Project 30: Sentiment on Technical Debt Dataset

Miro Kakkonen
University of Oulu
miro.kakkonen@student.oulu.fi

Mirko Kopsa
University of Oulu
mirko.kopsa@student.oulu.fi

I. Introduction

Software developers often indicate areas of potential improvement, incomplete features, or, generally, parts of program that require further attention in source code comments. This is referred to as self-admitted technical debt (SATD). Being able to recognize SATD is important for software maintenance and improvement, as it can help developers prioritize their work and allocate resources more effectively.

Automated recognition of SATD is a challenging task, as it requires the ability to understand the natural language text in the comments and the context in which they are written. In this project, we explore the use of multiple natural language processing and machine learning techniques to classify SATD in source code comments. We use a dataset of comments from open-source projects that have been manually labeled as SATD or not SATD. We are primarily interested in the effectiveness of sentiment analysis in augmenting the classification of SATD comments.

The report is organized as follows. We briefly survey the relevant literature for previous approaches and results in Section II. Section III defines the scope of the project by going through each individual approach used for exploring and classifying the data. The results are presented in Section IV and discussed in Section V. Section VI briefly considers the limitations and Section VII concludes our findings. Additional graphs are presented in the Appendix.

II. Related work

Several studies have explored SATD detection by focusing on identifying specific comment patterns. Potdar and Shihab (2014) [1] pioneered this approach, manually identifying 62 common patterns, such as ”FIXME” or ”TODO,” across multiple open-source projects. While effective, this pattern-based approach is limited by the need for extensive manual labeling, and it may miss SATD instances that don’t fit established patterns [2] [3]. To address these limitations, Huang et al. (2018) introduced a machine learning-based model leveraging text mining. Their approach involves feature selection from multiple projects to train classifiers, achieving superior F1-scores compared to traditional approaches by leveraging diverse linguistic features extracted from comments. This method demonstrated a significant increase in detection accuracy, highlighting the potential of automated text mining approaches to streamline SATD detection [3].

Furthermore, recent studies have sought to classify SATD into distinct types, enabling more targeted technical debt management. Maldonado and Shihab (2017) collected a largely used dataset as well as provided a taxonomy categorizing SATD into five types: design debt, defect debt, documentation debt, requirement debt, and test debt. Their work emphasized that design debt is the most prevalent, constituting 42-84 % of all SATD comments. This classification assists in prioritizing SATD management, as different types may have varying impacts on code quality and maintainability. [2]

Another line of research investigates the sentiment within SATD comments to assess debt severity. For instance, Cassee et al. (2022) examined the polarity of SATD comments, proposing that negative sentiment may correlate with higher priority issues. Their study found that certain SATD types, like incomplete functionalities, often carried negative tones, which may serve as a proxy for urgency. This nuanced analysis of sentiment allows project managers and developers to prioritize highrisk debt items effectively. [4]

In summary, advancements in SATD detection—from manual pattern recognition to automated text mining—have enhanced our ability to detect and classify SATD across large codebases. The integration of sentiment analysis and debt type classification offers additional insights, enabling developers to manage SATD more effectively and minimize its long-term impact on software projects.

III. Methodology

A. Dataset

The data collected by Maldonado and Shihab [2] contains 62 275 source code comments from 10 open source Java projects manually labeled as 1) ”defect”, 2) ”design”, 3) ”test”, 4) ”documentation”, 5) ”implementation” or 6) ”without classification”. The first five categories describe different kinds of technical dept, while the last one consists of comments not related to technical dept. The dataset is quite imbalanced, with 93 % of all comments being classified as ”without classification”. Table I contains one example entry from each class.

B. Most common words

The first task in any project dealing with large amounts of data is exploratory data analysis. We begin by performing pre-processing including removal of comment markers (//, /*, */, *) and stop words (and, or, in, ...), word tokenization and lemmatization. We then utilize word clouds and frequency histograms for graphical inspection of the most frequent words within each comment class. The words that are common to all classes as well as words unique to each class are also identified.

C. Latent Dirichlet allocation & FuzzyWuzzy

Latent Dirichlet allocation (LDA) is a generative probabilistic model based on Bayesian learning used for extracting latent topics within a collection of documents [5]. The input parameter n defines the number of topics the corpus will be divided into. We perform LDA on the comments with n = 6 to find out if the generated topics match with our six classes of interest.

FuzzyWuzzy is a simple Python library for calculating edit distances between strings using the Levenshtein distance. We utilize FuzzyWuzzy for string matching between the frequency distributions of our classes and the top 20 most important words associated with each topic discovered with LDA.

D. Empath

Empath [6] is a tool for analyzing text across lexical categories. Given seed words, Empath discovers related terms and validates a category with a crowd-powered filter. Additionally, it has 200 built-in, prevalidated categories drawn from existing knowledge bases and literature on human emotions.

Instead of generating new categories, we use the pre-defined ones to categorize the source code comments. Even though not all categories are useful in technical debt analysis, categories such as ”confusion”, ”fear”, ”hate”, ”negative emotion” and ”surprised” can indicate whether the source code comment is associated with a negative sentiment or not.

E. Sentiment analysis

SentiStrength [7] estimates sentiment strength in short texts, including informal language. Reported sentiment scores vary from not negative (-1) to extremely negative (-5) and from not positive (1) to extremely positive (5).

In addition to the positive and negative sentiment score, we calculate the sum of the positive and negative score (overall sentiment) and the average overall sentiment of each technical debt subcategory and the other (non-technical debt) category.

Additionally, we analyze the transitions between the sentiments of subsequent comments. We count the occurences where a comment of positive sentiment is followed by negative sentiment, when negative sentiment is followed by neutral, and so on.

F. Text classification

Finally, we attempt automated classification of source code comments based on supervised machine learning. We utilize tf-idf vectorization, which takes into account both the frequency of words in each document and their distribution across the entire corpus. We then augment this feature representation with the sentiment scores to address our primary research question:

Project name	Class	Comment
apache-ant-1.7.0	Defect	// I hate to admit it, but we don’t know what happened // here. Throw the Exception.
apache-ant-1.7.0	Design	// This is deprecated - use classespath in the future
argouml	Test	// TODO: Test if the generated string is correct.
columba-1.4-src	Documentation	/*TODO create javadocs for class */
jfreechart-1.0.19	Implementation	// TODO: add serialization support for images
jEdit-4.2	Without classification	//{{{ Getters and setters

TABLE I
EXAMPLES FROM THE DATASET

RQ: Does sentiment analysis have an impact on Technical Debt Detection? Does negative sentiment from source code comment contribute to technical debt?

The models chosen for this task are XGBoost and the BERT deep learning algorithm. XGBoost is a popular gradient boosting algorithm that has been shown to perform well on a variety of classification tasks. BERT is a transformer-based model that achieves state-of-the-art performance on a variety of natural language processing tasks.

We first try to classify each comment as either non-SATD or to their proper sub-category of SATD. Afterwards we discard the different subclasses and study the effectiveness of machine learning for binary recognition of SATD comments, where the specific sub-category of SATD is not of major interest.

IV. Results

A. *Most common words*

We utilized the NLTK library for basic preprocessing of the comments, including removal of stop words, tokenization and lemmatization. The resulting words are presented as word clouds for the SATD (classes 1-5) and non-SATD (class 6) categories in Figs 1 and 2 and as frequency distributions (top 20) in Figs 3 and 4.

We can see that words such as ”todo”, ”fixme” and ”need”, as well as the negative tokenization artifact ”n’t” (e.g. in ”doesn’t”) are very common in the SATD words. These words are also much less prevalent or completely absent from the non-SATD word cloud, implying their potential use in differentiating between these two classes.

Next we determined the words common to all six classes, as well as unique words for each class. The frequency histogram for common words is displayed in Fig. 5. For considering the cardinality of each class, the counts and proportions of common and unique words are presented in Table II. Despite the negligible amount of words common to **all six** classes, the proportions of unique words within each class are small enough to likely not have much value in differentiating between them.

The corresponding word clouds and frequency histograms for all the individual classes can be found in the Appendix.

Category	Count	Proportion (%)
Common words	105	0.004
Unique Defect words	328	0.058
Unique Design words	1773	0.054
Unique Test words	58	0.059
Unique Documentation words	36	0.044
Unique Implementation words	331	0.050
Unique Other words	21702	0.079

TABLE II
CARDINALITY OF COMMON AND UNIQUE WORDS

B. *Latent Dirichlet allocation & FuzzyWuzzy*

After converting the comments to a matrix of token counts with the CountVectorizer function of scikit-learn [8], we performed LDA analysis using LatentDirichletAllocation from the same library. The number of topics was chosen as $n = 6$ to match the classes within our dataset. The results are displayed in Table III.

Each topic is represented by the 20 most important words associated with that topic. Interpreting LDA results is not always easy, but if we look at e.g. topic 2, we see words like ”uml”, ”argouml”

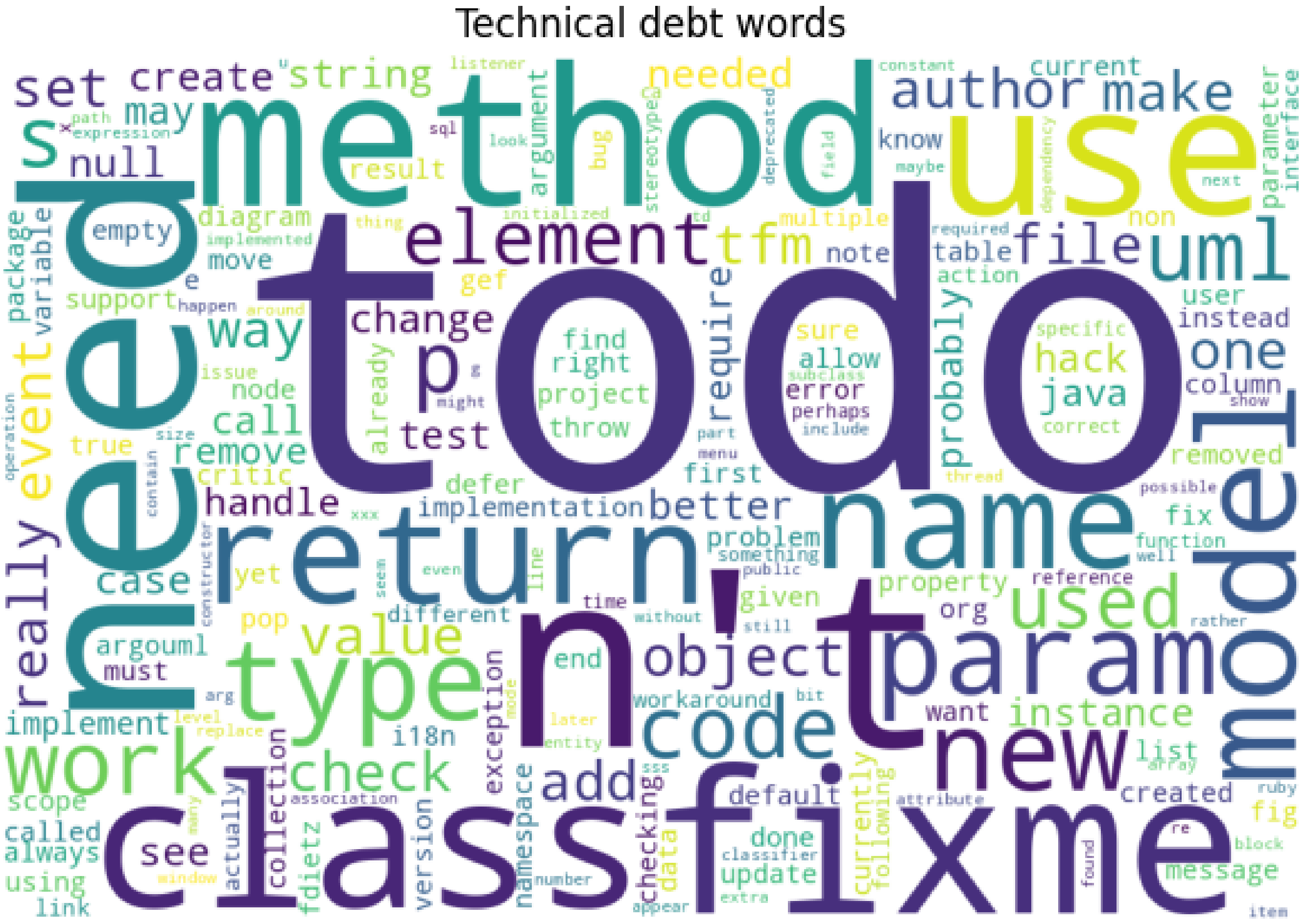


Fig. 1. Word cloud of SATD words

and ”diagram”, indicating that this topic probably contains comments related to the ArgoUML open source project.

Simple FuzzyWuzzy string matching between the topic words and the most frequent words of each class confirms that the topics found using LDA don’t correlate with the six categories of interest: every topic gets associated with category 6 (”without classification”), which is explained both by the dominating size of this category and the many different ways of classifying the comments within. Furthermore, the most important word for each topic is ”todo”, implying that no topics are actually related to the majority class, which consists of non-SATD comments.

C. Empath

Each comment is categorized into the built-in categories provided by Empath. The categorization can be processed further to detect technical debt but the existing categorization isn’t necessarily reliable. For example, the categories ”optimism”, ”business”, ”healing”, ”giving” and ”positive emotion” suggest a comment doesn’t imply technical debt. The comment is ”TODO: We need a better algorithm” and it is admitted as implementation debt.

In some cases the categorization can imply technical debt. The comment ”TODO: I suspect this isn’t needed call isn’t needed but don’t remove till out of alpha/beta stage” is admitted as implementation debt and it has 11 categories: ”dance”, ”crime”, ”dispute”, ”stealing”, ”communication”, ”hearing”, ”music”, ”speaking”, ”listen”, ”phone” and ”clean-

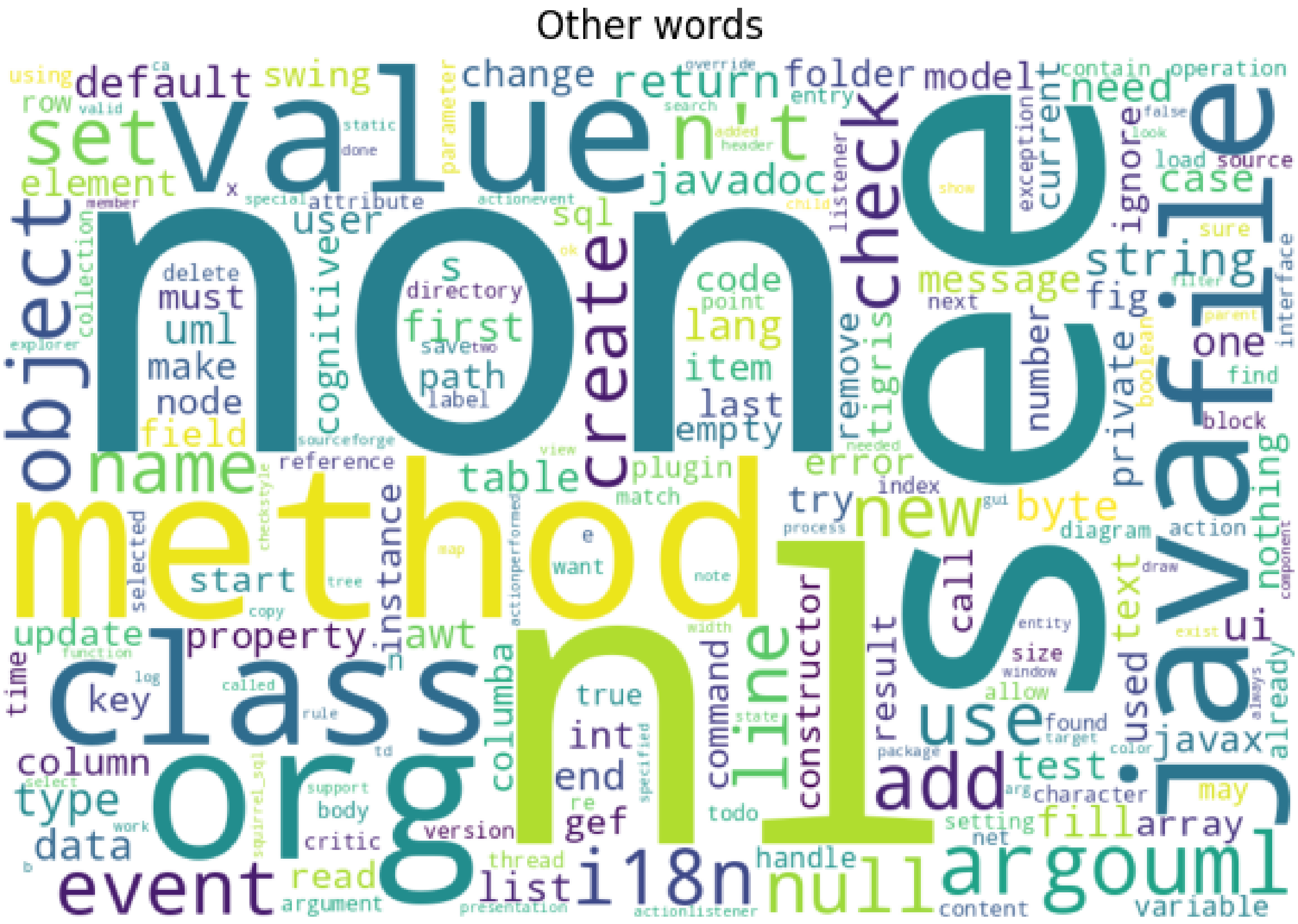


Fig. 2. Word cloud of non-SATD words

ing". While some of the categories are redundant, "crime", "dispute" and "stealing" have a negative connotation. "Cleaning" could imply the code needs improvements to clarity or refactoring. This is discussed in more detail in Section V.

D. Sentiment analysis

The average overall sentiment in each category is presented in Table IV. The most positive comment with the score of 4 is admitted as design debt ("TODO: It would be really, really nice to use this to also model components!"). The most negative comments with the score of -4 are admitted as design debt ("Name is set to the empty string (yuck!) by default - fix it"), implementation debt ("TODO: terrible implementation!") and non-technical debt ("we do not have to worry about queued additions to uninitialized collections, since they can only occur for inverse collections!").

The transitions between sentiments are presented in Table V and Table VI. Sentiment analysis of non-technical debt comments show the most sentiment transitions, as most comments are not admitted as debt, while design debt comments have the most positive to positive sentiment transitions.

E. Text classification

Before applying machine learning algorithms to the dataset we had to address the class imbalance. A simple solution is to undersample the majority classes. The amount of samples in each class before and after undersampling is presented in Table VII. The data was then divided into training and test sets with an 80/20 split.

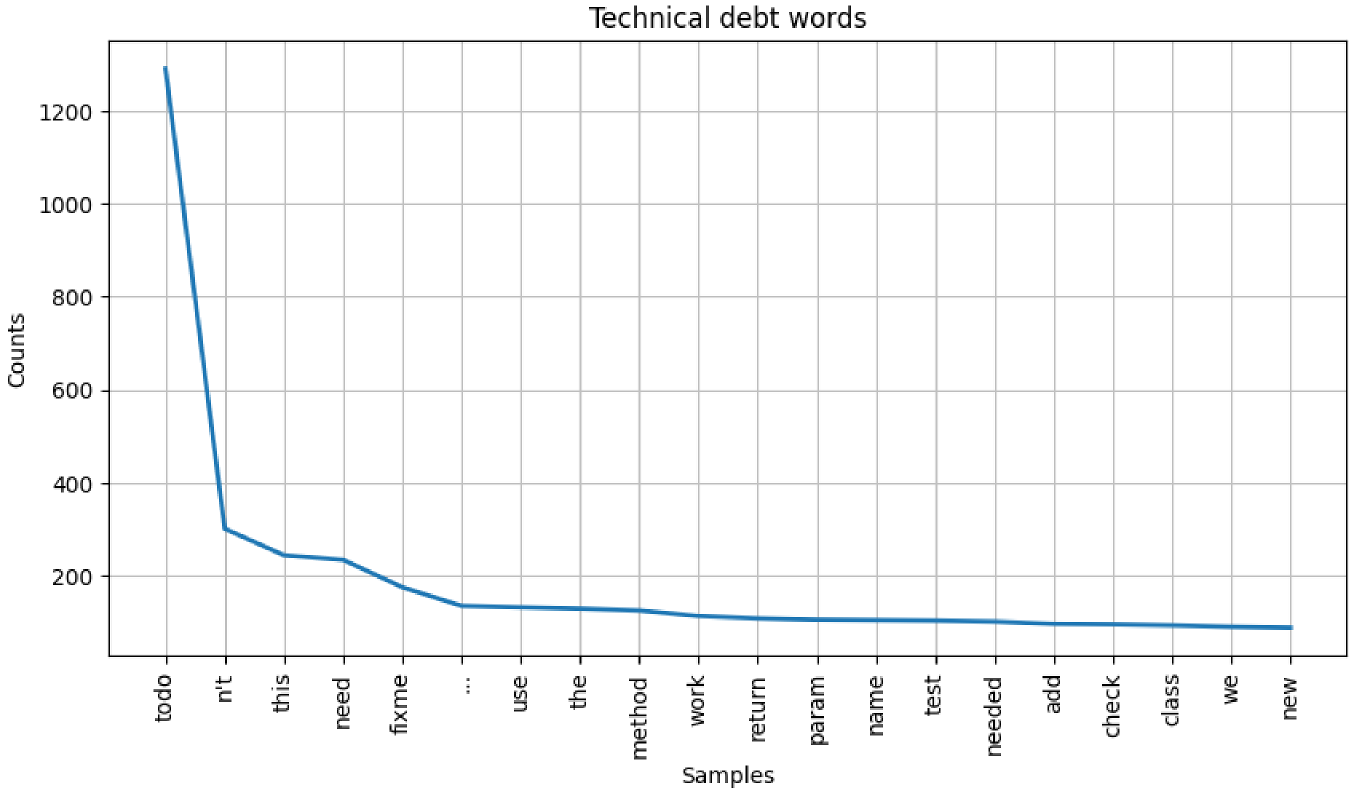


Fig. 3. Frequency histogram of top 20 SATD words

Our first approach to classification was to use tf-idf vectors and the XGBoost model. We then augmented the tf-idf vectors with sentiment scores and finally repeated both these processes with the BERT model. The resulting precision, recall and F1 scores are tabulated in Table VIII. [1]

Additionally, we studied the applicability of binary classification for automated recognition of SATD comments. Due to the high computational cost of the BERT model, we only used XGBoost for this task. All the SATD classes (defect, design, test, documentation & implementation) were combined into one and the model was trained to differentiate between SATD and non-SATD comments. The precision, recall and F1 scores are presented in table IX. These results are also evaluated through a small test set of 10 new comments created by the authors and presented in table X. It can be observed that all 10 comments get labeled accurately.

V. Discussion

As demonstrated by Cassee et al. [4], the comments in the SATD dataset can be categorized in more than one way. The topics found with Latent Dirichlet allocation prove the intuitive fact that these categories don't even necessarily need to have anything to do with technical debt. As such, LDA analysis did not prove very useful for our goal of recognizing SATD comments.

Using the built-in Empath categories can be used to detect technical debt, even though some limitations apply. While these categories can be used, they're not ideal for analyzing highly technical text. Notably, some of the built-in categories indicate sentiment, e.g. the ”negative emotion” category appears in 88 comments. However, sentiment analysis proves to be more effective in identifying negative sentiment within the comments. A smaller number of more precise custom categories could help detect technical debt more effectively.

As sentiment analysis takes into account both

[1] See the Conclusion (Section VII) for an explanation on the missing values.

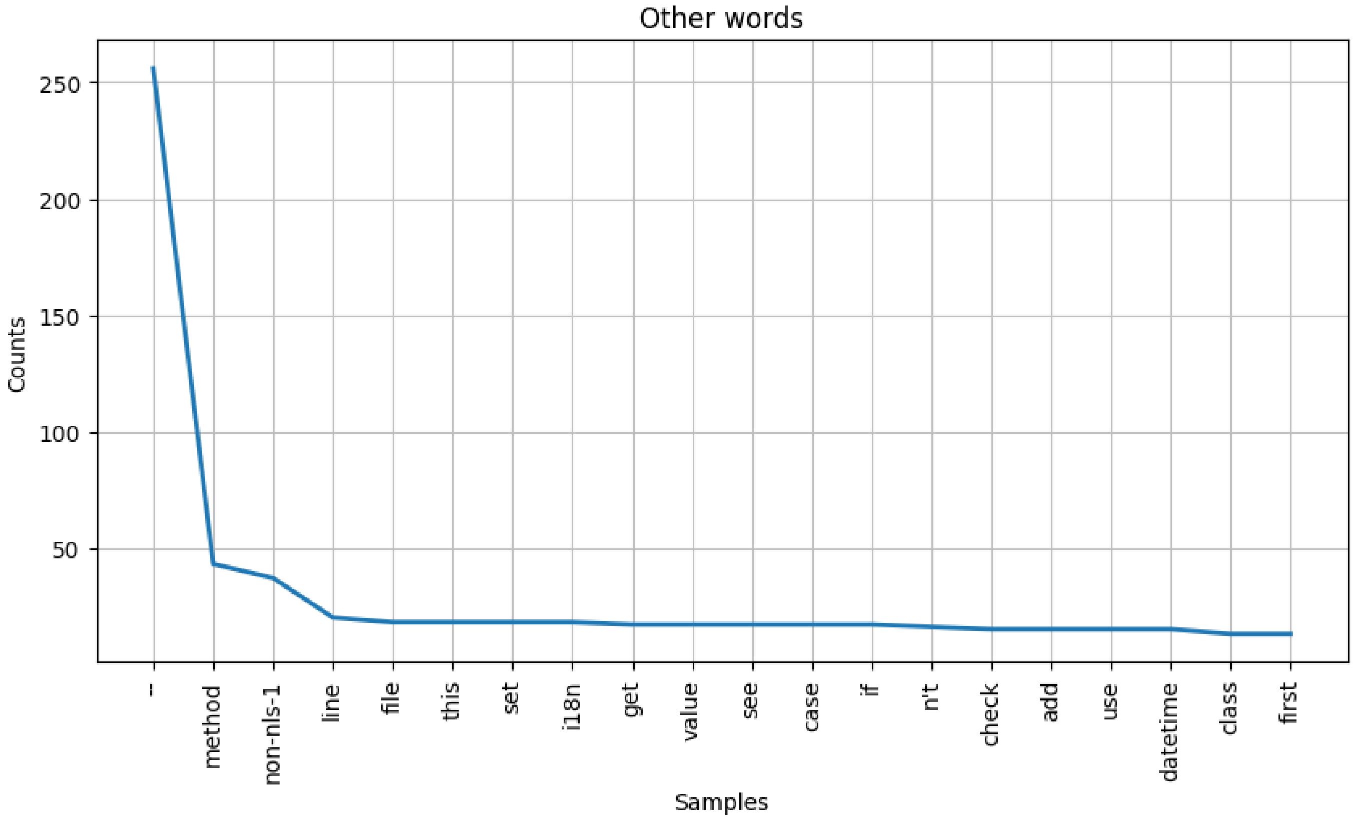


Fig. 4. Frequency histogram of top 20 non-SATD words

negative and positive sentiment, calculating the overall score as the sum of the two can cancel each other out. Taking only the negative sentiment score into account can be more effective as some of the technical debt comments can also be interpreted as having positive sentiment.

The threshold to classify the comment as positive or negative should be increased in order to better differentiate between positive, negative, and neutral comments. The highest and lowest score in the data is +/-4 out of 5, which makes the sentiment score range reduced from both ends together with the increased lower threshold.

Sentiment analysis faces the same challenge as using Empath to categorize source code comments, as it is not optimal for technical text. Cassee et al. [4] determined automated sentiment analysis tools to be inadequate for determining source code sentiment and decided instead to perform manual labeling of sentiment.

When analyzing the frequency of words in positive and negative comments, the sentiment analysis doesn't seem to provide any meaningful benefit. The semantic meaning of words seems more important to differentiate the categories, but even then, the words aren't directly related to categories without additional context.

The non-technical debt comments had the most sentiment transitions, as the majority of the comments were not categorized as technical debt. In the five technical debt categories, only design debt comments had comments with a positive sentiment followed by another positive comment.

As discussed above, the sentiment analysis results don't provide enough differentiation between negative and positive or neutral comments. For that reason, the transitions are spread out somewhat evenly, depending on which technical debt category had a higher number of comments.

Assuming that sentiment analysis is effective for source code comments and that the comment data is presented in the same order as it appears in

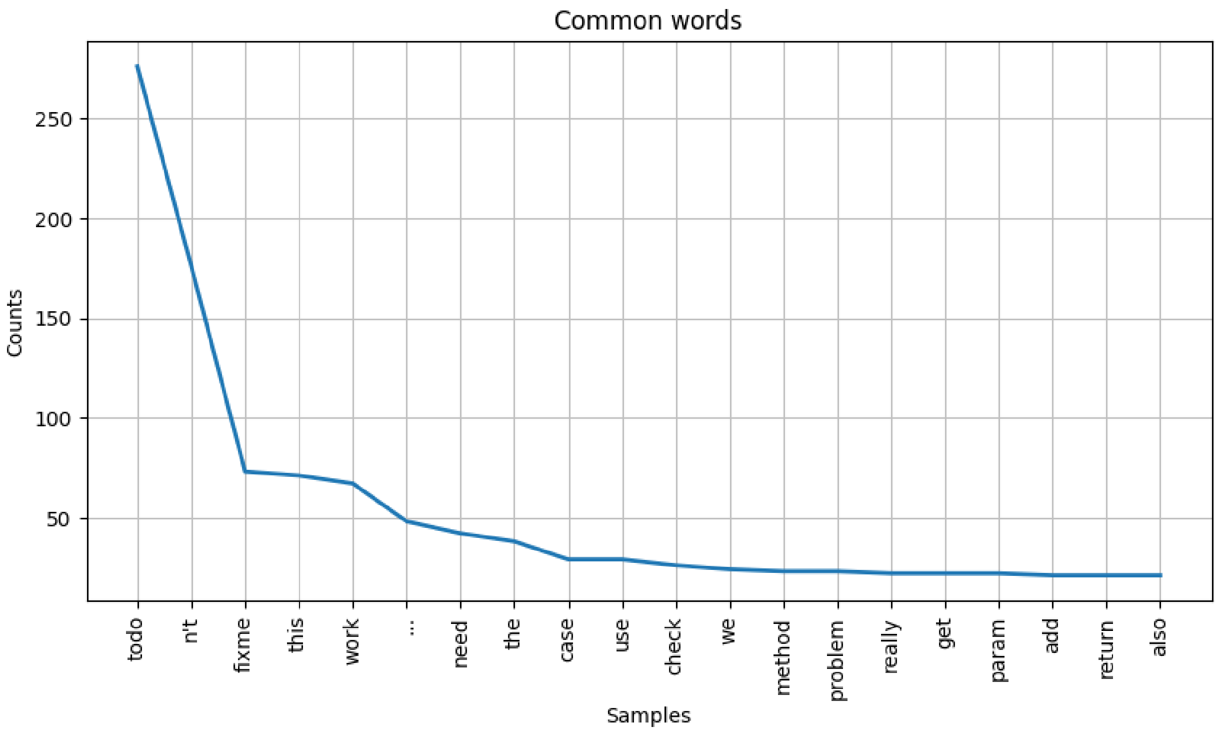


Fig. 5. frequency histogram of top 20 common words

Topic 1
todo implement null message need fixme don test author method error exception better update work want know correct checking set
Topic 2
todo needed uml make object need fixme argouml use case types implemented handle diagram implementation supported sure org problem class
Topic 3
todo use check add author need class list method fdietz work way tfm new event code possible table type test
Topic 4
todo value param need return fixme file set method block fix variable true null scope scopes elements current used does
Topic 5
todo don require pop non nls fixme path file java work method awt new make add really throw doesn thread
Topic 6
todo string td code right new i18n use return param needs model created add align object does project just fixme

TABLE III
TOPIC MODELING RESULTS

Category	**Average**
Defect	-0.046
Design	-0.035
Test	0.047
Documentation	0.046
Implementation	0.034
Other	-0.007

TABLE IV
OVERALL SENTIMENT SCORES BY CATEGORY

Cat.	**P.→P.**	**P.→Neg.**	**Neg.→Neg.**	**Neg.→Neu.**
Defect	0	6	25	7
Design	12	55	101	41
Test	0	1	3	1
Doc.	0	2	1	0
Impl.	0	8	10	1
Other	5	374	462	61

TABLE V
TRANSITIONS FROM POSITIVE TO POSITIVE, POSITIVE TO NEGATIVE, NEGATIVE TO NEGATIVE AND NEGATIVE TO NEUTRAL SENTIMENT BY CATEGORY

the source code, the sequences of sentiments could potentially be used to identify technical debt. Sequences of negative comments indicate that a part of the code needs refactoring or has readability issues. Neutral and positive comments can suggest that the code is well-documented.

Cat.	Neu.→Neu.	P.→Neu.	Neu.→Neg.	Neu.→P.
Defect	93	2	330	8
Design	536	24	1861	72
Test	11	0	67	1
Doc.	7	1	39	3
Impl.	71	1	656	9
Other	4803	44	52041	413

TABLE VI
TRANSITIONS FROM NEUTRAL TO NEUTRAL, POSITIVE TO NEUTRAL, NEUTRAL TO NEGATIVE AND NEUTRAL TO POSITIVE SENTIMENT BY CATEGORY

Class	Before	After
Defect	472	472
Design	2703	541
Test	85	85
Documentation	54	54
Implementation	757	757
Other	58204	583

TABLE VII
SAMPLE COUNTS BEFORE AND AFTER UNDERSAMPLING

Model	Features	Precision	Recall	F1 Score
XGBoost	tf-idf	0.73	0.61	0.64
XGBoost	tf-idf+sentiment	0.75	0.64	0.68
BERT	tf-idf	0.78	0.72	0.73
BERT	tf-idf+sentiment	-	-	-

TABLE VIII
CLASSIFICATION RESULTS

Model	Features	Precision	Recall	F1 Score
XGBoost	tf-idf	0.85	0.89	0.87
XGBoost	tf-idf + sentiment	0.89	0.92	0.90

TABLE IX
BINARY CLASSIFICATION RESULTS

When it comes to text classification with machine learning methods, the results are quite promising. Our best result was achieved in binary classification augmented with sentiment scores. An F1 score of 0.90 is quite high and indicates that the model performs well in recognizing comments describing SATD. We can also see that while not an extensive improvement, classification augmented with sentiment analysis does produce better results than pure tf-idf vectors in both class-based and binary classification. As such, we can conclude that **sentiment analysis does have a (slight) positive impact on technical debt detection**. However, based on the results in Table IV, **the contribution of negative sentiment in particular is inconclusive.**

VI. Limitations

It is worth noting certain limitations that impact the generalizability of our findings. Despite the substantial amount of comments collected, only 10 different projects are represented—all of them open source and written in the Java language. This is hardly representative of the entire software development community, with its myriad of languages, project types and programming styles. Additionally, significant class imbalance further limits the volume of data available for reliable predictive modeling, thus further constraining the broader applicability of our results.

VII. Conclusion

The one project specification we could not fulfill in time was the augmentation of the BERT model with sentiment scores. This is due to lacking sufficient hardware for multiple training runs of the deep learning model. However, this does not impact our main findings.

The results obtained from our experiments suggest that sentiment analysis, particularly the incorporation of sentiment scores, has a non-insignificant positive effect on technical debt detection in source code comments. The binary classification with sentiment scores achieved the highest F1 score, indicating that sentiment can aid in identifying technical debt, but its contribution, especially from negative sentiment, remains unclear.

Future work could explore deeper integration of sentiment analysis in more sophisticated models, such as BERT, with sufficient computational resources for multiple training iterations. Additionally, further refinement of sentiment analysis tools tailored specifically for technical text might improve the accuracy and utility of sentiment scores in this domain. It would also be beneficial to test different sentiment thresholds and analyze their effects on model performance, as well as experiment with other machine learning techniques to improve the identification of technical debt.

In conclusion, while sentiment analysis is not sufficient for detecting technical debt in source

Comment	**True class**	**Predicted class**
This doesn't work the way it should	SATD	SATD
Need to switch to a different library later on	SATD	SATD
TODO: Add support for multithreading and parallel processing	SATD	SATD
TODO add documentation for the function	SATD	SATD
The sky is quite blue today	non-SATD	non-SATD
Finds the local optima of the input	non-SATD	non-SATD
Calculates the sum of an array	non-SATD	non-SATD
display the spectrogram of an audio signal	non-SATD	non-SATD

TABLE X
NEW COMMENTS CREATED FOR TESTING

code comments, it does offer a valuable tool that can complement other methods such as tf-idf-based classification. With continued refinement and adaptation to the unique characteristics of source code, sentiment analysis could play an increasingly important role in automated technical debt detection and management.

REFERENCES

[1] A. Potdar and E. Shihab, "An exploratory study on self-admitted technical debt," *Proceedings - 30th International Conference on Software Maintenance and Evolution, ICSME 2014*, pp. 91–100, Dec. 2014. DOI: 10.1109/ICSME.2014.31.

[2] E. d. S. Maldonado and E. Shihab, "Detecting and quantifying different types of self-admitted technical debt," in *2015 IEEE 7th International Workshop on Managing Technical Debt (MTD)*, 2015, pp. 9–15.

[3] Q. Huang, E. Shihab, X. Xia, D. Lo, and S. Li, "Identifying self-admitted technical debt in open source projects using text mining," *Empirical Software Engineering*, vol. 23, no. 1, pp. 418–451, Feb. 2018.

[4] N. Cassee, F. Zampetti, N. Novielli, A. Serebrenik, and M. Di Penta, "Self-admitted technical debt and comments' polarity: An empirical study," eng, *Empirical software engineering : an international journal*, vol. 27, no. 6, 2022.

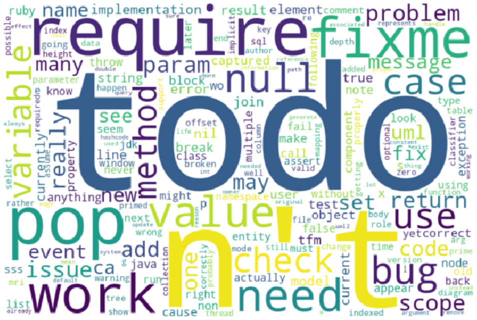
[5] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *J. Mach. Learn. Res.*, vol. 3, no. null, pp. 993–1022, Mar. 2003, ISSN: 1532-4435.

[6] E. Fast, B. Chen, and M. S. Bernstein, "Empath: Understanding Topic Signals in Large-Scale Text," en, in *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, San Jose California USA: ACM, May 2016, pp. 4647–4657, ISBN: 978-1-4503-3362-7. (visited on 11/09/2024).

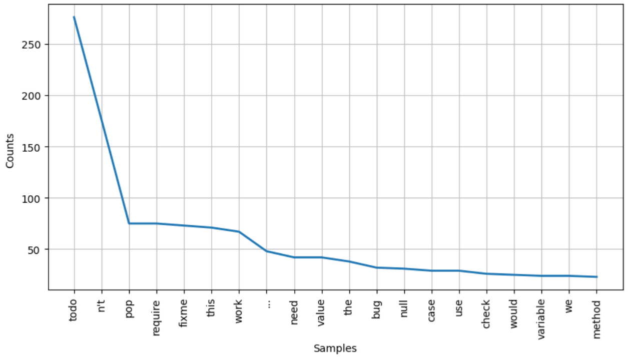
[7] M. Thelwall, *Sentistrength - sentiment strength detection in short texts - sentiment analysis, opinion mining*, http://web.archive.org/web/20080207010024/http://www.808multimedia.com/winnt/kernel.htm, Accessed: 2024-11-09.

[8] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

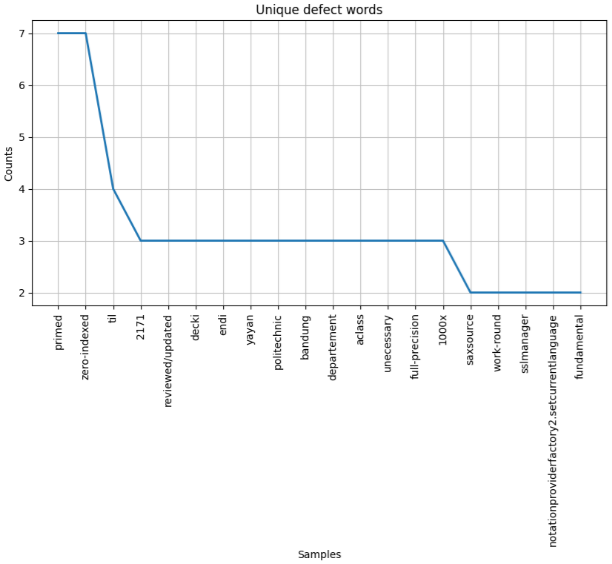
Appendix A
Additional graphs



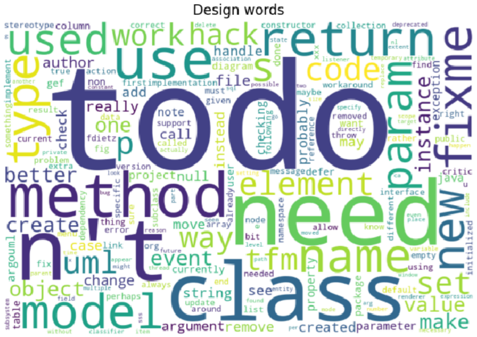
Defect words



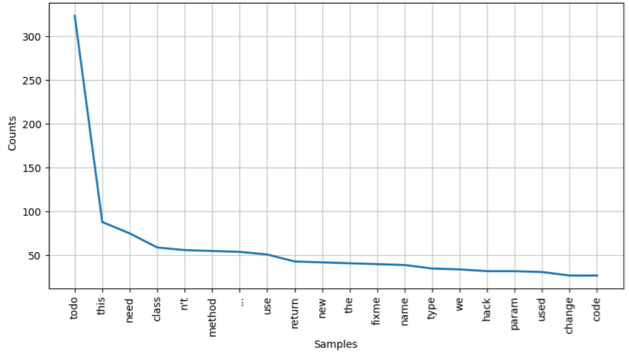
Defect words



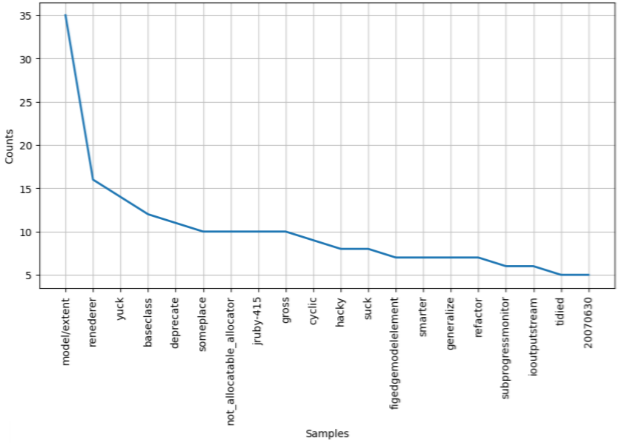
Unique defect words



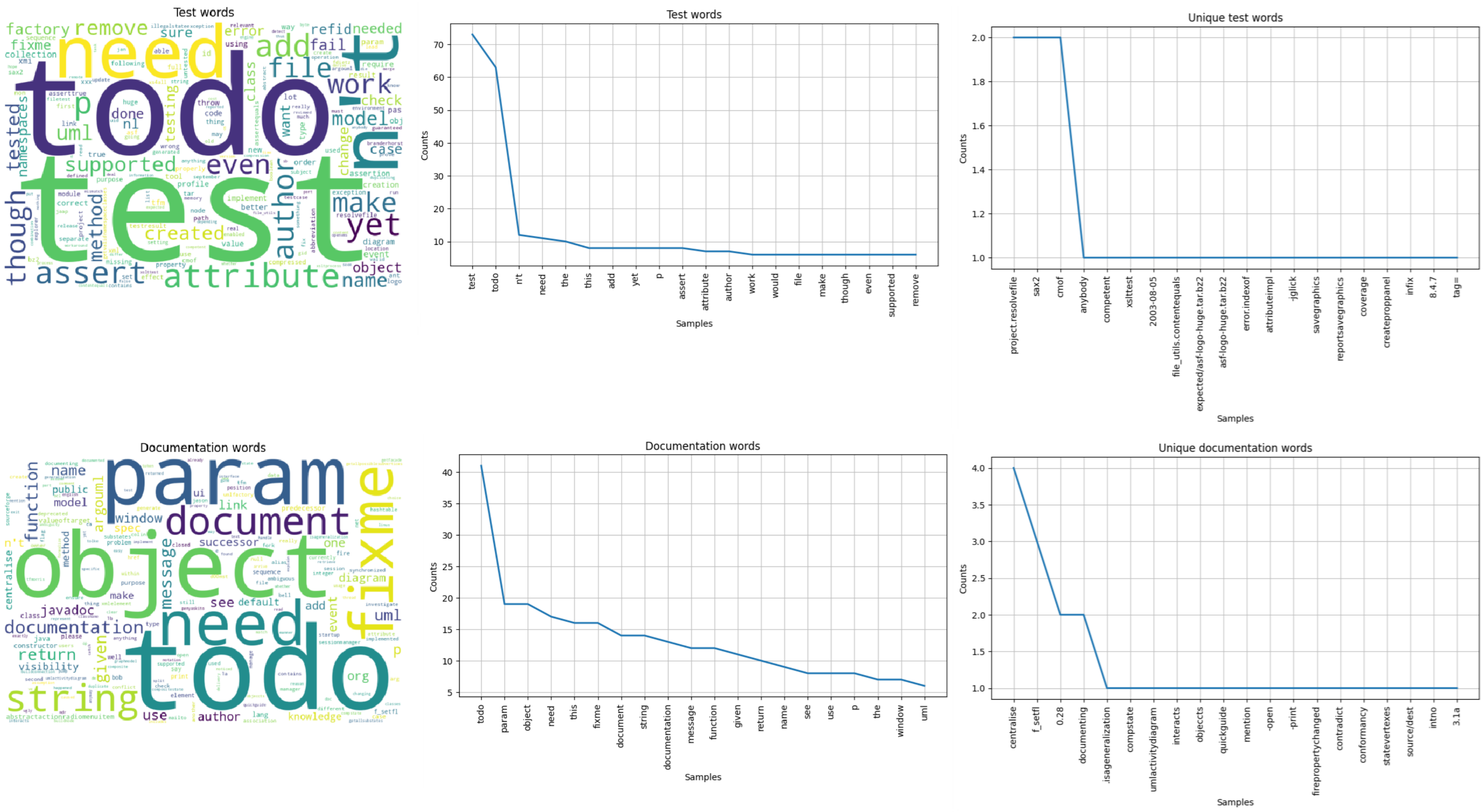
Design words

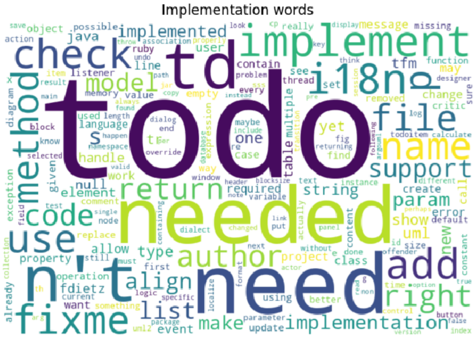


Design words

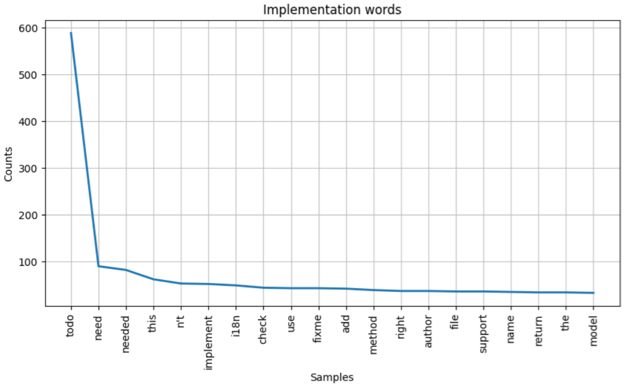


Unique design words

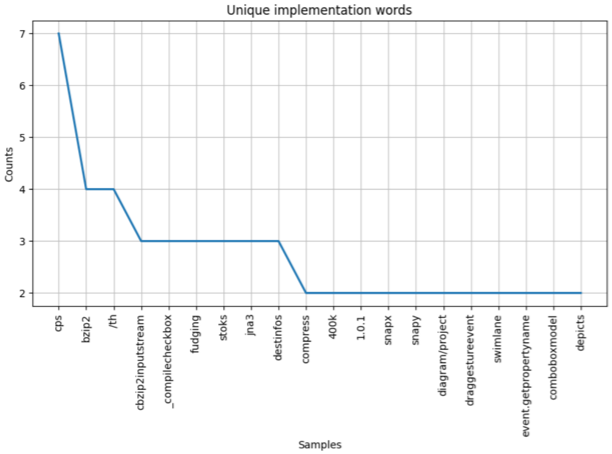

Test words
test
todo
n't
need
add
file
work
fail
refid
needed
factory
remove
sure
error
fixme
collection
check
model
case
p
done
nl
uml
tested
namespaces
testing
class
want
supported
even
author
make
yet
created
method
though
assert
attribute
name
object
Test words
Counts
70
60
50
40
30
20
10
test
todo
n't
need
the
this
add
yet
p
assert
attribute
author
work
would
file
make
though
even
supported
remove
Samples
Unique test words
Counts
2.0
1.8
1.6
1.4
1.2
1.0
project.resolvefile
sax2
cmof
anybody
competent
xslttest
2003-08-05
file_utils.contentequals
expected/asf-logo-huge.tar.bz2
asf-logo-huge.tar.bz2
error.indexof
attributeimpl
-jglick
savegraphics
reportsavegraphics
coverage
createproppanel
infix
8.4.7
tag=
Samples
Documentation words
param
document
object
fixme
need
todo
string
documentation
javadoc
return
visibility
function
name
public
model
window
link
successor
message
diagram
see
default
add
uml
given
author
use
org
Documentation words
Counts
40
35
30
25
20
15
10
5
todo
param
object
need
this
fixme
document
string
documentation
message
function
given
return
name
see
use
p
the
window
uml
Samples
Unique documentation words
Counts
4.0
3.5
3.0
2.5
2.0
1.5
1.0
centralise
f_setfl
0.28
documenting
.isageneralization
compstate
umlactivitydiagram
interacts
objeccts
quickguide
mention
-open
-print
firepropertychanged
contradict
conformancy
statevertexes
source/dest
intno
3.1a
Samples



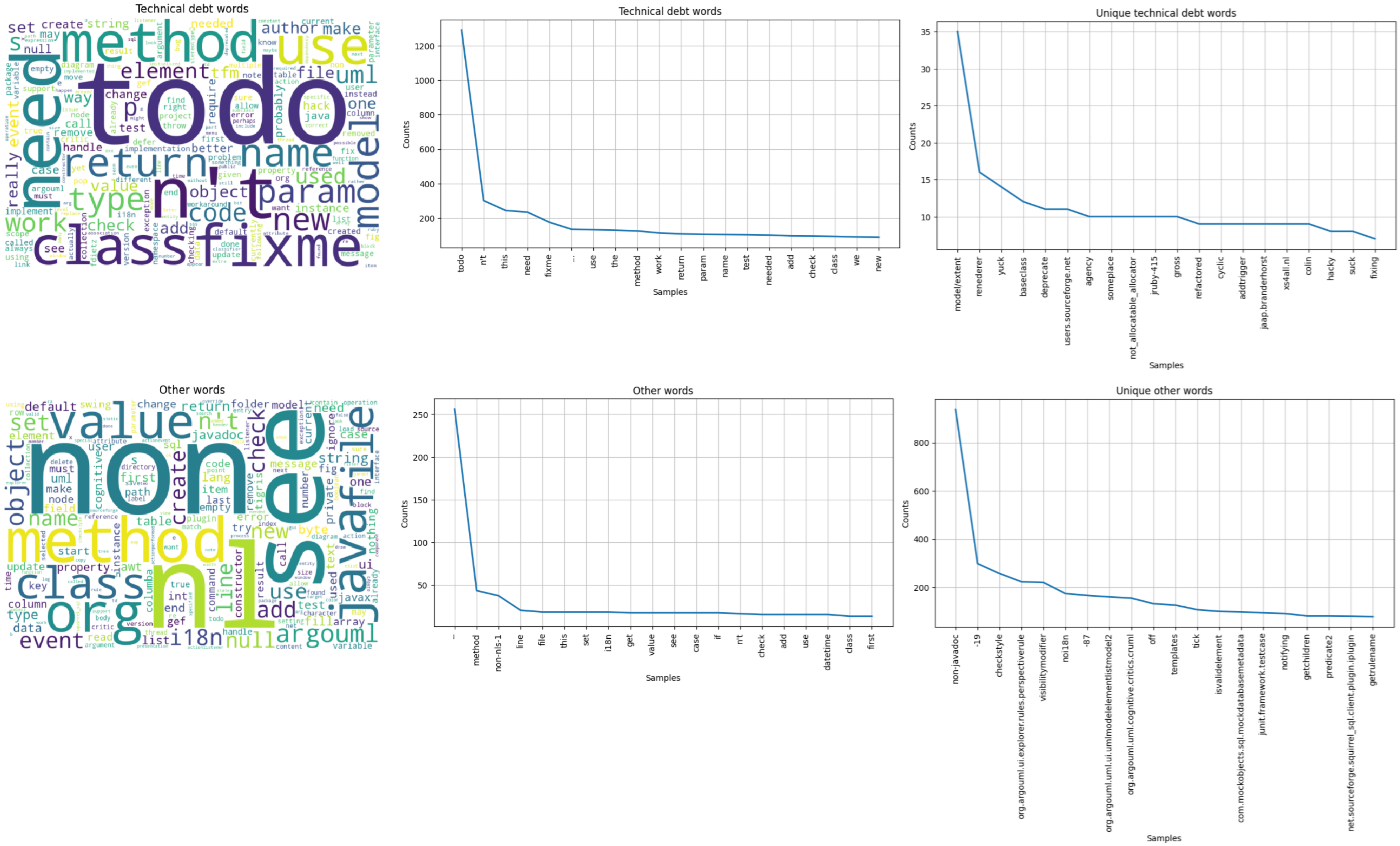
Implementation words



Implementation words



Unique implementation words


Technical debt words
todo
method
use
need
n't
return
name
fixme
class
model
type
param
code
new
object
element
tfm
file
uml
work
check
add
value
used
make
author
needed
string
create
set
null
way
change
test
call
handle
remove
event
really
implement
case
see
one
hack
java
require
probably
instance
better
Technical debt words
Counts
Samples
1200
1000
800
600
400
200
todo
n't
this
need
fixme
...
use
the
method
work
return
param
name
test
needed
add
check
class
we
new
Unique technical debt words
Counts
Samples
35
30
25
20
15
10
model/extent
renederer
yuck
baseclass
deprecate
users.sourceforge.net
agency
someplace
not_allocatable_allocator
jruby-415
gross
refactored
cyclic
addtrigger
jaap.branderhorst
xs4all.nl
colin
hacky
suck
fixing
Other words
none
nls
see
value
method
org
class
java
file
n't
set
default
object
name
create
check
argouml
i18n
null
list
event
use
add
line
string
message
javadoc
return
new
byte
ui
javax
type
column
key
update
cognitive
Other words
Counts
Samples
250
200
150
100
50
-
method
non-nls-1
line
file
this
set
i18n
get
value
see
case
if
n't
check
add
use
datetime
class
first
Unique other words
Counts
Samples
800
600
400
200
non-javadoc
-19
checkstyle
org.argouml.ui.explorer.rules.perspectiverule
visibilitymodifier
noi18n
-87
org.argouml.uml.ui.umlmodelelementlistmodel2
org.argouml.uml.cognitive.critics.cruml
off
templates
tick
isvalidelement
com.mockobjects.sql.mockdatabasemetadata
junit.framework.testcase
notifying
getchildren
predicate2
net.sourceforge.squirrel_sql.client.plugin.iplugin
getrulename