types assigned and thus cannot be useful for training machine learners. Threads that contain these messages should also be discarded. Applying a threshold of 3 or 2, leaves 12.2% or 1.0% of the corpus unannotated, respectively. Due to the large number of discarded threads for a threshold of 3, we choose to adopt 2 as our threshold. Thus, types that were assigned by at least two annotators are considered as gold standard, and are used to train machine learners, as described in the next section.

5. Classification experiments

The task that assigns one or more types to each message is multi-class and multi-label. Multi-class classification assigns to each instance one out of three or more possible labels. Multi-label classification assigns more than one label to each instance. The task at hand is multi-class, because there are more than two types in the hierarchy and multi-label because a message can be assigned more than one type.

[1]The corpus and guidelines are available on the LRE map.

Type ID	IAA	# messages
hierarchy	**75.4%**	4035
1	*46.9%*	130
1.1	51.3%	75.3
1.2	*40.8%*	54.7
2	**76.9%**	553.3
3	**74.6%**	698.7
3.1	*46.6%*	25.0
3.2	**68.1%**	203.3
3.2.1	*43.8%*	6.3
3.2.2	**69.0%**	197.0
3.3	*40.8%*	14.7
3.4	**67.9%**	67.3
3.5	**69.7%**	4.7
3.6	**89.5%**	221.7
3.7	50.0%	15.3
3.8	**64.5%**	35.0
3.9	**80.3%**	78.7
3.10	**72.6%**	33.0
4	**85.8%**	1354.3
4.1	**97.4%**	949.0
4.2	53.6%	78.0
4.2.1	*42.7%*	31.3
4.2.2	61.1%	46.7
4.3	**67.3%**	241.7
4.4	*38.7%*	85.7
5	57.6%	64.0
6	63.6%	249.7
6.1	*40.1%*	81.0
6.2	**85.2%**	30.4
6.3	**77.8%**	1.4
6.4	*48.0%*	34.0
6.5	**84.3%**	55.4
6.6	**94.1%**	34.0
6.7	*30.8%*	13.7
7	**69.4%**	241.3
7.1	**69.7%**	16.7
7.2	**74.3%**	163.0
7.3	55.4%	61.7
8	**68.8%**	743.7
8.1	*16.1%*	12.0
8.2	**98.1%**	70.0
8.3	**69.4%**	602.0
8.4	*24.3%*	21.0
8.5	—	0.0
8.6	*27.8%*	4.0
8.7	*40.9%*	34.7

Table 4: Inter-annotator agreement (IAA): scores less than 50% are in italics and greater than 65% are in bold face.

We perform classification experiments both in a single-label and a multi-label setting. The single-label setting is an approximation of the full multi-label task. The vast majority (91.73%) of corpus messages are assigned a single label. A single-label approximation of the task is only able to assign one type, rather than all types, to the small minority of messages (8.27%) that are assigned multiple labels. For messages with multiple labels, we kept the most frequent type, in an attempt to minimise the effects of the approximation. Frequency of types was computed over the entire corpus. For example, consider a message in which the author thanks the developers for their help in relevance to his previous request and also expresses a new question. This message is assigned two types, 4.4, "Further request/report", and 7.2, "Thank you message". In the single-label setting, the least frequent of these types, 7.2, is discarded.

position	content type (messages %)
1	4.1 (88.9%), 2 (4.1%), other (7.0%)
2	8.3 (38.6%), 2 (23.1%), 4.3 (7.4%), 3.6 (6.3%), 3.2.2 (5.5%), other (19.2%)
3	2 (14.74%), 3.6 (14.74%), 8.3 (12.35%), 4.3 (9.96%), 1.1 (9.56%), 6.6 (5.98%), 7.2 (4.78%), other (27.89%)
-1	4.1 (23.8%), 2 (16.6%), 3.6 (10.1%), 8.3 (7.4%), 7.2 (7.3%), 3.9 (5.5%), 3.2.2 (5.6%), other (23.6%)
-2	8.3 (21.6%), 4.1 (21.0%), 2 (14.4%), 3.6 (7.0%), 3.2.2 (4.1%), 4.3 (4.1%), other (27.8%)
-3	4.1 (47.55%), 8.3 (12.45%), 2 (11.70%), 3.6 (4.91%), other (23.40%)

Table 5: Content types and percentages per position in thread (-1: last, -2: one but last). Types with percentages $< 4\%$ are summed as *other*.

We used a linear Support Vector Machine (SVM) (Cortes and Vapnik, 1995), because this was the most successful among other classification algorithms (Radial Basis Function (RBF) SVM and Random Forest (Breiman, 2001)) for a simpler but similar binary classification task (Korkontzelos and Ananiadou, 2014)[2]. We observe that the position of a message within a thread is strongly correlated with its content type. For example, in 88.9% of the threads, the first message is labelled with content type 4.1, *initial request/report*, while in 38.6% of the threads, the second message is labelled with type 8.3, *thread-relevant information*. Table 5 presents the distribution of content types of messages that appear in different positions within threads: first, second, third, last, last but one and last but two position.

Classification features include the position of messages in threads and the heuristics that were shown to correlate with requests or non-requests in our previous work (Korkontzelos and Ananiadou, 2014). In particular, the heuristics are the following three observation and their combinations[3]:

- the prefix "RE:" or "Re:" in the message subject
- the occurrence of one or more question marks in the message body
- the occurrence of question words, such as what, when and where in the message body

[2]The SVM implementation used is LIBSVM (Chang and Lin, 2011).

[3]For details see Section 4 in (Korkontzelos and Ananiadou, 2014).

frequency threshold	parts of speech (PoS)	classification accuracy
1	All PoS	70.0%
2	All PoS	70.0%
3	All PoS	69.3%
4	All PoS	69.3%
5	All PoS	69.6%
10	All PoS	69.2%
15	All PoS	68.9%
20	Nouns, Adjectives, Verbs	69.1%
25	Nouns, Adjectives, Verbs	68.8%
30	All PoS	68.7%
35	All PoS	68.7%
40	All PoS	68.6%
45	Nouns, Adjectives, Verbs	68.1%
50	Nouns, Adjectives, Verbs	68.0%

Table 6: Evaluation results for the single-label setting. Indented previous messages were removed and tf-idf feature values were used for unigrams.

frequency threshold	Accuracy	micro-average	macro-average
1	62.0%	65.5%	32.0%
2	62.4%	65.8%	32.3%
3	62.4%	65.6%	32.3%
4	62.1%	65.4%	32.3%
5	61.9%	65.3%	32.2%
10	61.9%	65.2%	31.8%
15	61.5%	65.0%	32.0%
20	61.8%	65.1%	32.0%
25	61.1%	64.3%	31.4%
30	60.9%	64.0%	31.0%
35	60.3%	63.8%	30.0%
40	60.4%	64.0%	30.3%
45	60.2%	63.9%	29.8%
50	60.2%	64.0%	30.2%

Table 7: Evaluation results for the multi-label setting. The *label combination* approach and the linear SVM was used. Indented previous messages were removed. Unigrams of all part-of-speech were considered and encoded with binary values.

Removing previous messages in the thread that appear as indented text in messages was also considered. As features we also considered unigrams that do not occur in a typical stoplist and occur more frequently than a threshold $T \in [1, 2, 3, 4, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50]$. We evaluated unigrams belonging to all parts-of-speech (PoS), or only specific ones, i.e. nouns, nouns and adjectives, nouns, adjectives and verbs. We considered binary, frequency and tf-idf feature values. For all experiments, evaluation was performed on a 10-fold cross validation basis and results are reported as classification accuracy scores.

Table 6 shows the best result achieved for each unigram frequency threshold. Due to the large number of experiments run to cover all setting combinations, we present here only the best performing ones. For all unigram frequency thresholds, the best accuracy was achieved when using tf-idf feature values and considering unigrams of all PoS. For some frequencies, the results are slightly better when only nouns, adjectives and verbs are considered. The best performing unigram frequency threshold is 1 or 2, achieving 70% accuracy. However, using other thresholds can achieve similar accuracy. It should be noted that excluding unigrams or not removing indented previous messages affects the results detrimentally.

To address the full multi-label task, we employed the implementation of the *label combination* approach and the *binary* approach provided by LIBSVM Tools (`csie.ntu.edu.tw/~cjlin/libsvmtools/multilabel`). *Label combination* treats each combination of types assigned to some message as a single class and then proceeds according to the single-label setting. *The binary approach* extends the one-against-all multi-class method for multi-label classification. For each content type, it builds a binary-class problem so that messages associated with that type are in one class and the remaining ones are in the other class. In these experiments, we employed the same feature space as in the single-label setting, while we experimented with both the linear SVM and the RBF kernel. Evaluation was performed on a 10-fold cross validation basis and results are reported as classification accuracy scores.

Table 7 shows the best result achieved for each unigram frequency threshold in the multi-label setting. The linear SVM performed better than the RBF kernel for all experiments, while the *label combination* outperformed the *binary* approach. Accordingly, the table only reports results for these settings. Similarly to the single-label setting, considering unigrams of all parts of speech performed better than selecting only certain parts-of-speech. In all experiments, accuracy is higher when including unigram features and removing indented messages that occur as parts of other messages. The best performing unigram frequency threshold is 2 or 3, but several other values achieve similar accuracies. The fact that micro-average accuracy is much higher than macro-average highlights that the corpus is skewed in terms of the assignment of content types to messages.

Comparing tables 6 and 7, we observe that the accuracy in the multi-label setting is slightly lower. This is expected because the single-label setting is a simpler problem. For the complexity of the task, the accuracies achieved by both settings can be considered encouraging, given that the most frequent class baseline is 28.3%. The most frequent class baseline is the accuracy achieved by a method that assigns the most frequent content type to all messages. In addition, the inter-annotator agreement score on annotating the OSSMETER Threaded Corpus, approximately 75%, sets an upper bound to the maximum accuracy that can be achieved by a machine learner.

6. Conclusion

In this paper, we have presented a fine-grained hierarchy of message content types and a manually annotated corpus of messages threads related to open source software (OSS). The corpus has been annotated manually according to the hierarchy by 4 annotators. The hierarchy and the corpus are intended to be used to train machine learners to identify

the content type of OSS related messages in communication channels offering support to OSS users. Being able to identify the type of each message allows the measurement of statistics about the level of support provided by the community of an OSS project to its users. This information, among other sources, can be valuable to potential users of an OSS project, either people or organisations, to help them to decide if it is suitable for adoption. Apart from the hierarchy of content types, the corpus and the annotation process, we presented extended classification experiments for assigning content types to previously unseen OSS-related communication messages. Evaluation considered 10-fold cross-validation and a large range of settings. The classification accuracy achieved can be considered highly encouraging, given the size of the hierarchy and the complexity of the classification task. As future work, we plan to enrich the feature space of the classifiers assigning hierarchical types to communication channel messages. The current feature set only considers the text, the subject and the position of the message in its thread. We plan to add features generated by the remaining metadata associated with messages, such as author names (Bhatia et al., 2012).

7. Acknowledgments

This work was funded by the European Community's Seventh Framework Program (FP7/2007-2013) [grant number 318736 (OSSMETER)]. The authors would like to thank all OSSMETER partners for their constructive contribution in developing the hierarchy of content types and annotating OSSMETER threaded corpus.

8. Bibliographical References

Bhatia, S., Biyani, P., and Mitra, P. (2012). Classifying user messages for managing web forum data. In *WebDB '12*.

Breiman, L. (2001). Random forests. *Machine Learning*, 45(1):5–32.

Carvalho, V. R. and Cohen, W. W. (2005). On the collective classification of email "speech acts". In *SIGIR '05*. ACM.

Chang, C.-C. and Lin, C.-J. (2011). LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27. Available at `csie.ntu.edu.tw/~cjlin/libsvm`.

Cong, G., Wang, L., Lin, C.-Y., Song, Y.-I., and Sun, Y. (2008). Finding question-answer pairs from online forums. In *SIGIR '08*. ACM.

Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine Learning*, 20(3):273–297.

Ding, S., Cong, G., Lin, C.-Y., and Zhu, X. (2008). Using CRFs to extract contexts and answers of questions from online forums. In *ACL-08: HLT*. ACL.

Elsner, M. and Charniak, E. (2008). You talking to me? a corpus and algorithm for conversation disentanglement. In *ACL-08: HLT*. ACL.

Feng, D., Shaw, E., Kim, J., and Hovy, E. (2006). Learning to detect conversation focus of threaded discussions. In *HLT-NAACL '06*. ACL.

Hripcsak, G. and Rothschild, A. S. (2005). Technical brief: Agreement, the F-Measure, and reliability in information retrieval. *JAMIA*, 12(3):296–298.

Kim, J. and Beal, C. (2006). Turning quantity into quality: Supporting automatic assessment of on-line discussion contributions. In *AERA '06*.

Kim, J., Chern, G., Feng, D., Shaw, E., and Hovy, E. (2006). Mining and assessing discussions on the web through speech act analysis. In *WCMHLT '06 Workshop*.

Kim, S. N., Wang, L., and Baldwin, T. (2010). Tagging and linking web forum posts. In *CoNLL '10*. ACL.

Korkontzelos, I. and Ananiadou, S. (2014). Locating requests among open source software communication messages. In *LREC '14*. ELRA.

Lampert, A., Dale, R., and Paris, C. (2008). The nature of requests and commitments in email messages. In *EMAIL-08*.

Lin, C., Yang, J.-M., Cai, R., Wang, X.-J., Wang, W., and Zhang, L. (2009). Modeling semantics and structure of discussion threads. In *WWW '09*. ACM.

Shrestha, L. and McKeown, K. (2004). Detection of question-answer pairs in email conversations. In *COLING '04*. ACL.

Spinellis, D. and Szyperski, C. (2004). How is open source affecting software development? *IEEE Software*, 21(1):28–33.

Thompson, P., Iqbal, S., McNaught, J., and Ananiadou, S. (2009). Construction of an annotated corpus to support biomedical information extraction. *BMC Bioinformatics*, 10(1):349.

Wang, L., Lui, M., Kim, S. N., Nivre, J., and Baldwin, T. (2011). Predicting thread discourse structure over technical web forums. In *EMNLP '11*. ACL.

Weimer, M., Gurevych, I., and Mühlhäuser, M. (2007). Automatically assessing the post quality in online discussions on software. In *ACL '07*. ACL.