

Text Mining Studies of Software Repository Contents

Bartosz Dobrzyński and Janusz Sosnowski^a

Warsaw University of Technology, Institute of Computer Science, ul. Nowowiejska 15/19, Poland

Keywords: Issue Tracking System, Issues and Comment Classification, Software Repository, Text Mining.

Abstract: Issue tracking systems comprise data which are useful in evaluating or improving software development processes. Revealing and interpreting this information is a challenging problem which needs appropriate algorithms and tools. For this purpose, we use text mining schemes adapted to the specificity of the software repository. They base on a detailed analysis of the used dictionaries which comprise Natural Language Words (NLW) and are enhanced with specialized entities in issue descriptions (e.g., emails, code snippets, technical names). They are defined with specially developed regular expressions. The pre-processed texts are submitted to original text mining algorithms (machine learning). This approach has been verified in commercial and open-source projects and showed possible development improvements.

1 INTRODUCTION


Software development is managed with the use of Issue Tracking Systems (ITS) supported with Software Version Control (SVC) and other systems. They provide data repositories comprising rich data which document software engineering activities during the project life cycle (Vidoni, 2021; Sosnowski et al., 2017; Huang, et al., 2019). They are specified as issues in ITS created by project stakeholders (actors): project analyst, developers, testers, users. Issues are specified in some structural form and include various fields targeted at specific features, e.g., title, summary and description of the relevant problem, issue type, priority, reporter id, status and history of processing, links to other files.

Recently, text mining techniques gained significant interest (e.g., Li et al., 2022; Yahav, et al. 2019). Typical analysis goals are information extraction, data mining and knowledge discovery, text categorization, sentiment analysis, document summarization, etc. For this purpose, various algorithms have been developed based on Natural Language Processing (NLP) which involve information retrieval and extraction, lexical, structural and semantical analysis, data mining and visualization, etc. Unfortunately, texts used in software repositories differ significantly from those analysed in classical text mining.

The developed text mining scheme has been enhanced with a deep insight into the used dictionary considering words from NL thesaurus and other entities. Non-NL entities are identified and classified in relevance to their semantical meaning. They extend the space of data mining as opposed to classical text mining covering only NL words. The contribution of the paper relates two three aspects:

- i) Taxonomy and extensive analysis (statistical and semantical) of textual contents in IST repositories supported with developed regular expressions identifying non-NL entities.
- ii) Investigating the impact of feature selection on the text classification efficiency in machine learning schemes adapted to searched issue properties.
- iii) Assessing the quality of issue documentation in relevance to project stakeholders' activities.

The structure of the paper is as follows. Section 2 outlines the background of our research in relevance to other publications in the literature. Section 3 presents an original analysis of text features in repositories and provides illustrative statistics. Section 4 outlines text mining methodology and algorithms illustrated with some experimental results. Sections 5 discusses possible extensions of our approach which is concluded in section 6.

^a <https://orcid.org/0000-0001-6640-1585>

2 LITERATURE REVIEW

Software development processes are documented in diverse repositories supported by issue tracing (IST), software version control (SVC) and other systems. Various automatization approaches have been proposed in the literature targeted at specified aspects facilitating development optimization.

Umer, Li & Sultan (2019) deal with the problem of approving or rejecting issues for further processing. They use natural language processing techniques for sentiment analysis (positive, negative) of the issue summary based on frequently used words in the reports. Nadem et al. (2021) classified issues in three categories bug, enhancement, and question. The proposed approach uses neural network RoBERTa tuned to issue reports and admits multi-label settings.

In practice, issues are supplemented with diverse textual comments added during the issue handling by project participants. They provide abundant information about the associated issues useful in their processing and monitoring the project progress, activities and competence of project participants, etc. Wentig et al., (2019) propose techniques to acquire interesting information for stakeholders. They traced issues from Github projects containing numerous comments related to discussions involving many participants (on average 10). Hence, they formulated 15 categories of relevant discussions, which included expected and observed behaviour, bug reproduction, solution discussion, task progress, testing, new issues and requests, social conversation, etc. This study gives only some general insight on stakeholder discussions, the level of issue understanding, etc.

Herbold and Trautsch (2020) analyse the possibility of classifying bug and non-bug issues basing on combined analysis of issue title and description with trained models. This problem has been raised due to the observation that the reported issue types often do not match the description of the issue. Ferreira Gomes et al., (2019) provide a comprehensive survey on issue severity prediction. Similarly, we can identify security bugs. Most described methods use unstructured text features, machine learning and text mining techniques. Nagvani & Verma (2012) propose the bug classification algorithm CLUBAS which combines text clustering, frequent term calculation and term mapping techniques. It is used to search similar groups of bugs (with cosine similarity), groups of developers relevant to categorized bug groups (for optimizing fixing times), etc.

Text mining is also useful in resolving some specific problems of issue handling, e.g., related to

bug diagnosis and triaging, identifying duplicated issues. In most projects new issues or bugs are manually triaged by an expert developer. This may result in excessive time costs resulting from inappropriate assignment of a developer to fix the problem and reassigning it to other persons. This can be improved by automatic issue triaging based on text classification techniques (Banerjee et al., 2017) to recommend appropriate developers. It is based on extracting and matching bug and developer categories' expertise scores for correlation with reported bugs. Zhang et al. (2016) combine issue triaging (fixer assignment) with severity identification.

Fan et al., (2018) provide a survey of automated bug report management techniques which include bug triaging, detection of duplicate bug report, bug severity/priority assignment. It is extended with a method of discriminating valid and not valid bugs based on random forest algorithm. Zhang et al., (2019) propose a tool to method-level fine-grained bug localization. It uses semantic similarity, temporal proximity, and call dependency scores.

An important issue is automatic identification of duplicated reports to eliminate redundant actor activities and reducing the amount of time a triager spends in analysing the incoming reports. Depending upon the project, duplicate issues constitute a few up to 30% of all issues. We can use semantic and lexical similarities in checking an incoming report with other existing reports in the repository to assess the probability of being a duplicate. Hindle & Onuczko (2019) survey various deduplication methods, they base on diverse techniques using TF-IDF term frequency, machine learning, topic analysis, or deep learning. Banerjee et al., (2017) assess issue text and summary similarities with the cosine similarity metric. Most approaches to automatic detection of duplicate bugs use natural language processing, only a few (Ebrahimi, et al., 2019) consider also the execution information (stack traces).

The presented literature review confirms practical significance of text mining techniques supporting issue handling in the project lifecycle. The relevant publications are targeted at algorithms tuned to specific problems and they lack detailed studies of semantical and structural features of issue reports and comments. Typically, repositories of many projects comprise a significant percentage of non-natural language terms. Introducing a taxonomy of these terms we developed regular expressions defining diverse categories of terms. Replacing these terms in the texts with tags relevant to different term categories enhances semantic impact of original texts

and provides new dimension of text classification and interpretations. Deriving various issue report features with text mining we can combine them with other statistics and issue handling processes which is also neglected in the literature. These features can be interpreted in a wider context of issue handling dependencies, so the analysis are more project assessment oriented. In the paper we present a deeper study of these problems, which augments the space of assessing issue handling and documentation.

3 TEXT FEATURE ANALYSIS

Having analysed software repositories of many projects we found that they constitute a mixture of NL phrases and sentences and other included objects. Syntactic features of these records are not standard, partially retaining grammatical rules and some specific styles of reports or imposed by the company. In this situation the crucial point is a deeper analysis of the object and word classes which can be considered as dictionary taxonomy. Objects are complex text entities of dedicated meaning and specified structure, e.g. code, panel, test outcome. Within word classes (continuous character string) we distinguish natural language words (NLW) consistent with specified thesaurus (e.g., English), functional words (FW) and non-classified words (NCW). In most cases NLW class relates to English thesaurus, however sometimes reporters may include other language words (e.g., Polish) which can be considered separately or included in NCW class (if appear sporadically). NCW class may comprise words with not defined meaning (can be processed to include them in appropriate FW class or a new one) or erroneous words, e.g. due to typos.

Functional words are project dependant, nevertheless in general we can distinguish some typical ones: references to attachments of different types (e.g. graphical, textual, logs), external links and pull request references, repository user identifiers, names of code class or packet, email address. Recognition of diverse types of objects and words within the considered classes can be performed with the use of the language thesaurus and regular expressions defining diverse objects and FW types. For an illustration we present a list of such expressions (specified in POSIX standard notation):

- Expressions identifying graphical, textual, and log attachments: `\\S+[.png|.jpg|.gif]!;`
`\\S+.txt!` and `\\S+.log!`, respectively (attachment with any extension `\\S+!`).

- Pull request references: `(\\(.*?)pull-request?.*\\)(https:\\\\.*?pull-request[\\S]*)`
`(\\(.*?)pull\\/?.*\\)(https:\\\\.*?pull[\\S]*.?)`
`(https:\\\\.*?commit[\\S]*.?)`
- External references: `(http:\\\\S+)(\\[\\S+\\S+\\])(\\[http.*?\\])(\\[https:\\\\S+\\])(https:\\\\S+)(http:\\\\S+)`
- Panel and code sections: `{panel(.\\n)*?{panel} and ({code(.\\n)*?{code}})({noformat} \\n)*?{noformat}}|({code:java}{.\\n)*?{code}}`
- Names of classes and code packages (including js files): `(([a-zA-Z_][a-zA-Z\\d]*\\.){1,}[a-zA-Z_][a-zA-Z\\d]*|((\\b[a-z][A-Z]+(\\b[a-z][A-Z]+)+.js)| (\\b[a-z][A-Z]+(\\b[a-z][A-Z]+)+\\b)))`
- CamelCase names of classes or methods: `\\w*[A-Z]w*[A-Z]w*;` `\\b[a-z]+[A-Z]+S*\\b`
- Link to Email specification - regular expressions: `https://stackoverflow.com/a/201378`

They can be used to derive the structure of issue dictionary which gives some general view on the complexity of the further text mining algorithms and quality of reports. An important issue is also tracing dictionary features in short (e.g. monthly) and long-time perspectives. This is illustrated in Fig.1 for the analysed commercial project C1.

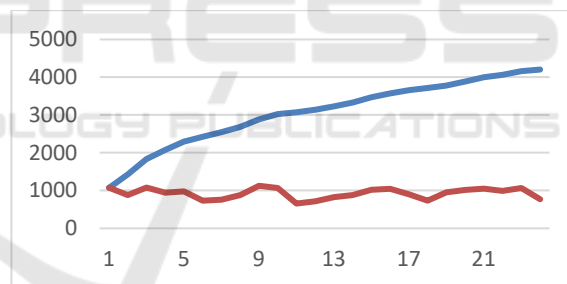


Figure 1: NL dictionary size (number of words) for project C1: monthly (lower plot) and accumulated statistics (upper plot) related to subsequent development months.

The presented plots show the sizes of dictionaries in two perspectives: accumulated (taking into account issues from 24 months) and monthly (for issues registered within subsequent months). It is interesting that the cardinality of NL dictionary (unique words) within months is relatively stable (700-1127, average 870); the long-term dictionary (accumulated) increases slowly up to 4205 for 24 months and 4763 for 36 months. Similar statistics we have derived for open access projects. For example, in case of *MongoDB* NL dictionary is richer: the monthly dictionary size was in the range 1916 – 2700, the accumulated size for 24 months assumed 8779. Nevertheless, here we also observe relatively small

increase in time. This allows to deeper analysis of used words or phrases characterising issues. The monthly cardinality of non-NL word classes is significantly lower, however long-time statistics showed linear increase. Dealing with these statistics it is also reasonable to refer them to the number of recorded issues and included tags. For *MongoDB* we have 119 - 307 issues per month (average 228), 12864 - 31563 tokens, average 21385, so average issue size is 94 tokens. For the commercial project C1 we have 73-163 issues per month 116 (1418- 9382 tokens) with average 50 tokens per issue.

Complex words correspond to class names, program variables etc., they usually are specified with appropriate name conventions, e.g. CamelCase, snakecase, Pascal. In most analysed project we found CamelCase notation, so such word phrases are classified as CCW. Classes specified in relevance to program packet, e.g.: `java.lang.String` – class `String` in `java.lang` packet are denoted as CPW.

Non classified words assumed on average 5.3% and 9.5% of all words for C1 and *MongoDB* projects, respectively. These statistics for subsequent months fluctuated in the ranges 3.8% - 8.1% and 8.2% - 11.0%, respectively. Here, it is also important to analyse the cardinality of unique non-classified words. For the commercial project it was 0-90 per month which resulted in the whole period of 3 years dictionary of 875 words in total. For *MongoDB* it was: 172-320 per month, and total for 2 years 1780. Hence, it can be verified manually and refined by introducing other word categories, e.g. technical acronyms. Such statistics can also be derived separately for diverse issue types. For the commercial project the description size of user story, task, new function and bug issues was 146, 40, 112 and 49 words/issue, respectively. The used NL word dictionary in new function was about 1.4 times bigger than for other issues. The ratio of non-NL elements per issue was 1.3-1.5 for external link, 1.5-3.0 for class name (with maximum 3 for user story), ratios of binary attachments for new function and bugs were 0.4 and 0.7, respectively, for the remaining issue types it assumed 0.16. The rate of other non-NL elements was in the range 0.03-0.05.

Some statistics of non-NL objects included in issues is presented in Tab. 1 for commercial (C1) and *MongoDB* (MDB) projects. They relate to email addresses, code snippets, classes, binary and image appendices, external and code change references, panel sections, respectively. Tab. 1 shows monthly ranges followed by average values (the second row).

Issue descriptions may comprise diverse technical words (TW), which are often relevant to the project

domain and implementation. Hence, it is reasonable to identify and interpret them. They can also appear in NLW or other word classes. Using TW words, we can correlate the considered text with specific problems, e.g., performed functionality, performance drawbacks. TW words can be extended for technical phrases (n-grams) playing a similar role to keywords - useful in characterizing topics of issues. These word sets and relevant n-grams can be derived iteratively and updated, they can also be a subject for team discussions to improve and standardize descriptions of issues, including sets of representative keywords.

Table 1: Monthly distribution of non-NL objects in issues.

Project	Adr	Code	Class	ApBin
C1	53-271	0-10	0-199	10-114
	119.1	0.9	22.7	47.1
MDB	0-7	30-117	23-249	0.176.9
	1.1	35.5	63.2	
	ApIm	ExtRef	ChRef	Pan
C1	-	6-211	0-1	-
		56	0.03	
MDB	0-29	5-21	0-11	0-12
	0.19	10.9	3.2	1.7

Dealing with special words (SW) defined by regular expressions it is worth identifying their context, e.g., preceding them phrases. For example, in project C1 external link in issue descriptions is preceded by “Go to” or “open”, in comments “please see”, “verified”; emails are preceded by “login with”. Attachments are usually preceded with “Please see” phrase. Sometimes issue descriptions comprise complex sections, e.g., code snippets. Nevertheless, less formalized section (not well structured) can also appear. Text feature statistics are useful in assessing informative value of reports, competence of reporters and improvement suggestions.

4 CLASSIFICATION SCHEMES

Software development and maintenance processes are documented in diverse repositories supported by issue tracing (IST), software version control (SVC) and other systems. In our studies we focused our attention on repository issues and relevant comments. In the first case we consider issue textual elements: title, summary, or description; comments are treated as indivisible entities. We have decided to preprocess the original texts according to classical text mining recommendations enhanced with our original special word transformations and derived text feature statistics. The transformed text is submitted to

classification schemes adapted to the semantic analysis goals (section 4.1). Some illustrative results of this original approach are presented in section 4.2.

4.1 Algorithms

The developed text mining of software repository textual contents is performed in two phases: i) text pre-processing, ii) classification. The first phase is composed of the following steps of *Algorithm 1*:

1. Extracting reports from the software repository for the analysis, using relevant API (e.g. Jira API), they can relate to a specified time period.
2. Creating the set of original textual entities (OTE) labelled with issue/comment ids.
3. Transforming OTE set into a signature form by replacing objects, FW and NCW words by special word tags (compare section 3). Here, we can use two conversions generalized and distinctive. In the first case we use a general tag for all words in the considered class (e.g. email, code, reporter) in the second one different words are attributed different tags (e.g. reporter#1, reporter#2,). Depending upon the analysis goal we can also admit a mixed approach with some word classes tagged distinctively (e.g. to trace issues generated by specific reporters, or referred to specific code commits).
4. Text reduction, e.g., using lemmatization of words, removing stop words, removing numerical words.

As opposed to classical text mining we do not unify upper- and lower-case characters (used in program variables, classes, etc.). The extracted text reports are manually analysed to derive the training set of text reports for the assumed classification. Here, we try to assure balanced representation of considered class categories. This set is pre-processed according to *Algorithm 1*, it can be enhanced with additional derived features (section 3) and submitted to classifier training block.

The developed classification scheme uses available text processing and machine learning packages from *sklearn* library. The set of considered text features include information from the fields of reported issues (e.g. description, title of the issue and diverse derived attributes/factors) or comment contents. It is also extended by derived sentiment factor (positive, negative, neutral) with *NLTK Sentiment Intensity Analyser* module. Some numerical properties, e.g. text length can be defined using t-shirt sizing method used in Agile story Point Estimation. The developed classification involves the following steps of *Algorithm 2*:

1. Prepare the input data for classification: textual data pre-processed with *Algorithm 1* and other features (e.g., text sentiment, diverse numerical properties), create learning and testing subsets with labelled classes.
2. Define sets of input data configuration (batches) for the further analysis
3. For each data configuration perform machine learning transformations with packages from *sklearn* for specified classification models
4. For each classification model perform cross validation using *sklearn GridSearchCV* package and select the best model which can be used in classification

The training process is based on cross validation technique with labelled data for training and for validation. For each classifier we evaluate its accuracy and select the best one to be used for future classification of transformed texts.

Examples of data configuration sets in step 2 for issue classification can be title and description, description + sentiment + number of attachments (compare section 4.2). In step 3 the text processing is performed with *sklearn Column Transformer* which combines input data properties in a unified matrix of numerical values. Textual components (e.g., issue description, title) are processed with *sklearn Tfidf Vectorizer*. We can use accuracy as scoring metric as our training data is evenly distributed.

Labelled features (e.g.: text sentiment, description length range) are processed by *OneHotEncoder* which generates sparse matrix representation. Numerical features (e.g. number of email references) are standardized with *StandardScaler*. In practice, it is reasonable to limit the number of features, so an important issue is their selection. Basing on our experience we decided to take into account description, title and up to 4 additional features, which resulted in diverse combinations submitted to classification experiments facilitating identification of the ones with high impact on classification quality.

It is important to note that step 3 of *Algorithm 2* should be adapted to the classification goal and analysed text specificity, which can be refined by an expert and the gained practice. It is possible to use diverse text feature combinations (e.g. original title, transformed description). Hence, it is reasonable to check classification efficiency (accuracy) of diverse combinations of the transformed text for the analysis and select the one with the best accuracy. In this selection some intuition of the expert is advantageous to select features giving the highest impact of classification. The standard classification accuracy (ACC) and F1 score are used here.

4.2 Illustrative Results

The effectiveness of our approach is illustrated for two classification tasks: issue types and comment categories. We distinguish 4 issue types: user story, task, new feature, bug. We verified the classification accuracy (ACC) for 6 classification models and diverse combinations of input data stream: pre-processed (according to algorithm 1) issue description (D), issue title (T), sentiment of title or description, the number of special words (|SW|), emails (|Em|), references to external appendices (|EA|) or to code changes (|ChanR|), classes, percentage of technical words (TW%), sizes of description or title in bytes (denoted with |.| brackets). This allowed us to assess the impact of diverse input data features.

Table 2: Issue type classification for C1 project.

Data configuration	LSV	MLP	RR
D	0.778	0.842	0.786
D+ EA + Sw + Em	0.783	0.861	0.767
D+T+TW%	0.931	0.925	0.903
D+Ts+ Sw	0.756	0.831	0.772
TW%+ Sw + T + D	0.772	0.794	0.758
D+T+ D	0.922	0.897	0.889

Table 2 presents an excerpt of issue type classification results for three best classifiers: LSV (linear super vector), MLP (Multi-Layer Perceptron), RR (Ridge Regression). The highest accuracy (ACC) relates to Description + Title + TW% data composition. For other classifiers the best results were achieved for the same data composition: NB (Naïve Bayes) - 0.872, Passive Aggressive (PA) - 0.894, KNN (k nearest neighbours) - 0.803. For data based only on the issue description text, the lowest values were 0.542, 0.531, and 0.661 for NB, PA and KNN classifiers, respectively. For other data combinations in most cases the results were lower than best ones by about up to 0.1, except a few up to 0.4 (PA, NB). We can notice that the dominant impact of accuracy has the pre-processed issue description and title. Adding numerical features needs proper selections, e.g. replacing TW% by 4 other numerical ones reduced ACC to 0.892. The calculated F1 scores were a little bit lower than ACC (typically 0.95ACC).

To check the impact of introduced input data features we verified the classification score taking into account only original comment description (without replacing non-NL words with tags) and we obtained lower ACC values. Issue type classification into four categories was rather a simple task in relevance to description and title. Repository reports in project C1 were systematically improved (in

particular titles). Such classification for open-source projects is less accurate. The classifications in relevance to other goals e.g., quality of description, diagnostic capabilities may show higher advantage of using transformed data and additional features. Here, it is worth noting that high accuracy of issue type classification can also be considered as some metric of issue reporting quality.

Classifying comments, we distinguished 4 categories: positive, response, question, fix. This classification is more demanding than the issue type. Here, we also checked the impact of diverse input data configurations: content of the comment text (pre-processed according to algorithm 1), sentiment, the numbers of change references, emails, external appendices references, classes, code snippets. Other features (e.g., content size) had negligible impact.

Table 3: Comment classification for project C1.

Data configuration	LSV	MLP	KNN
Cont	0.798	0.818	0.766
Cont + ChanR	0.805	0.805	0.785
Cont+ Sent+ ChanR	0.885	0.881	0.901
Cont+ ChanR + Email	0.795	0.798	0.772
Cont + ChanR + EA	0.801	0.801	0.762
Cont+ Code + class	0.798	0.815	0.766

Table 3 shows selected results for project C1. For all classifiers the best results related to the third data combination. We have also assessed comment classification for open-source project Groovy. The best accuracy ACC was 0.810 (F1=0.785) for LSV classifier and configuration: Cont + |ChanR| + |?|. Where |?| denotes the number of question marks in the text, Cont is the pre-processed comment text.

Better results for the commercial project can be explained by our knowledge of this project, stable team and systematic assessment of its quality including report ratings and critics provided by an external company.

5 DISCUSSION

The presented analysis of textual entities within software repositories confirmed that they comprise significant percentage of words/phrases in natural language which are mixed with diverse non-NL elements. Replacing these elements with labelled tags facilitates classification and interpretation of reports (issues, comments). The set of not classified entities can be further investigated to find uncovered ones and possibly create additional new classes. The class categories must be defined in relevance to the scope

of the analysis. Classes beyond the analysis scope can be skipped from the input data to concentrate on the considered ones and avoid blurring the analysis goal.

In case of long-term projects, the contents of dictionaries can change as shown in section 3 (Fig.1). Dictionaries for subsequent months are usually quite small and they increase for longer periods, however some saturation is observed. Hence, it is recommended to periodically refine the classification processes using upgraded training sets. The developed text pre-processing (*Algorithm 1*) can be used also in other analysis problems, e.g., based on finding similarities or clustering analysed reports/comments. It can be combined with other text mining/machine learning algorithms and tools.

The textual repository analysis can be extended considering other features of reports, e.g., specified in issue fields (priority, reporter id, software component) or timing properties. This hybrid approach may provide deeper insight into project problems. For an illustration we present the problem of so-called debt of defects (Zabardast et al., 2020). In practice, some reported issues are considered as negligible, and their solution/handling is postponed in time. This may become critical while the number of such defects becomes significant. This abrupt increase appears due to programmer fatigue with resolving similar defects. We have developed *Algorithm 3* of identifying this situation which includes two steps:

- 1) identifying significant non-linear increase of postponed defects (creating list P_D of these defects),
- 2) selecting suspicious defects in P_D with text mining based on recognizing some critical words in the description or using classification algorithms and training sets.

In the considered commercial project C1 in the first development phase each month appeared 1 postponed negligible defect, after project stabilization for the next 8 months we observed 8 such defects per month, then a rapid increase appeared from 13 to 63 defects in subsequent 6 months. Similar effects we observed also for some open-source projects. Using text mining based on predefined critical words (10-30 words) allowed us to identify typically 10% of postponed defects as critical. The selected suspicious defects have been confirmed as critical ones in 82%. Some of these defects were identified as fixed, with skipped registration of completed state in the issue state history (deficiency of reporters).

In many projects some issues are labelled as *wantfix*, which denotes that they will not be handled (Panichella et al., 2021). Typical reasons for this are: feature request/enhancement already implemented or

not needed, feature fixed in the context of previous ones or too expensive, etc. Such issues can be identified sooner with classification based on text mining and rejected without processing.

The quality of issue reports depends upon the recommendations imposed by project managers, competence and experience of relevant stakeholders. Good reports facilitate problem identification, diagnosis, and resolution. It is important to match what developers need and what issue reporters provide (Zimmerman et al., 2010). Hence, monitoring the quality of issue reporting is needed, it can base on some measurements and exchanged experience between the project team and users. This may result in deriving diverse improvement recommendation targeted at better readability, comprehension, illustration, automatic classification, etc. Issue descriptions are enhanced with generated comments during their analysis and resolution. The quality of these comments is also important. Tracing comment sequences is useful in this process.

Table 4: Comment sequence features in C1 project.

Comment sequence	Delay1	CDur
Fix, Pos, Res, Fix, Res, Posit	5d 23h	4d 2h
Fix, Que, Posit	5d 1h	5d 2h
Fix, Fix, Posit	0.1h	24d 2h
Fix, Que	7d 1h	4d 21h
Res, Que, Res, Que, Posit	0.3h	62d 2h

Analysing classified comments, we traced their sequences and timing properties. Table 4 presents some results for the commercial project. For the presented comment sequences (2-6 comments) we give time lapse (in days) between issue opening and the delay to the first comment (Delay1) followed by commenting duration (CDur). Some sequences are bizarre, e.g., lacking response (Res) after question (Que). Res comment was skipped (negligence of developer), however the problem has been resolved and the issue closed. The initial Res comment in the last sequence refers to the issue description.

For some classification problems we can perform combined text mining involving textual features of issue and comment elements extended with other selected numerical attributes. This seems to be valuable in case of assessing reporting quality, predicting issue resolution form. Moreover, cross sectional analysis is possible by restricting the considered issues or comments to specified types, analysis periods, involved reporters, etc.

6 CONCLUSIONS

The analysis of reported issues and relevant comments showed that a deeper study of the impact of non-NL elements is needed to explore semantical aspects of reports. This extends the space of the report analysis. The introduced text pre-processing and derived text features facilitate understanding the classification decisions and assure better accuracy. The validity threat to our study is the result restriction to a few projects. Nevertheless, the presented methodology is universal due to similarities in created software repositories (contents and structure).

The main text mining is targeted at classification or clustering of the considered textual objects, here we can use diverse statistical and machine learning techniques, which can be combined and adapted to project specificity and searched properties. This can be enhanced with contextual and correlations analysis. Distinction between texts generated by bots, authored by users, developers or testers could narrow semantic searches and extend the space of repository studies. Further research is targeted at correlating issue handling processes schemes and times with semantical aspects of textual descriptions and other issue features (basing on our previous experience - Sosnowski, et al., 2017; Polaczek & Sosnowski, 2021). This can be enhanced with questionnaire studies involving project participants.

REFERENCES

- Banerjee, S. et al. (2017). Automated triaging of very large bug repositories. In *Information and Software Technology*, 89.
- Ebrahimi, N. e al. (2019). An HMM-based approach for automatic detection and classification of duplicate bug reports. In *Information and Software Technology* 113 (2019) 98–109.
- Fan, Y., Xia, X., Lo, D., Hassan, A.E. (2018). Chaff from the wheat: characterizing and determining valid bug reports. In *IEEE Transactions on Software Engineering*. August 2018.
- Ferreira Gomes, L.A., et al. (2019). Bug report severity level prediction in open source software: A survey and research opportunities. In *Information and Software Technology* 115 (2019) 58–78.
- Herbold, S., Trautsch, A., Trautsch, F. (2020). On the feasibility of automated prediction of bug and non-bug issues. In *Empirical Software Engineering* 25, 5333–5369.
- Hindle, A., Onuczko, C. (2019). Preventing duplicate bug reports by continuously querying bug reports. In *Empirical Software Engineering*, vol. 24, no. 2.
- Huang, Y., et al. (2019). An empirical study on the issue reports with questions raised during the issue resolving process. In *Empirical Software Engineering* 24, 718–750.
- Li, Q. et al. (2022). A survey on text classification: From traditional to deep learning In *ACM Transactions on Intelligent Systems and Technology* vol. 13, no. 2.
- Nadeem, A., Usman Sarwar, M., Zubair Malik, M. (2021). Automatic issue classifier: a transfer learning framework for classifying issue reports. In *IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, October.
- Nagvani, N.K., Verma, S. (2012). CLUBAS: An algorithm and Java based tool for software bug classification using bug attributes similarities. In *Journal of Software Engineering and Applications*, vol.5, no. 6, 436-447.
- Panichella, S., Canfora, G., Andrea Di Sorbo (2021). ‘Won’t we fix this issue?’ Qualitative characterization and automated identification of wontfix issues on GitHub. In *Information and Software Technology*, Vol. 139, Nov., 106665.
- Polaczek, J., Sosnowski, J. (2021). Exploring the software repositories of embedded systems: An industrial experience. In *Information and Software Technology*, vol. 131.
- Sosnowski, J., Dobrzyński, B., Janczarek, P. (2017) Analysing problem handling schemes in software projects. In *Information and Software Technology*, vol. 91.
- Umer, Q., Liu, H., Sultan, Y. (2019). Sentiment based approval prediction for enhancement reports, In *Journal of Systems and Software*, 1555 (2019) 57-69.
- Vidoni, M. (2021). A systematic process for Mining Software Repositories: Results from a systematic literature review. In *Information and Software Technology*, vol 4 December.
- Wenting, D.A, et al. (2019). Analysis and detection of information types of open source software issue discussions. In *ICSE, IEEE/ACM 41st International Conference on Software Engineering*.
- Yahav, I., Shehory, O, Schwartz, D. (2019). Comments mining with TF-IDF: The inherent bias and its removal. In *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 3, pp. 437-450.
- Zabardast, E. Gonzalez-Huerta, J., Šmite, D. (2020). Refactoring, bug fixing, and new development effect on technical debt: An industrial case study. In *46th Euromicro SEAA Conference*, pp. 376-384.
- Zhang, T., et al. (2016). Towards more accurate severity prediction and fixer recommendation of software bugs. In *Journal of Systems and Software* 117 166–184.
- Zhang, W. et al. (2019). FineLocator: A novel approach to method-level fine-grained bug localization by query expansion. In *Information and Software Technology* 110 (2019) 121–135.
- Zimmermann, T. et al. (2010). What makes a good bug report? In *IEEE Transactions on Software Engineering*, vol. 36, no. 5, pp. 618-643.