2016 IEEE/ACM 13th Working Conference on Mining Software Repositories

Externalization of Software Behavior by the Mining of Norms

Daniel Avery
University of Wollongong
Australia
da488@uow.edu.au

Hoa Khanh Dam
University of Wollongong
Australia
Hoa@uow.edu.au

Bastin Tony Roy Savarimuthu
University of Otago
New Zealand
Tonyr@infoscience.otago.ac.nz

Aditya Ghose
University of Wollongong
Australia
Aditya@uow.edu.au

ABSTRACT

Open Source Software Development (OSSD) often suffers from conflicting views and actions due to the perceived flat and open ecology of an open source community. This often manifests itself as a lack of codified knowledge that is easily accessible for community members. How decisions are made and expectations of a software system are often described in detail through the many forms of social communications that take place within a community. These social interactions form norms which are influential in dictating what behaviors are expected in a community and of the system. In this paper, we provide a tool which mines these social interactions (in the form of bug reports) and extract norms of the system, externalizing this information into a codified form that allows others within the community to be aware of without having witnessed the social interactions.

1. INTRODUCTION

Norms are rules or standards that govern communities and societies [5]. Norms develop over time through social interactions between individuals in societies, and also through reactions and perceptions to others' actions. The large majority of norms take the form of tacit knowledge, propagating from one actor to another through socialization and witnessing the sanction of a norm being applied. Norms dictate what behavior is desired, prohibited or expected within the community, often applying a sanction on those who do not abide by them.

Open Source Software Development (OSSD) communities can be viewed as socio-technical systems [33]. OSSD allows developers to integrate with non-technical members to form a broader, more transparent community. Like all social systems, they too, are governed by norms [35]. However, due to the decentralized control and egalitarian nature of open source communities, norms develop more organically, appearing slowly over time and are a product of social interactions within the community [11].

Problems can arise during development when an actor unknowingly violates the expectations of the system (such as implementing undesired functionality to the system) or the expectation of a development procedure or practice (such as not commenting code). This can lead to inconsistency within the system and propagation of bad practices [42] to those who have witnessed the violation and perceived it as correct. Yet decisions are made at a rapid pace through the various means of communication (such as mailing lists, code comments and bug reports). Hence, it is difficult for an actor to remain educated to a complete set of expectations [23]. These expectations take the form of norms [35].

In this work, we propose a tool to remedy this situation called Norms Miner. Our tool mines textual communications within a community, and extracts the norm placed on an actor or system. The norm is also classified in regards to its deontic context: obligation, prohibition, or requesting for a norm that does not currently exist. In doing so, Norms Miner effectively codifies expectations placed on the system and community through social interactions, and makes available these expectations to those who did not witness the interaction take place.

In OSSD, bug trackers or bug databases such as Bugzilla are used to discuss and track defects of the project. This makes bug reports an ideal data source to discover norms due to the social nature of a bug report; in terms of reporting a defective behavior for peer review (i.e. a behavior that is violating an expectation and therefore should be reported). Additionally, due to the corrective intent of a bug report; describing behavior that is undesirable and also corrective behavior to remedy the bug, further increases the effectiveness of bug reports as a data source. OSSD bug trackers can also be used to report, discuss and track feature requests or patch/policy discussion [20]. This can be interpreted as desired expectations that currently do not exist (i.e. future norms).

While norms of the community can be extracted, in our results we find a large majority of the extracted norms are norms of the system (for reasons discussed in Section 6). While both forms are treated as norms, community norms

MSR'16, May 14-15, 2016, Austin, TX, USA

DOI: http://dx.doi.org/10.1145/2901739.2901744

govern how the community should interact with one another. System norms govern how the system should interact with actors, akin to a system requirement. Viewing these expectation of the system as norms allows to view these crowd source system norms under a social theory context.

There are several advantages of an automated tool that is able to capture implicit norms of an OSSD community by continually mining both past and incoming social interactions between community members. First, as discussed in Krogh et al. [42], joining an OSSD can be a long and difficult process partially due to the challenge of becoming informed and aware of the communities and systems practices, procedures and behaviors without any formal means to do so. By extracting and presenting norms to prospective members, joining a community can become more accessible and less challenging. Tacit knowledge of the communities behavior can be more easily transferred to the prospective member through externalization provided by Norms Miner, which would otherwise have had to be sought out and interpreted manually. This allows for a more agile turnover of community members.

The classification of extracted norms can also provide statistical information that can be used to estimate the health of the community and system. For example, a recent surge in norms extracted as a violation of a prohibition can indicate that the current work is being done incorrectly. A high number of norms extracted as a violation of an obligation can indicate that current work is not meeting the desired effect. Similarly, a high number in future obligations can indicate current work is not meeting stakeholder needs.

Lastly, externalizing norms in a continuous style to community members can increase their agility, responsiveness, consistency and awareness due to the conversion of the tacit knowledge to explicit, codified knowledge. Norms will no longer rely on socialization as their primary means of propagation [11] but instead can be broadcast to the entire community.

The contributions of this paper are:

- An automated tool that can be used for mining natural language contained in textual social interactions of a bug report to discover, extract and classify norms.
- A taxonomy for classifying norms found in bug reports under deontic modality and a technique for classification.
- Evaluations over the tool, reporting a recall of 0.74, precision of 0.73 and f-measure of 0.73 in classifying norms.

The paper is organized as follows. Section 2 discusses the background and related work. Section 3 presents the running example, which is referred to throughout the paper. Section 4 describes the approach used for extracting norms. Section 5 presents the evaluations of our framework. Section 6 details the discussion of results, presenting our interpretations and possible impacts. Section 7 presents future work and concludes the work presented.

2. BACKGROUND AND RELATED WORK

Norms are expected behaviors an individual should adhere to, when interacting within a society or community. Norms are fundamental to understanding the social structure of societies and communities [38]. Norms are an active area of research in a variety of fields, including sociology [13], law [12], economics [27] and computer science [34] [43]. Due to this wide field of applicability that norms are subjected to, a multitude of definitions exist for norms. Habermas [18] defines a norm as *"fulfilling a generalized expectation of behavior"*. This definition of a norm is widely accepted in research, thus, we apply to this definition in our work.

In the field of law and contracts, extraction is often derived from a semantic understanding of the language used in the document to detect normative language; often utilizing *(Natural Language Processing* (NLP) and *Information Retrieval* (IR) techniques to identify and extract the norm [21] [15]. Work in the field of agents detect norms by observing the social interactions of agents [2] [32]. By monitoring how an agent responds to another agents behavior, inferences can be made about the norms governing the behavior observed. For example; an agent $a_1 \in A$ exhibits behavior b_1 and soon afterwards $a_n \exists A$ of surrounding agents start exhibiting a behavior b_2 that negatively impacts the utility of a_1. We can infer that b_1 was a violation of a norm and that the surrounding agents a_n enforced a sanction b_2 unto a_1 due to this violation. With adequate sensors and historical records, multiple occurrences of this inference can provide evidence that $\not b_1$ is a norm and that b_2 is associated and not an outlier.

In our work, we prescribe to the first technique, relying on semantic understanding for extraction rather than social observances. This is due to the type of data available such as bug reports that can be semantically modeled albeit not as structured as is the case in the Law and Contract domains.

There have been several categorizations of norms proposed by researchers (cf.[31]). We believe that deontic norms - the norms describing prohibitions, obligations and permissions studied by the NorMAS community [26] is an appropriate categorization for norms that may be present in OSSD communities.

- *Obligation norms* are behaviors or actions that community members or the system are expected to be performed or abided. Failing to meet this expectation, may incur a sanction against the offender. For example, members in OSSD are often expected to follow a coding convention, failure to adhere to this obligation may result in the violating code being rejected by the community.
- *Prohibition norms* are behaviors or actions that have been deemed unacceptable or should be avoided; they should not be performed within the community. However, when these actions are performed, a sanction may be applied to the offending community member or system. A system storing passwords in plain text is an example of a prohibition norm.
- *Permission norms* describe the permissions provided to the system or community members (e.g. actions they can perform). For example, a user playing the role of the project manager is permitted to create code branches or forks.

A norm extracted as an obligation will be received and acted on entirely differently as opposed to the same norm being extracted as a prohibition. For example, the norm *"use only capital letters when writing bug reports"* does not

entail if community members should abide by this behavior or avoid exhibiting it. By applying a deontic classification over the norm, in this case a prohibition, it becomes clear that the community should avoid exhibiting this behavior, making the norm a prohibition norm.

Social structure, such as norms, are regarded as a major contributing factor to the challenges faced by a prospective developer interested in joining an OSSD project. Ducheneaut [11] analyzes the process of joining an OSSD, commenting on how a prospective developer utilizes bug reporting, documenting and patching to gain exposure to the communities social structure, before using this exposure to assimilate into the community. This is further confirmed in work by Krogh et al. [42] who provide a case study of joining an OSSD community.

In previous work by Savarimuthu and Dam [33] the gap existing in mining norms from OSSD repositories is identified. They explore the motivations, potential issues and impacts surrounding this area of research while also presenting a high level agent based architecture for mining norms in OSSD repositories. Considering the current challenges of big data, they conclude that a systematic approach to mining and understanding norms will assist in bridging the gap between social research and computer science, providing deeper insights into big data. Thus, the work presented in this paper is a continuation of the preliminary work conducted, adopting the motivations and discussions presented previously.

This work is extended in Dam et al. [8] where empirical evidence is shown of the life cycle of a norm in an OSSD project while also tracking norm compliance, noting the large discrepancy between adopted conventions and what has actually been carried out within the projects development creating an alignment problem. This work motivates the need for a data driven method of capturing what actors are doing in practicality (via extraction of norms) to validate or assess against what they are formally expected to be doing (via official documentation).

Breaux et al. [4] utilizes normative phrases in regulation texts to provide alignment of requirements to regulations in health care systems. By exacting rights and obligations from healthcare regulations text, they are able to validate that regulations have been fulfilled by requirements while also handling exceptions and identifying ambiguity. This work highlights how norms discovered from background information can be used as requirements in software engineering. Applying this concept, Norms Miner could be extended to not just present norms of the system but also to feed these norms back into requirements, facilitating support for regulation validation and requirement verification.

Sorbo et al. [37] present a tool for classifying the intent of sentences within software development mailing lists. By training ad-hoc heuristics for capturing typed dependency patterns over a pool of sample communications, they are able to effectively classify the intent of sentences into one of five categories such as "feature request" or "solution proposal" with a precision of 0.90% and a recall of 0.70%. While intent is closely related to expectations; a main contribution in our work is the identification and extraction of expectations (norms), and not solely classification.

Our approach to norm extraction is most similar to Gao and Singh [15] who detail a framework for extracting normative relationships from business contracts. Extracting and classifying norms from natural language, normative relations take the form of commitments, prohibitions, authorizations, powers and sanctions, where a subject applies a consequence over an object with an optional antecedence acting as a guard. While sharing the same overall general approach, our work differs in intent; mining norms from software repositories as opposed to normative relations between actors. Our work considers noisy and ill-structured bug reports while theirs is on well-structured documents.

Bug Id: 39934[1]
Title: Ant copy with filters is not regenerating new target files
Description: If the <filtersfile file="" /> is changed, ant should regenerate new copies. My copy filter target below:

```
<!– Copy config files to the temp directory, replacing tokens as required –>
<copy todir="${pkg.config.dir}">
    <fileset dir="${pkg.source.dir}"/>
    <filterset>
        <filtersfile file=
        "${env.props.dir}/${target.env}.properties"/>
    </filterset>
</copy>
```

Only with the overwrite="true" on the <copy /> task will force a regeneration of the files. I am not sure this is the intention of the overwrite attribute.

Figure 1: The initial bug report of the running example

3. RUNNING EXAMPLE

Throughout this paper, an example is utilized to track how a norm is extracted from a sample bug report (seen in Fig. 1). The example will follow the reporting of an observation made by an actor; in this case, an incorrect behavior of the system, and how such a report can be used to extract information about an expectation of the system in the form of a norm.

Through human interpretation, it can be seen that the sentence "*If the <filtersfile file="" /> is changed, ant should regenerate new copies*" describes expected behavior from Ant. Specifically, the actor *ant*, is obligated to perform the action *generate* to the item *new copies*. This is guarded by the condition *If the <filtersfile file="" /> is changed:*, which must be true for this behavior to be an obligation of the system.

Extracted as a subject-predicate-object-antecedent quadruplet; "ant" is the subject, "generate" is the predicate, "new copies" is the object and "If the <filtersfile file="" /> is changed": is the antecedent. The norm extracted is a current obligation the system has to its users.

4. APPROACH

The approach consists of five steps as shown in the Norm Miner framework presented in Figure 2. These are 1) Pre-Processing of bug reports, 2) Identifying a candidate sentence for norm extraction from a bug report, 3) Applying the feature vector to the candidate sentence, 4) Extraction of the norm from the candidate sentence and 5) Classifica-

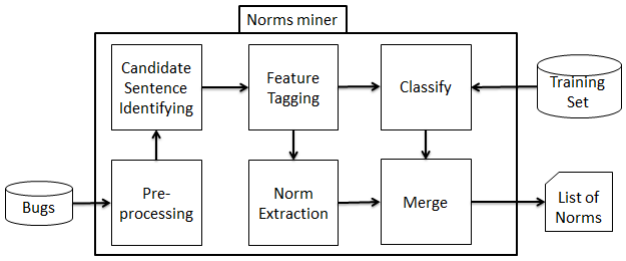

Figure 2: An overview of the framework

tion of the extracted norm. Each stage is elaborated in more detail in the following sub-sections.

4.1 Pre-processing

The first step involves inputting bug reports and preparing the bug report for natural language processing. The majority of preprocessing is done through Stanford's Natural Language Processing tool [25], which was used to tokenise a sentence into words, identify part-of-speech (POS) tag, split sentences and parse the title and description fields. XML fields are also purged from the report.

4.2 Identifying the Candidate Sentence

Bug reports generally only report on a single violation or a single feature request. This is due to the commonly adopted bug reporting principle of "One bug per report" as seen in Bugzilla bug writing guidelines[1]. Accordingly, we consider bug reports to have, at most, a single norm inclusive of this violation or request. Norms are considered to exist at the sentence level [15] for extraction purposes. As a bug report can consist of multiple sentences, a challenge is faced in identifying which sentence is most inclusive of the intent of the bug report and also postulates a currently unknown norm. To overcome this challenge, a set of heuristics are used to identify which sentence is used for the extraction of a potential norm. We refer to this sentence as the candidate sentence.

Due to variations in culture, traditions and norms of each community [7], the reporting of a bug is affected by these variations. For example, one community may strictly adhere to a reporting template while another community may heavily use specific jargons. This makes generalizing heuristics to identify the candidate sentence difficult. Thus, heuristics are created *ad-hoc* through manual observations using grounded theory [17] on a sample set (training set) of the community being mined. Guiding this process are three categories that are essential for the identification of the candidate sentence. These categories are listed with an example of the heuristics used on the Ant data set.

1. Expressiveness heuristics - they help make sure that the candidate sentence is expressive of the bug reports intent

   e_1 Sentences that are lexically similar to the bug reports title are preferred.

   e_2 Longer sentences are preferred over shorter sentences.

   e_3 Sentences that occur earlier in the bug report are preferred to those that are towards the bug reports end.

2. Positivity heuristics - they help in positively identifying that a candidate sentence contains a norm

   e_4 Sentences with modal verbs[2] are preferred. Examples of modal verbs are must, should and can.

   e_5 Sentences with domain specific verbs that imply expectation and are not modal verbs are preferred. These words include get, produces and run.

3. Exclusiveness heuristics - they help in removing sentences that are not relevant

   e_6 Sentences that start with a "negative starter" are less preferred.

   e_7 Sentences that contain words that imply how to replicate the bug or how to fix the bug are less preferred. These words include replicate, example and fix.

   e_8 Sentences that contain evidence of code, stack traces, XML, etc. are less preferred.

Table 1: The candidate sentence of the running example

Sentence Number: 1
Sentence: If the <filtersfile file="" /> is changed, ant should regenerate new copies.

e_1	The sentence has a high cosine score with the title.
e_2	The sentence is considered short.
e_3	The sentence occurs earlier.
e_4	The sentence contains the modal verb "should".
e_6	the sentence contains the negative starter "if".

These heuristics are applied to every sentence within the bug report in order to derive a weighted value for each sentence. The highest weighed sentence is selected as the candidate sentence. Table 1 shows the selected candidate sentence for the running example (Section 3). Even though various heuristics such as e_2 and e_6 weigh against the sentence as a candidate sentence, other heuristics, in this case e_1, e_3 and e_4, help to overcome this deficit to make the sentence the highest weighted sentence in the bug report, thus making it the candidate sentence. It is important to note that the candidate sentence does not yet imply a norm. Rather, it potentially contains a norm. The formula used to derive a sentences weight is available on-line[3].

Heuristics such as e_1, e_2 and e_3 aim to ensure the candidate sentence is expressive of the bug report's intent. The similarity measure in e_1 is determined by a cosine [30] value between the sentence and the title of the bug report. Observations made over the data indicate the title of a bug report is usually highly descriptive of a bug reports intent. Thus, lexically similar sentences are also highly descriptive of a bug reports intent. e_2 favors sentences of at least 90 characters or more in length. This is to ensure that the sentence can sufficiently describe a normative behavior and to avoid e_1 heavily preferring short similar sentences when

[1]https://landfill.bugzilla.org/bugzilla-4.4-branch/page.cgi?id=bug-writing.html

[2]Modal verbs express modality of a governing verb.

[3]http://www.uow.edu.au/~da488/data.html

the title of the bug report is also short and possibly uses generic words. A bug report's intent is best represented in the first half of the bug report, as the report often describes the problem before detailing how to replicate or pose potential remedies in the latter half of the report. Accordingly, e_3 prefers sentences that occur in the first half of the bug report.

Heuristics such as e_4 and e_5 aim to ensure the candidate sentence contains a norm. Modal verbs are used to express modality of a sentence (i.e. should, must, shall), entailing the deontic nature of a norm [6]. This is indicative of the sentence containing normative behavior. Modal verbs are detected through the use of Stanford NLP Part-of-Speech (POS) tagger. e_5 aims to capture sentences that indicate they fulfill some expectation (in this way e_5 is similar to e_4) that was not expressed using modality. For example, the sentence "calling foo() gets me the input" contains no modal verb yet describes an expectation of what should happen when foo() is called. Detection of sentences that contain these words is achieved via regular expressions.

Heuristics such as e_6, e_7 and e_8 aim to remove sentences that could potentially be a candidate sentence but are contextualized or phrased in such a way that undermines the validity of the potential norm. As stated previously, the norm we ideally wish to extract is inclusive of the bug report's intent expressed as an expectation. Thus, potential norms that exist in code fragments e_8 should be disfavored. Sentences that do not pertain to describing the bug report's intent, such as sentences that explain how to fix or replicate a bug e_7, should also be disfavored as we aim to capture what was violated (the intent of the report). Similarly to e_7, e_6 aims to disfavor a sentence that does not describe the bug report's intent by detecting what we refer to as "negative starters". Keywords at the start of a sentence can contextualize the remainder of the sentence. For example, by using the word *"can"* at the start of the sentence, implies that the sentence is posing a question rather than a statement. Norms detected in such a sentence risk their validity as they tend to be a question rather than a declaration. Examples of negative starters include "but", "how" and "can". All three heuristics are detected by regular expressions.

4.3 Applying the Feature Vector

The next step involves applying the candidate sentence with features that are used during the classification process. All but one feature are applied to a candidate sentence by identifying words or patterns in the sentence that share a strong semantic relationship with the feature. For example; the words *must*, *should* and *need* all imply a notion of a strong[4] expectation, and thus are grouped under such a feature. These features are binary and are applied as either true for an occurrence of the feature or false for no occurrence having been detected.

Similarly to how the previous heuristics were derived, words and patterns are assigned to features through a process of manual observation and grounded theory [17] across. However, due to these features acting as a semantic source coding, they offer good generalization across different open source communities. Detection of these words and patterns is achieved via regular expressions. Upon detection of a word or pattern, the owning feature is applied to the sentence.

[4]Strong expectation refers to an expectation more strictly committed to than a weak expectation

Table 2: List of features and sample words

Feature	Example
Tense	NLP extracted
Negation	not, doesn't, won't
Strong expectation	must, need, should
Weak expectation	could, would, might
Desire	good, better, convenient
Reported behavior	gets, returns, generates
Faults	hanged, crashed, exception
Missing	ignore, no longer, absent
Unexpected	however, strangely, rather
Modification	add, change, update
Support	provide, allow, improve

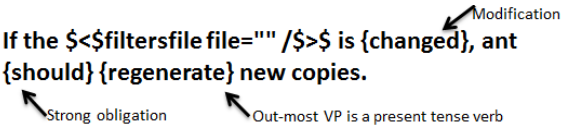

Figure 3: Tag annotation of candidate sentence

Features were selected with the intention to capture semantic distinctions that can clearly separate the meaning of a sentence into a single classification (detailed below). For instance, the feature desire is decisive in separating an obligated behavior from a prohibited behavior, and vice-versa for the feature faults.

The tense feature differs from this process in that it uses a natural language parser to detect the outermost verb phase before applying Part-of-Speech (POS) tags over the phrase to determine the primary tense of a sentence. The tense feature is recorded as either past/present or future tense. A complete list of features as-well as sample words used to detected them are listed in Table 2.

In the case of the running example (Section 3), the candidate sentence contains the following features: past/present for the tense feature, and a true value for modification and strong expectation as seen in Fig. 3.

4.4 Classification

Bug trackers are commonly used for requesting new requirements [20], accordingly, norm categories are split temporally, describing current violations of norms (norms in the system) and also requested norms (norms that are proposed to be added in the future). As a bug report describes a violation of correct behavior of the system, we consider norms extracted in the past and present tense as a violation of a norm. Following the deontic classification of norms described in Section 2, the following taxonomy for classification is proposed:

VO *Violation of an existing obligation* - These are norms that have been reported for not meeting expected behaviors, practices, principles and functionality. For example; a build engine unable to read a build script is considered to be a violation of an obligation.

VP *Violation of an existing prohibition* - These are norms that have been reported as they perform what has been deemed unacceptable within the community. An example of a VP is code causing a null pointer error.

Table 3: The norm extracted from the motivating example

Bug Id: 39934
Sentence: If the <filtersfile file="" /> is changed, ant should regenerate new copies.

Classification:	Violation of an obligation
Subject:	ant
Predicate:	should regenerate
Object:	new copies
Antecedent:	If the < *f*iltersfile file="" /> is changed

Table 4: A sample norm extracted

Bug Id: 38458
Sentence: Unfortunately, some code in Task.java assumes that project is not null and can throw NullPointerExceptions.

Classification:	Violation of a prohibition
Subject:	some code in Task.java
Predicate:	can throw
Object:	NullPointerExceptions
Antecedent:	N/A

FO *Future expectation of an obligation* - Are potential obligation norms that currently do not exist but could exist in the future. They are expected behaviors, practices, principles or functionality proposed by community members with the intention of the norm becoming accepted and applied to the community. An example of an FO is a member suggesting a new reporting format for unit tests.

FP *Future expectation of a prohibition* - Are potential prohibition norms that currently do not exist but could exist in the future. They are proposed behaviors, practices, principles or functionality that should be deemed unacceptable or avoided within the community. An example of an FP is suggesting a coding module to be depreciated.

The results from the training process described in Section 5.2. Note that the deontic concept of permission was excluded as a class since bug reports usually contain information on what should (obligation) and should not happen (prohibition). Discussion of what is allowed to happen (permissions) was rarely observed.

Classification occurs as a supervised multi-class statistical classification utilizing the feature vector of the candidate sentence and a sample set trained with grounded truth values to provide a statistical history to the classifier. Classification of a candidate sentence is applied before the norm is extracted due to the extraction process potentially losing contextualized information (as is the case in Table 4) needed to classify the potential norm. If a norm is successfully extracted from a candidate sentence, the sentence's classification is applied to the norm.

The running example (Section 3) was classified as a VO. This is due to containing the past/present feature which strongly correlates to VO and VP, as well as containing the feature of strong expectation which strongly correlates to VO and FO.

4.5 Extraction

Norms are extracted from candidate sentences as a subject-predicate-object[5] triplet with a possible antecedent expressing the pre-condition that must be met in order for the norm extracted to be relevant. This allows extracted norms to be expressed in a constituent word order, building a clear description of a norm as a subject applying a predicate over an object. To this end, typed dependencies provided by the Stanford Parser [9] are used, along with an automated process inspired by Rusu et al. [29] and Gao et al. [15] work, in the extraction of the triplet.

[5]Also known as subject-verb-object

Subject: The subject is located by performing a breadth first search over the typed dependencies of the sentence, searching for a nominal subject relation (nsubj). During this search, various modifiers such as an adverbial clause modifier (advcl) are pruned to ensure the subject remains in the independent clause and not a subordinate or dependent clause. Once a nominal subject relation is found, the dependent of this relationship and all of its dependencies are extracted as the norm's subject.

Predicate: The predicate of the sentence is extracted by conducting a breadth first search for either a modal verb or a domain specific verb (used in e_3) over a constituency-based parse tree. Once found the parent verb phase of the found verb is used as the norm's predicate.

Object: Similar to the extraction of a subject, the extraction of an object is achieved by performing a breadth first search over typed dependency relation, searching for a direct object (dobj) relation while pruning various modifiers such as an adverbial clause modifier (advcl). Once located, the dependent of this relationship and all of its dependencies are extracted as the object of the norm.

Antecedent: The antecedent is extracted by performing a breadth first search for various modifiers, such as an adverbial clause modifier (advcl) that imply a dependent or subordinate clause. The obtained clauses are then checked to ensure it contains conditional keywords such as "if", "unless", "provided", etc. before being extracted as the antecedent of the norm.

The extracted norm from the running example (Section 3) is seen in Table 3. The subject applies an action (the predicate) over an object which can optionally be guarded by an antecedent to form a norm. The classification of the candidate sentence that was used in deriving this norm is then applied to the norm in order to add context to it. In the case of the motivating example which was classified as a VO, it can be seen that the norm extracted is an obligation norm of the system as it is violation was reported as a bug. Similarly, the norm extracted in Table 4 was classified as a VP, thus the norm extracted is a prohibition norm of the system.

5. RESULTS AND EVALUATIONS

In evaluating the approach used, the following research questions are investigated.

R_1 *Can text from social interactions within a community be mined for norms about the systems expected behavior?* This research question aims to evaluate the core contribution of this work. Extracting norms of system behavior in an explicit form.

Table 5: The Data set

Project	Reports	Valid Reports	Norms	Sentences
Ant	200	178	162	1735
Eclipse	100	94	64	814
Total	*300*	*272*	*226*	*2549*

R_2 *Can mined norms be classified effectively into deontic concepts?*
This research question aims to explore if a deontic classification of extracted norms is reasonable, effective and applicable.

R_3 *Can rules and heuristics used in mining norms from one community be generalized for mining Norms within another?*
This research question aims to explore if data trained on from one community can be generalized or is applicable to another.

5.1 The Data

For these experiments, 200 bug reports that had the closed status were randomly selected from Apache Ant bug database[6] and 100 from Eclipse Platform[7]. Reports that were considered spam[8] or contained purely code/logs/exception message[9] were removed, leaving a total of 178 bug reports for Apache Ant and 94 for Eclipse Platform.

While at most, only a single extraction occurs per report, the miner is tasked with assessing every sentence when selecting one for extraction, thus 1735 sentences are mined for Apache Ant and 814 for Eclipse Platform.

5.2 Experiment Design

Three sets of reports were then manually examined and annotated. One set by the authors, referred to as the *development set* and two sets by two independent evaluators (one person per set), refereed to as the *truth set*. In an effort to remove bias, the independent evaluators were external to this work and have not been involved in its development. Evaluators worked in isolation from each other during annotation and were only assisted by the authors in questions regarding to language and grammar.

Annotating a report consists of examining every sentence within a bug reports description, identifying if it contains a norm and if so what class does it belong to (as outlined in Section 4.4) or an N/A classification if the norm did not fit into any possible class. Lastly, each sentence identified to contain a norm was ranked in terms of its relevance. Relevance was defined as "a sentence that is more descriptive of the intent of the report". The interpretation of the intent of a report was decided by the evaluator. In order to maintain consistency among the evaluators, sentences were determined and split by Stanford NLP "WordToSentence" annotator with the setting of property "newlineIsSentenceBreak" property set to two. Treating the occurrence of two consecutive newlines as a sentence break. as-well as its normal sentence splitting.

Once annotation was complete, only the highest ranked sentence of each report along with the classification is used in constructing a combined truth set from each evaluator's *truth set*. Inconstancies in sentence rankings or sentence classification was resolved by a unanimous decision by both evaluators after discussing amongst each other. Once both sets are consistent they form the ground truth set used in evaluating the various stages of the miner. The final outcome was 162 norms extracted from Apache Ant and 64 norms extracted from Eclipse Platform as detailed in Table 5.

In answering R_1 we identify three critical areas that must be evaluated to determined the effectiveness of mining norms from social text. These are: 1) identifying the candidate sentence, 2) classification of norms, and 3) extraction of norms. Furthermore, two independent communities are used to assess the robustness and resilience of the miner. In answering R_2, statistical results from evaluation of the classification step will be reported on. In answering R_3, heuristics made for and data trained on the Apache Ant project, will be utilized in testing on the Eclipse Platform set, breaking down what is and is not effective.

5.3 Identifying the Candidate Sentence

In evaluating the effectiveness of the candidate sentence selection step for R_1, we use traditional relevance measures with the ground truth set serving as truth values. The heuristics for identifying the candidate sentence for Apache Ant project (presented in Section 4.2) was developed and refined from examining 20 randomly selected reports from the *development set* of Ant. The identical reports were also removed from the ground truth set. The remaining 158 reports containing 1561 sentences for Apache Ant were used in evaluation. As only one candidate sentence is selected per report, upon a selection (a positive prediction), all remaining sentences are treated as a false prediction. Due to this precept, type one and type two errors are treated as identical. For Apache ant, the heuristics were able to identify the candidate sentence with a precision, recall and F-score of 0.83.

In assessing if heuristics that were tuned to Apache Ant is generalizable to Eclipse Platform as per R_3, the heuristics for Apache Ant was applied to Eclipse Platform in order to identify the candidate sentence. The results of a precision, recall and F-score of 0.45 clearly indicate that such a generalization was not possible due to the heterogeneity of reporting styles between OSSD communities.

5.4 Classification

To evaluate the classification step of R_2, ten-fold cross-validation of the ground truth set with three different classification techniques were used. These techniques are Naive Bayes classifier [14], Support Vector Machine (SVM) [3] and C4.5 [28] and are commonly applied in classifying text within a bug report [1, 22, 40]. The Weka tool suit [19] is used, along with the ground truth set to classify the candidate sentence. As features are based in semantics, they do not need to be specifically tuned to a project, unlike the heuristics for selecting a candidate sentence. However, features were created and refined, based on performance measures using the Apache Ant *development set* as the truth set.

[6]http://ant.apache.org/bugs.html
[7]https://bugs.eclipse.org/bugs/
[8]https://issues.apache.org/bugzilla/show_bug.cgi?id=29960
[9]https://issues.apache.org/bugzilla/show_bug.cgi?id=53291

Table 6: Weighted averages of the three classifiers

Classifier	Recall	Precision	F-score	ROC
Apache Ant				
Naive Bayes	0.742	0.733	0.736	0.859
SVM	0.708	0.705	0.703	0.775
C4.5	0.680	0.674	0.674	0.785
Eclipse Platform				
Naive Bayes	0.436	0.414	0.390	0.635
SVM	0.362	0.207	0.248	0.489
C4.5	0.415	0.379	0.386	0.534

Table 7: Detailed results of Naive Bayes classification

Class	Freq.	Recall	Precision	F-score	ROC
Apache Ant					
VO	50	0.647	0.717	0.680	0.825
VP	69	0.797	0.724	0.759	0.851
FO	56	0.786	0.786	0.786	0.913
FP	2	0	0	0	0.489
Avg.		0.742	0.733	0.736	0.859
Eclipse Platform					
VO	30	0.167	0.417	0.238	0.563
VP	37	0.757	0.438	0.554	0.683
FO	23	0.348	0.444	0.390	0.722
FP	4	0	0	0	0.233
Avg.		0.436	0.414	0.390	0.635

As features were constructed and refined from a process of grounded theory used on bug reports from Apache Ant (including reports not used in the evaluated data set), we evaluate the classification step by utilizing an independent project (Eclipse Platform), that was not utilized in the formation of features. This is to infer, if features and classes are generalizable to other projects as per R_3.

We note that the evaluation of this step is conducted, after the candidate sentence has been selected by the miner and not in isolation (i.e. only correct candidate sentences are classified), as R_1 aims to evaluate the process in its entity and not in separate components. Thus, an incorrectly selected candidate sentence, by the tool will be treated as a false positive for any classification other than *N/A*.

The results of each technique can be seen in Table 6. Naive Bayes moderately outperformed both competing techniques in recall, precision and F-measure. Notably, the ROC area for Naive Bayes is significantly higher, at 0.859 for Apache Ant and 0.635 for Eclipse Platform, indicating that it is a far more reliable model than the others, assuming correct and incorrect classification share equal cost across all classes.

Table 7 shows a more detailed look, at the results from classifying both Apache Ant and Eclipse Platform with Naive Bayes. Note that FP's are severely under-represented within ground truth set and thus, perform very poorly during classification. The very small representation of FP's conforms, with the natural assumption that a feature request details, what an actor wishes to include and not make a request for exclusion. FO's performed exceedingly well, upon closer examination. This is partially due to the tense feature, which accurately labels the correct tense for 147 of the 178 norms for Apache Ant. VO's performed below par, due to being semi-frequently mis-classified as a VP. This indicates features that correlate strongly towards VO's are underrepresented in how they are applied, to candidate sentences or that features that correlate strong to VP's are too excessively applied to candidate sentences.

Table 8: Results from the extraction evaluation

Project	Precision	Recall	F-score
Ant	0.76	0.15	0.26
Eclipse	0.90	0.14	0.25

The recall score of VO's were relatively poorer compared to other classes such as VP's across all evaluations. Scoring 0.647 compared to VP's recall value of 0.797 in the Apache Ant evaluation and significantly poorer at 0.167 compared to VP's recall of 0.757 in the Eclipse Platform evaluation. However, with a ROC area of 0.635, this indicates that results presented are more reliable then selecting at random. While it may seem appropriate, to infer that the classification step suffers from poor generalization (in regards to R_3), on closer exception, the primary reason for mis-classification was due to the poor result of the candidate sentence selection step as only 29 of the 64 norms was correctly identified. However, 21 of these 29 norms were correctly classified, representing a precision of 0.72.

In order to set a benchmark of what can be deemed as "effective" in R_2, we refer to Gao and Singh [15] to compare and contrast Norms Miner's classification results, to ensure competitiveness at a state of the art level. As discussed in Section 2, Gao and Singh propose a framework for extracting normative relations, from business contracts that shares a relatively similar approach to what is used in this paper. In evaluating their approach they reported a recall of 0.74, a precision of 0.75 and an F-measure of 0.74 on a ten-fold cross-validation of their trained data set of 868 classified sentences[10] in contrast to our Naive Bayes results of 0.74 recall, 0.73 precision and an f-measure of 0.74 for Apache Ant. When comparing the two approaches, it is important to note that the input of each approach are vastly different in terms of language used due to the different domains. The data source used in their work are business contracts that are well-written in plain English, and contain well-formed deontic statements. In their approach they extract each sentence and identify whether it contains a norm or not. In our work, the description of the bug can contain solecistic statements, programming code, slangs etc. Also, we were tasked with identifying the most important sentence in a given bug report. Theirs does not have this issue. Considering these facts, we believe our framework provides at least comparable performance to theirs.

5.5 Extraction

In evaluating the final step of the miner for R_1, we compare the final extracted norm with its classification against the ground truth set and a survey completed by the two independent evaluators. The ground truth set is used to determine if the correct candidate sentence was used for extraction and that the correct classification was assigned. As the extraction may be phrased in several different ways it could not be validated by the ground truth set, instead a survey was conducted asking the two evaluators if the extraction

[10]In evaluating a smaller but independent data set, they reported higher scores of 0.83, 0.86 and 0.84 in recall, precision and f-measure respectively.

presented by the miner is a suitable summary of the norm they identified in the ground truth set, with a unanimous decision required. For a positive prediction, the extraction process must output a *subject-predicate-object*. Accordingly if this triplet cannot be extracted, a negative prediction is made.

Thus, if the correct candidate sentence was used, with the correct classification and a unanimous positive decision of the extraction, the extraction is regarded as a true positive. A false positive is recorded when the extractor makes a positive prediction but either the wrong candidate sentence was used, the wrong classification assigned or the extraction was rejected by the evaluators. A true negative is recorded when the extractor makes a negative prediction and the report was founded to not contain a norm discerned from the ground truth set. Lastly, a false negative is recorded when the extractor provided a negative prediction but the report was found to contain a norm.

From the 178 reports in Apache Ant, 33 positive predictions were made, of which 25 were confirmed true. This represents a precision of 0.76, and recall of 0.15 and an f-score of 0.26. From the 94 reports in Eclipse Platform, 10 positive predictions were made, of which 9 were confirmed to be true. This represents a precision of 0.90, and recall of 0.14 and an f-score of 0.25. These results can be seen in Table 8.

Results show that the precision of this step remains at a similar level to the previous two steps, however, the recall substantially drops, resulting in a much lower f-score. This results in the final output of the miner providing a high level of certainty in the norms it presents but only encompasses a small subset of norms that have been identified to exist.

In retrospect, a positive or negative prediction is directly dependent on if a *subject-predicate-object* triplet can be found and does not implicate if extraction is normative or the correct norm. This leads to the conclusion that the lower recall relative to the previous two steps can be attributed to the solecism, in the input and the method used to train the part-of-speech and typed dependencies. As by default, the NLP tool suite used is trained over a corpus of newspapers articles, often undergoing strict editing to ensure proper language and grammar is used while also limiting the use of technical language and jargon. The opposite is true in regards to bug reports.

6. DISCUSSION

In line with motivations of this work discussed in Section 1, we present a sample set of norms extracted from the data set used in our evaluation to highlight and postulate possible usages of such a miner. We note that due to the small input of reports used and the relatively low recall of the extractor that only 34 norms are analyzed. However, even with such a small sample set, certain inferences can be made.

For example, the miner demonstrates how it can be used to effectively mine future obligations. By externalizing this information, core developers can quickly be notified about the current needs of the software, that is currently missing in a highly summarized form (as seen in row 1, 2 and 6). This can also be presented to new joining members, who are interested in developing code but do not know where to start. The extractions shows detail at a very low level of granularity, offering specifics in exactly what is currently lacking, giving a clear direction to what needs to be done. For example, row 2 outlines a current attribute, lacking desired functionality and instructs the attribute to be extended to support lists of lists.

FO's can also be used to warn developers of potential changes that might affect their work. As seen in row 7, by capturing future changes, an actor who has code that may be deprecated, when a dependency is updated can be provided warning to resolve issues before the change is committed. This also works retroactively, as the same actor who in this scenario failed to receive the warning and can look at recent FO's, to determine if anything has been changed (by seeing if the parent report is now marked resolved) that may be reason for the now broken code.

With appropriate information retrieval techniques, VO's and VP's can be provided based on the context, of where an actor is currently working. Due to the egalitarian and flat nature of an OSSD community, a strategy of risk acceptance (as opposed to risk avoidance or mitigation) is commonly taken. By providing and contextualizing VO's and VP's (that the report they were discovered in is still open), an actor can make more informed decisions of how to interact with another persons code. For example, row 4 details behavior that under a certain pre-condition, a new jar will always be created. An actor presented with this information can now make more informed decisions such as to avoid using the jar task or provide a default "external manifest file". This can directly assist with slowing or stopping the propagation of bad code in new development.

Lastly, VO's can be used as a method of troubleshooting as seen in row 3 and 8. A user facing an issue but unable to locate a cause (either to fix or avoid it) can query extracted norms with appropriate terms (in this case "tomcat" and "JSP") to discern what the cause of the issue is. This in turn can be used in bug reporting, allowing a reporter to query if the behavior they observe is a (known) violation or an expected behavior.

A result of utilizing bug reports as a data source for norm externalization is that all but a few extracted norms where norms of the system and not of the community. Upon investigation this was determined to be due to the nature of a bug reports content which is to report on the system behavior and not actor behavior. Preliminary results of running Norms Miner on a mailing list repository has shown to capture and extract norms between actors; as actors and their behaviors are included as topics of discussion in such a data source.

Currently, mining big data faces problems in analyzing data from a social perspective [33]. Software development trends over recent years have shown a significant growth in OSSD [10], which has a higher focus on social interaction between contributors then previous traditional means. It is our view that applying social context to current data-driven software engineering techniques is becoming more imperative. By making a software development community more aware of its social influences and effects, greater insights can be gained. These gains can include understanding which normative behavior contributes to the success of a project and which ones do not. Furthermore, by clearly defining and increasing awareness of norms, a community can apply a more pragmatic approach to norm detection and enforcement.

6.1 Threats to Validity

Table 9: A sample set of extracted norms

Row	Bug Id	Extracted Norm	Class
		Apache Ant	
1	16871	Ant's javadoc task should quote the file names correctly when generating the external file list.	FO
2	39456	It would be nice to be able to define a <fileset> as the union of other more granular <fileset>s.	FO
3	28439	Since I'm using Tomcat 5.0, I cannot compile my jsp with Ant.	VO
4	29683	The jar task which contains a nested zipgroupfileset element, and does not specify an external manifest file, will always rebuild the jar file, even if it is up to date.	VP
		Eclipse Platform	
5	427511	some UI elements (generally text) gets painted out of its expected region.	VP
6	411405	the section should also update the toogle's tooltip so that the user also sees the tooltip when he hovers over the toggle hyperlink field.	FO
7	405814	for a new service version of ICU4J, we'll be picking up version 50.1.1 instead of 50.1.0.	FO
8	414142	we disable disableControl(tpl.clientArea, toEnable); and that should disable all typing in the views and editors.	VO

In extracting norms from bug reports, our evaluations are faced with the following threats to validity.

The data set used to evaluate the accuracy of the classifier can be considered relatively small of just 300 bug reports. However, due to the miner examining the entire bug report, a significantly larger number of sentences are analyzed. In total 2549 sentences were examined with 272 valid bug reports. While in traditional papers in this domain greatly exceed this number, we attribute this to exploiting a pre-existing oracle [24, 39, 16] which utilize fields of the report itself as a truth value (such as duplicates, time to fix or who the report was assigned to). Instead we refer to similar works that have no means of a pre-existing oracle and thus, must create one through manual annotation. Gao and Singh [15] utilize 1099 sentences for evaluating their norm extraction in contrast with our 2549 sentences while Sorbo et al. [37] classify 400 messages from mailing lists in their evaluation in contrast with our 300.

Subjectivity of the oracle is also acknowledged as a threat. To mitigate this, all information used in deriving the oracle was constructed from independent sources, that were not familiar with what heuristics where employed, what the features where and were not permitted to see the authors *development set*. Evaluators also must unanimously agree on information, before adding it to the oracle further reducing subjectivity.

In the evaluations of the classification stage, averages of performance measure are used along with an ROC area measure. It is important to acknowledge that by using such measurements, we accept the assumption that the cost of a correct or incorrect classification, is static in cost across all categories. While such an assumption allows an objective evaluation of the classifier, it does not provide an evaluation of its utility. For example, it is justifiable that a community would find a violation of a current obligation more pressing than a proposal for a new obligation. Thus, a classifier that has a lower overall score but weighs more decisive classes higher may be preferred. However, such a weighting scheme would be highly subjective to each individual community, offering poor generalization over any evaluations conducted.

7. CONCLUSION AND FUTURE WORK

In this paper we presented a tool called Norms Miner that automatically extracts and classifies norms from bug reports in open source communities. We presented a set of heuristics for identifying a candidate sentence, within a bug report that is expressive of the reports intent and potentially contains a norm. We described a classification process that classifies a norm within a candidate sentence, into one of four norm types based on deontic norms. We presented an extraction process that extracts a norm, from a candidate sentence in the form of a *subject-predicate-object* triplet with and optional antecedent.

The work presented in this paper, intends to serve as an initial step towards modeling norms in OSSD. It is the authors intention to further explore the issue, further refining Norms Miner into a more comprehensive and robust tool for mining norms. Towards this end, several directions of work can be taken.

Currently, Norms Miner can only extract norms from a bug tracker. By expanding the framework to utilize more data sources such as mailing lists, code comments, commit logs and supporting documentation, a more comprehensive range of norms could be extracted, providing a better coverage of normative behavior across the entire OSSD community.

Due to extracting a norm from a single candidate sentence, the norms relevance is questionable and lacks metadata. While being descriptive of the norm, extracted norms do not currently inform the wider communities conformance of the norm, if its enforced or what sanctions does it entail. Inclusion of this information into a normative model built around extracted norms would further increase the usefulness of our framework. To address this issue we propose to codify extracted norms. This would provide several substantial benefits, such as presenting the same norm encountered in several different places as a single norm, allowing a user to track the conformance of a norm, sanctions applied due to the violation and how it emerged within the community.

Lastly, results from Norms Miner can be utilized for empirical research into how norms influence and effect OSSD. Providing a method for answering questions such as; which norms when violated create high priority bugs or if similar norms correlate to the time a bug report takes to fix.

8. REFERENCES

[1] G. Antoniol, K. Ayari, M. Di Penta, F. Khomh, and Y.-G. Guéhéneuc. Is it a bug or an enhancement? *Proceedings of the 2008 conference of the center for advanced studies on collaborative research meeting of*

minds - CASCON '08, page 304, 2008.

[2] T. Balke, C. Pereira, F. Dignum, E. Lorini, A. Rotolo, W. Vasconcelos, and S. Villata. Norms in MAS : Definitions and Related Concepts. *Normative Multi-Agent Systems*, 4:1–31, 2013.

[3] B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory - COLT '92*, pages 144–152, New York, New York, USA, 1992. ACM Press.

[4] T. D. Breaux, M. W. Vail, and A. I. Anton. Towards regulatory compliance: Extracting rights and obligations to align requirements with regulations. In *Requirements Engineering, 14th IEEE International Conference*, pages 49–58. IEEE, 2006.

[5] R. B. Cialdini and M. R. Trost. Social influence: Social norms, conformity and compliance. *The Handbook of Social Psychology*, 2:151, 1998.

[6] J. Coates. Modal Meaning: The Semantic–Pragmatic Interface. *Journal of Semantics*, 7(1):53–63, 1990.

[7] J. S. Coleman. *Foundations of social theory*. Harvard University Press, 1994.

[8] H. K. Dam, B. T. R. Savarimuthu, D. Avery, and A. Ghose. Mining software repositories for social norms. In *Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference on*, volume 2, pages 627–630. IEEE, 2015.

[9] M.-C. De Marneffe, B. MacCartney, C. D. Manning, and Others. Generating typed dependency parses from phrase structure parses. In *Proceedings of LREC*, volume 6, pages 449–454, 2006.

[10] A. Deshpande and D. Riehle. The total growth of open source. In *Open Source Development, Communities and Quality*, pages 197–209. Springer, 2008.

[11] N. Ducheneaut. Socialization in an Open Source Software Community: A Socio-Technical Analysis. *Computer Supported Cooperative Work (CSCW)*, 14(4):323–368, jul 2005.

[12] R. C. Ellickson. Law and economics discovers social norms. *The Journal of Legal Studies*, 27(S2):537–552, 1998.

[13] J. Elster. *The cement of society: A survey of social order*. Cambridge University Press, 1989.

[14] N. Friedman, D. Geiger, and M. Goldszmidt. Bayesian network classifiers. *Machine learning*, 29(2-3):131–163, 1997.

[15] X. Gao and M. P. Singh. Extracting normative relationships from business contracts. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*, pages 101–108. International Foundation for Autonomous Agents and Multiagent Systems, 2014.

[16] E. Giger, M. Pinzger, and H. Gall. Predicting the fix time of bugs. In *Proceedings of the 2nd International Workshop on Recommendation Systems for Software Engineering*, pages 52–56. ACM, 2010.

[17] B. G. Glaser and A. L. Strauss. *The discovery of grounded theory: Strategies for qualitative research*. Aldine Publishing, New York, 1967.

[18] J. Habermas. *The theory of communicative action*, volume 2. Beacon press, 1989.

[19] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.

[20] K. Herzig, S. Just, and A. Zeller. It's not a bug, it's a feature: How misclassification impacts bug prediction. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 392–401. IEEE Press, 2013.

[21] A. K. Kalia, H. R. Motahari-Nezhad, C. Bartolini, and M. P. Singh. Monitoring commitments in people-driven service engagements. In *Services Computing (SCC), 2013 IEEE International Conference on*, pages 160–167. IEEE, 2013.

[22] P. S. Kochhar, F. Thung, and D. Lo. Automatic fine-grained issue report reclassification. In *Proceedings of the IEEE International Conference on Engineering of Complex Computer Systems, ICECCS*, pages 126–135, 2014.

[23] G. F. Lanzara, M. Morner, and Others. The knowledge ecology of open-source software projects. In *19th EGOS Colloquium, Copenhagen*, 2003.

[24] K. Liu, H. B. K. Tan, and H. Zhang. Has This Bug Been Reported ? In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, pages 82–91, 2012.

[25] C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, and D. McClosky. The {Stanford} {CoreNLP} Natural Language Processing Toolkit. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 55–60, 2014.

[26] J.-J. C. Meyer and R. J. Wieringa. Applications of deontic logic in computer science: A concise overview. *Deontic Logic in Computer Science: Normative System Specification*, pages 17–40, 1993.

[27] D. C. North. *Institutions, institutional change and economic performance*. Cambridge university press, 1990.

[28] J. R. Quinlan. *C4. 5: programs for machine learning*, volume 1. Morgan kaufmann, 1993.

[29] D. Rusu, L. Dali, B. Fortuna, M. Grobelnik, and D. Mladenic. Triplet extraction from sentences. In *Proceedings of the 10th International Multiconference" Information Society-IS*, pages 8–12, 2007.

[30] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24(5):513–523, 1988.

[31] B. T. R. Savarimuthu and S. Cranefield. Norm creation, spreading and emergence: A survey of simulation models of norms in multi-agent systems. *Multiagent and Grid Systems*, 7(1):21–54, 2011.

[32] B. T. R. Savarimuthu, S. Cranefield, M. A. Purvis, and M. K. Purvis. Internal agent architecture for norm identification. In *Coordination, Organizations, Institutions and Norms in Agent Systems V*, pages 241–256. Springer, 2010.

[33] B. T. R. Savarimuthu and H. K. Dam. Towards mining norms in open source software repositories. In *Agents and Data Mining Interaction*, pages 26–39.

Springer, 2014.

[34] Y. Shoham and M. Tennenholtz. Emergent conventions in multi-agent Systems: initial experimental results and observations. *KR-92*, pages 225–231, 1992.

[35] M. P. Singh. Norms as a basis for governing sociotechnical systems. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 5(1):21, 2013.

[36] I. Sommerville and G. Kotonya. *Requirements engineering: processes and techniques*. John Wiley & Sons, Inc., 1998.

[37] A. D. Sorbo, S. Panichella, C. A. Visaggio, M. D. Penta, G. Canfora, and H. C. Gall. Development Emails Content Analyzer: Intention Mining in Developer Discussions. In *30th international conference on Automated Software Engineering (ASE 2015). Lincoln, Nebraska*, 2015.

[38] W. G. Sumner. *Folkways: A study of the sociological importance of usages, manners, customs, mores, and morals*. Ginn, 1906.

[39] C. Sun, D. Lo, S.-C. Khoo, and J. Jiang. Towards more accurate retrieval of duplicate bug reports. *2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*, pages 253–262, nov 2011.

[40] F. Thung, D. Lo, and L. Jiang. Automatic defect categorization. In *Proceedings - Working Conference on Reverse Engineering, WCRE*, number October, pages 205–214, 2012.

[41] R. Tuomela. *The Importance of Us: A Philosophical Study of Basic Social Notions*, volume 108. Stanford University Press, 1995.

[42] G. von Krogh, S. Spaeth, and K. R. Lakhani. Community, joining, and specialization in open source software innovation: a case study. *Research Policy*, 32(7):1217–1241, jul 2003.

[43] A. Walker and M. Wooldridge. Understanding the Emergence of Conventions in Multi-Agent Systems. In *ICMAS*, volume 95, pages 384–389, 1995.