The Journal of Systems & Software 178 (2021) 110976

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

An empirical study on the co-occurrence between refactoring actions and Self-Admitted Technical Debt removal☆

Martina Iammarino, Fiorella Zampetti, Lerina Aversano, Massimiliano Di Penta *

University of Sannio, Italy

ARTICLE INFO

Article history:
Received 17 June 2020
Received in revised form 23 March 2021
Accepted 8 April 2021
Available online 16 April 2021

Keywords:
Self-Admitted Technical Debt
Software refactoring
Software quality

ABSTRACT

Technical Debt (TD) concerns the lack of an adequate solution in a software project, from its design to the source code. Its admittance through source code comments, issues, or commit messages is referred to as Self-Admitted Technical Debt (SATD). Previous research has studied SATD from different perspectives, including its distribution, impact on software quality, and removal. In this paper, we investigate the relationship between refactoring and SATD removal. By leveraging a dataset of SATD and their removals in four open-source projects and by using an automated refactoring detection tool, we study the co-occurrence of refactoring and SATD removals. Results of the study indicate that refactoring is more likely to co-occur with SATD removals than with other commits, however, in most cases, they belong to different quality improvement activities performed at the same time. Moreover, if looking closely at refactoring actions co-occurring with SATD removal in the same code entities, a relationship between these activities can be found. Finally, we found how both source code quality metrics and SATD removals play a statistically significant role in the likelihood that the commit applies a refactoring action.

1. Introduction

As defined by Cunningham (1992), Technical Debt (TD) is *"not quite right code which we postpone making it right"*. The reasons why TD occurs in software projects can be many-fold, ranging from deadline pressure to sub-optimal API choice or availability. In many circumstances, developers annotate TD by "admitting" it in source code comments or commit messages. This is referred to as Self-Admitted Technical Debt (SATD) (Potdar and Shihab, 2014a).

Different studies have investigated how "not quite right code" is made right, *i.e.,* how SATD has been addressed, or else what are the characteristics of SATD-affected (or, in general, TD-affected) source code. Zampetti et al. (2017) found that source code metrics, readability metrics, and static analysis warnings could be used as predictors for the presence of design TD. In a different study, Zampetti et al. (2018) found that SATD removals follow specific source code change patterns – for example changes in conditionals, method calls, or method signatures – although there is a quite large proportion of removals that happens accidentally, *e.g.,* because a method or a class disappears. This confirms what previously found by Tufano et al. (2017) and by Chatzigeorgiou and Manakos (2014) about code smell removals. In summary, while one may be tempted to conjecture that SATD is removed with changes aimed at enhancing source code quality, there may be several cases in which such removal is due to chance.

One specific action aimed at improving source code quality is represented by refactoring, *i.e., "the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure "* (Fowler, 1999). Previous research has shown how refactoring is reflected by improvements in some source code metrics (Alshayeb, 2009; Hegedüs et al., 2018; Moser et al., 2007; Shatnawi and Li, 2011; Stroggylos and Spinellis, 2007; Szoke et al., 2017), but also, some contradictory results, pointing out how refactoring rarely improves metrics or remove code smells (Bavota et al., 2015).

Based on the results of previous research, this paper aims to shed light on the relationship between SATD removals and refactoring actions. More specifically, we investigate whether (i) refactoring actions and SATD removals co-occur over the project change history; (ii) when refactoring actions and SATD removals co-occur, whether the refactoring contributes to the SATD removal, or whether they just happen to occur together because of the commit devoted to various quality improvement activities; (iii) what types of refactoring actions relate more to SATD removals; and (iv) whether, compared to other metrics capturing software quality, commits involving SATD removals have more chances to contain refactoring actions.

☆ Editor: Alexander Chatzigeorgiou.

* Corresponding author.
E-mail addresses: iammarinomartina@gmail.com (M. Iammarino), fiorella.zampetti@unisannio.it (F. Zampetti), aversano@unisannio.it (L. Aversano), dipenta@unisannio.it (M. Di Penta).

https://doi.org/10.1016/j.jss.2021.110976
0164-1212/

This study has been performed by relying on curated SATD instances from four Java open-source projects part of an existing dataset (Maldonado et al., 2017a), and by automatically detecting refactoring actions using the RMiner automated tool (Tsantalis et al., 2018). Results of the study indicate that (i) refactoring actions have three to six more chances to co-occur with SATD removals than with other commits, (ii) with some exceptions, such a co-occurrence does not directly contribute to the SATD removal, (iii) when such a contribution occurs, it is often due to extracting method refactoring actions or to renaming combined with other changes, and (iv) even if quality metrics (*i.e.,* CBO and LOC) have a statistically significant effect on the likelihood that the commit applies a refactoring action, the removal of SATD has 1.7 more chances of applying a refactoring action in the commit.

This paper extends a preliminary study we performed (Iammarino et al., 2019) along with the following directions:

1. A qualitative analysis and discussion of the cause–effect relationship between refactoring actions and SATD removals. To this aim, we manually inspected SATD comments, commit messages, and source code diffs.
2. A more thorough statistical analysis of the results, in particular, to investigate the types of refactoring co-occurring more with SATD removals.
3. A new research question that investigates, from a quantitative perspective, the correlation of SATD removals and quality indicators trends. Specifically, we use a mixed-effect generalized linear model of logistic regression considering the refactoring occurrences as the dependent variable and SATD removals events together with source code metrics as independent variables.

The remainder of the paper is organized as follows. Section 2 describes the study definition and planning. Results are presented and discussed in Section 3, while threats to the study validity are discussed in Section 4. After a discussion of the related literature (Section 6), Section 7 discusses the implications of our work, concludes the paper, and outlines directions for future work.

The working dataset used in our empirical study is available for replication purposes (Iammarino et al., 2021).

2. Empirical study definition and planning

The *goal* of our study is to investigate the co-occurrence between refactoring actions and SATD removals, with the *purpose* of understanding whether refactoring contributes to such removal, or whether, instead, both activities just co-occur in the context of a general quality improvement activity. The *context* of the study consists of the evolution history of four Java open source projects – Camel, Gerrit, Log4j, and Tomcat – for which SATD and their removals are available in a dataset by Maldonado et al. (2017a), and for which we detected refactoring actions using the RMiner tool (Tsantalis et al., 2018).

To address the aforementioned goal, we first investigate to what extent refactoring actions happen in the same commits in which SATD is removed, and whether this happens with a greater proportion than in other commits. Therefore, we address our first research question:

RQ$_1$: *To what extent do refactoring actions co-occur with SATD removals?*

This research question provides a "birds-eye" view to the refactoring occurrences and aims at simply quantifying the proportions of those occurring in quality-improvement commits, and specifically those where a SATD has been removed.

While simply investigating the co-occurrence of SATD removals and refactoring actions is *per se* interesting as it shed the light on those commits that generally relate to source code quality improvement, it is also clear that refactoring actions may or may not have contributed to the SATD removal. For example, in the same commit, a SATD could be removed by replacing an API, while a refactoring has occurred elsewhere. Therefore, we address our second research question:

RQ$_2$: *To what extent do refactoring actions actually contribute to SATD removal?*

RQ$_2$ constitutes the essence of our study considering that refactoring actions may either co-occur with SATD removal because a developer is performing multiple quality improvement changes altogether or because the refactoring actually contributes to removing the SATD.

The considered dataset of SATD removals pertains to design debt affecting Java methods, for which a previous study highlighted that the removal follows specific change patterns (Zampetti et al., 2018). Therefore, some types of refactoring actions may occur more frequently than others when addressing design SATD. This is investigated in our third research question:

RQ$_3$: *What are the types of refactoring actions that co-occur more with SATD removals?*

The rationale of this research question is to determine, from the perspective of a maintainer, what kinds of refactoring types usually contribute to SATD removals, as opposed to other refactoring types performed in different circumstances.

Finally, previous literature is somewhat controversial about the relationship between SATD and source code metrics (Bavota and Russo, 2016; Zampetti et al., 2017), as well as between refactoring actions and metrics improvement (Bavota et al., 2015). In this paper, we want to study whether the presence of refactoring actions in a commit can be statistically explained not only by source code quality metrics but also by the presence of SATD.

Therefore, we state our fourth, and last, research question:

RQ$_4$: *Compared to quality metrics, how does SATD removal explain the presence of a refactoring action?*

The rationale of this research question is to determine whether, from a statistical point of view, the occurrence of a refactoring correlates not only with the typical indicators of the need for refactoring, *i.e.,* a high value of some source code metrics but also with SATD. By looking at the research question results, we also evaluate the magnitude of these refactoring symptoms. This research question also contributes to mitigating threats to internal validity in the analysis of the relationship between SATD removal and refactoring, by considering other factors, such as quality metrics, that can correlate with refactoring.

2.1. Data extraction and analysis methodology

To perform the analysis, we need to combine, for each project, two datasets. The first dataset – originally provided by Maldonado et al. (2017a) and refined by Zampetti et al. (2018) – consists of a set of methods tagged with design SATD, and the commit ID where the SATD has been removed. More specifically, Maldonado et al. (2017a) identifies the introduction of a SATD by detecting the presence of a SATD-related comment added to classes or methods. The removal is identified by detecting the commit in which the SATD-related comment disappears from the code. Zampetti et al. (2018) have also distinguished cases in which this happens because a class or method has been completely removed.

To avoid any possible noise in the dataset, and since during a preliminary analysis we found some suspicious (false) cases of removal (*e.g.,* a SATD was moved in a different class), we performed a manual inspection aimed at discarding false cases of removals. The analysis was performed by two authors independently, and the cases where there was a different classification were discussed and resolved in a meeting involving two more authors. In the end, from the original dataset of Zampetti et al. (2018), we pruned out 13 cases of false removal.

The second dataset has been produced for this study and consists of all refactoring actions performed in the observed history of the analyzed projects. To this aim, we have executed – on all commits of all branches, excluding merge commits – the RMiner tool (Tsantalis et al., 2018). Specifically, given a commit ID, RMiner produces a list of refactoring actions performed in that commit, specifying the refactoring type and the involved components (*i.e.,* source/target classes and methods).

The inner join of the two datasets provides information about refactoring actions that occurred together with SATD removals, whereas the left join between the refactoring dataset and the SATD removal dataset identifies the whole set of commits in which there are no SATD removals while there exist refactoring actions.

Table 1 reports, for each project, the number of branches, and the number of commits analyzed, as well as the total number of SATD, the number of SATD removal commits, and the number of commits in which at least one refactoring action occurred.

To address **RQ**$_1$, we use Fisher's exact test (Fisher, 1962) and Odds Ratio (OR) to compare the proportion of refactorings that occurred together with SATD removals and with other commits. Specifically, we test the null hypothesis:

> H_{01}: there is no significant difference between the proportion of refactoring actions that occurred together with SATD removals and with other commits.

An OR greater than one indicates that refactoring actions have higher chances to occur with SATD removals than with other commits.

Then, we repeat the analysis for two different scenarios (i) one in which we do not consider renaming refactoring actions, and (ii) one in which we consider only cases in which refactoring actions occurred in the same class where the SATD was removed. The rationale of the former is that, while renaming could certainly be used to improve readability (Arnaoudova et al., 2014), they may inflate the results. As for the latter, we assume that refactoring actions occurring in other files are likely to be unrelated to the SATD removal. While we are aware that a refactoring performed on the same class may or may not relate to the SATD, it may be likely that multiple changes churn occurred in the same class within the same commit can be either related or, at least, can be part of the same intervention. Therefore we assume that such changes are likely to be related. Nevertheless, we leverage the qualitative analysis of RQ$_2$ to determine more accurately whether this is the case or not.

To address **RQ**$_2$, we complement the quantitative analysis of RQ$_1$ with a manual analysis of the detected SATD/refactoring co-occurrences. The goal of this manual analysis is to perform an in-depth investigation of whether the refactoring actions actually contribute to the removal of SATD. In particular, we analyze all the cases where there was a co-occurrence between the removal of SATD and refactoring. We considered instances of all commits where there was at least one refactoring action in the same file where SATD occurred, therefore considering a total of 201 cases of all projects.

Therefore, we evaluated the presence of refactoring where there had been removal, analyzing the removed SATD comment as well as changes (in particular refactorings) that occurred in the SATD-affected source code. To ease this analysis, we relied on the GitHub diff visualizer, as well as in the RMiner refactoring visualization plugin.[1] Each SATD removal has been evaluated along four dimensions: (i) whether the SATD-affected code and the refactored code overlap; (ii) whether, according to the SATD comments, the TD was about maintainability or readability (hence, more likely to be removed through a refactoring); (iii) whether the refactoring actually contributed to the SATD removal; or (iv) if the latter did not happen, whether the refactoring was a consequence of the SATD removal (*e.g.,* variables were renamed as a consequence of the SATD removal, or the SATD removal made an extract method necessary).

The manual analysis has been performed by all four authors, and, in particular, each SATD removal was scrutinized by two annotators. In the first phase, all four annotators jointly annotated 10 cases, to reach a consensus on the classification criteria. Then, they annotated the remaining cases separately. During the manual validation, we found some false cases of removals, more specifically 15 cases in which we realized that the SATD comment was not removed, but was moved elsewhere in the source code, or was only partially deleted.

The Krippendorff's alpha (Krippendorff, 2012) inter-rater agreement was 0.73 for (i), 0.78 for (ii), 0.72 for (iii), and 0.79 for (iv). Hence, in all cases, the annotators achieved a substantial agreement. Overall, out of $201 - 13 = 188$ annotated removals, the annotators disagreed on at least one dimension for 44 instances (23%). Finally, the four annotators discussed and resolved the cases with conflicting classification.

To answer **RQ**$_3$, we report the number of different types of refactoring actions that occurred on methods where the SATD was removed. We then perform a statistical comparison among the different types of refactoring to determine the ones that contribute more, in proportion to SATD removal. More specifically we use the *pairwise.prop.test* function in R. This function performs a proportion test followed by post-hoc analysis. The null hypothesis being tested is:

> H_{02}: there is no significant difference between the proportion of SATD removals that co-occurred across different types of refactoring actions.

Since the post-hoc analysis performs multiple comparisons, the *pairwise.prop.test* function adjusts *p*-values using the Benjamini–Hochberg procedure (Yoav and Yosef, 1995).

Finally, to address **RQ**$_4$, we investigate whether, from a quantitative perspective, the presence of a SATD removal in a commit correlates with refactoring in a model comprising also other quality indicators.

Specifically, we build a logistic regression mixed-effect generalized linear model (GLM), where the dependent variable captures, for each commit, whether it involves at least one refactoring action. The random effect is represented by the project ID, and helps to account for differences between projects in terms of metrics, the presence of SATD, and refactoring practices. The independent variables are the presence of a SATD removal (Boolean) and a pool of quality metrics, including Chidamber and Kemerer metrics (Chidamber and Kemerer, 1994):

- Weight Method Count per Class (WMC);
- Response For a Class (RFC);
- Depth of Inheritance Tree (DIT);
- Coupling Between Objects (CBO);
- Lack of Cohesion of Methods (LCOM); and

[1] https://github.com/tsantalis/RefactoringMiner#chrome-extension.

Table 1
Characteristics of the studied projects.

Project (short name)	Branches	Commits	SATD	SATD removal commits	Refactoring related commits
apache/camel (Camel)	49	1,359,764	1282	441	20,429
gerrit-review/gerrit (Gerrit)	15	148,227	150	57	27,691
apache/log4j (Log4j)	7	13,928	113	37	1,759
apache/tomcat (Tomcat)	1	420,244	1184	299	2,353

- Non-Commented, non-empty Lines of Code (LOC).

All the aforementioned metrics have been extracted from the source code using the *ck* tool by Aniche (2015).

To avoid multi-collinearity, we use the *R* (R Core Team, 2012) *redun* function of the *Hmisc* package (Harrell, 2017) for removing redundant variables. The *redun* function stepwise removes variables, starting from the most predictable one until no variable can be predicted with an adjusted R^2 greater than a given threshold (0.8 in our study). It is important to point out that we use the whole dataset to perform correlation analysis because we intend to build an explanatory model and not a predictive model.

As a result of this analysis, the *redun* function discarded the WMC and RFC variables, retaining all the remaining ones.

Since the value of our independent variables can depend on projects' characteristics, and to properly interpret the importance of each variable in the model, we normalize variable values, within each project, in the interval [0, 1]. This is done by subtracting the minimum and dividing by the difference between the maximum and minimum.

After that, we build the mixed-effect generalized linear model using the *glmer* function of the *lme4* (Bates et al., 2015) R package.

To answer **RQ$_4$**, we report the details of the model, among others the coefficient of each factor in the model, and the *p*-value indicating whether the factor is statistically significant or not (for a significance level of 95%). We also report the odds ratio (OR) which, for a logistic regression model, is given by e^{c_i} where c_i is the coefficient of the *i*th factor. An OR $>$ 1 indicates that a unity increase of variable increases of OR times the chances of a refactoring action to occur.

3. Empirical study results

This section reports the results achieved in our study aimed at addressing our four research questions.

3.1. To what extent do refactoring actions co-occur with SATD removals?

Table 2 reports the percentage and the absolute number of commits in which at least one refactoring action was performed. The table compares the set of commits where SATD was removed (first column), with all other commits (third column). The second and fourth columns show the same comparison without considering the refactoring activities related to renaming. Looking at the table it is possible to state that the percentage is substantially higher – about four times higher – for SATD removals than for other commits. In particular, commits where SATD removals occur have a greater chance to contain refactoring actions than other commits.

This result is also confirmed by Fisher's exact test results, reported on the left side of Table 3. Since all (adjusted) *p*-values are < 0.05 for each project, and the OR varies between 3.04 and 5.99, we can state that for all projects the observed differences are statistically significant. In summary, we can conclude that commits where SATD removals occur have 3 to 6 higher odds to also contain refactoring actions than other commits.

Since RMiner also detects refactoring actions dealing with the name of the software components (*e.g.,* classes, methods, variables), namely *Renaming* refactoring, (besides more complex and substantial refactoring actions), it is possible that this would inflate or possibly alter our results. Clearly, this may be debatable, because also source code lexicon improvement could be useful to improve readability and maintainability (Lawrie et al., 2007; Marcus et al., 2008), and therefore identifier renaming could be performed (Arnaoudova et al., 2014).

To determine whether renaming actions could change the results, we also compute results without considering them. These results are reported in the column "Without Rename" of Table 2. Although for some projects, *i.e.,* Tomcat and Log4j, the percentage of *Renaming* is higher than other types of refactoring actions, we can state that our statement "refactoring actions occur more together with SATD removals than in other commits" is still valid. As the right-side of Table 3 shows, also in this case, Fisher's exact test always reports statistically significant results, with OR ranging between 1.57 (Tomcat) and 5.02 (Log4j).

While the results shown so far indicate that refactoring actions highly *co-occur* with SATD removals, this may or may not imply that refactoring actions actually *contribute* to the SATD removal. Table 4 reports, for each project, among the whole set of commits

Table 2
Commits with refactoring actions: SATD removals vs. other commits.

Project	SATD		Baseline	
	All refactoring	Without rename	All refactoring	Without rename
Camel	226 (51%)	111 (33%)	204,071 (15%)	108,781 (8%)
Gerrit	33 (58%)	24 (50%)	27,655 (19%)	16,972 (12%)
Log4j	15 (41%)	8 (27%)	1744 (13%)	882 (7%)
Tomcat	83 (28%)	43 (17%)	2267 (11%)	1374 (7%)

Table 3
Proportion of refactoring actions in SATD removal commits vs. other commits: Fisher's exact test *p*-values and OR.

Project	All refactorings		Without renamings	
	p-value	Odds ratio	p-value	Odds ratio
Camel	$2.20\ e^{-16}$	5.92	$2.2\ e^{-16}$	2.91
Gerrit	$5.52\ e^{-11}$	5.99	$9.21\ e^{-7}$	4.35
Log4j	$1.85\ e^{-5}$	4.76	$6.70\ e^{-4}$	5.02
Tomcat	$6.22\ e^{-15}$	3.04	0.009	1.57

Table 4
Percentage of SATD removals co-occurring and not with refactoring operations performed within the same class.

Project	Co-occurring	Not co-occurring
Camel	867 (9%)	8532 (91%)
Gerrit	138 (33%)	275 (67%)
Log4j	54 (31%)	119 (69%)
Tomcat	9102 (60%)	6083 (40%)

Table 5
Proportion of refactorings on classes that also involve SATD-affected code: exact Fisher test results.

Project	p-value	Odds ratio
Camel	$1.92\ e^{-7}$	2.07
Gerrit	0.64	1.27
Log4j	1.00	1.05
Tomcat	0.91	0.95

Table 6
SATD removals with explicit cases of refactoring documentation in the commit message.

Project	Commits	(%)
Camel	21	(18%)
Gerrit	3	(19%)
Log4j	2	(25%)
Tomcat	11	(25%)
Overall	37	(22%)

involving both the SATD removal and at least one refactoring action, in how many cases the two different operations involve the same code component (*i.e.,* the refactoring action has been performed on the same class containing the SATD being removed in that commit) as opposed to those cases, where the SATD removal affects a class that has not been refactored in that specific commit.

Going deeper on the results highlighted in Table 4, it is possible to notice that the percentage of cases in which refactoring actions are done on the same source code components in which the SATD removal occurs varies in the range [9–60]% for Camel and Tomcat respectively. More specifically, only for Tomcat, we have a majority of refactoring actions that happen on SATD-affected code but, as discussed in RQ_1, Tomcat is also a project for which the majority of refactoring actions deal with renaming.

For the remaining three projects, instead, only a minority of refactoring actions occur on SATD-affected source code. Based on this result we conjecture that within the same commit a developer may perform several actions aimed at improving the overall software quality including both (i) SATD-removals in some source code components and (ii) refactoring actions on a different set of source code components.

Results of Fisher's exact test, shown in Table 5, indicate that only for Camel there is a statistically significant difference (p-value < 0.05) with SATD-affected components having chances (OR $= 2.07$) to undergo refactoring actions than other source code components. Moreover, the difference is marginally significant for Gerrit (p-value $= 0.64$) with OR $= 1.27$.

RQ_1 summary: in general, refactoring actions have significantly higher odds to occur together with SATD removals. This conclusion is still valid if renaming actions are discarded.

3.2. To what extent do refactoring actions actually contribute to SATD removal?

As shown in Table 6, the manual analysis of SATD-removal commits involving refactoring actions within the same class where SATD was removed revealed that only in 22% of the cases there is an explicit reference to a refactoring.

For example, a Tomcat commit removing SATD[2] mention *"refactor to optimize its behavior"*. Finally, in some cases the commit message[3] *"CAMEL-1651: Fixed gzip content encoding only being applied if really ……needed. Polished the code as well."*) refers to the case of floss refactoring, *i.e.,* when, upon removing a SATD, refactoring actions were performed together with other changes.

While, as shown in Table 6, we have observed a small yet not negligible percentage of commits where the refactoring was explicitly documented, we then performed an in-depth analysis of the link (if any) between the SATD removal and the refactoring action(s).

Table 7
Results of the manual validation.

	#	%
1. SATD-affected and refactored code overlap	77	41.62%
2. SATD is about maintainability/readability	39	21.08%
3. If 2 is YES, the refactoring likely addressed it	14	7.56%
4. If 2 is NO or 3 is NO, the refactoring is consequence of SATD removal	41	22.16%

Table 7 reports the number (and percentage) of cases in which (i) there is an overlap between source code involved in at least one refactoring action and SATD removal, (ii) the SATD comment refers to maintainability or readability issues, (iii) the SATD comment referring to maintainability or readability issues is (likely) related to the refactoring being performed, and (iv) the refactoring is a consequence of the SATD removal. As reported in Table 7, in 42% of cases the refactoring affects the same source code lines related to the SATD. In 41 cases, the refactoring is a direct consequence of the SATD removal. This may be expected, and it is in line with previous research finding how most refactorings are "floss", *i.e.,* they occur as a consequence of other changes (Murphy-Hill et al., 2012). At the same time, it is possible to note that the SATD comment a few times refers to maintainability or readability issues (21%), and out of 39 cases in which this occurs, refactoring is directly related to addressing such a problem in 14 cases only.

In the following, we report and discuss some examples, belonging to the different categories mentioned above.

In Camel, a commit[4] that removed a SATD comment reporting the need for splitting a long method having more than one responsibility (*i.e.,* "refactoring TODO below in a separate method"). This is a clear example of SATD referring to a maintainability issue addressed by applying an extract method operation. The latter implies that the refactoring is specifically targeted to address the SATD comment. Moreover, the commit message highlights how the SATD removal was the main goal of the change: "...split a big method into 2 so its easier to deal with".

Looking at those cases where the refactoring occurs as a consequence of the SATD removal, we found a case in Camel[5] where the extract method operation was performed as the changes addressing the SATD comment "// TODO: like response object with type refer to model" made it necessary, in particular for restructuring a too complex source code fragment.

At the same time, there are also, as the table reports, many cases in which the SATD removal and the refactoring are unrelated. For example, in Gerrit, we found a[6] change in which there is a refactoring operation involving renaming a method. This modification is carried out precisely on the method object of the removal of the SATD, but it does not appear that the two activities have a connection, because this type of refactoring does not solve the technical debt present in the source code, since the comment in support of the SATD does a reference to a problem that is not solved. Therefore the two activities in this case, as in others, simply co-occur.

Finally, we found cases in which refactoring and TD removal have nothing in common, starting from the code on which the two activities are carried out, up to the reason for which they are performed, and also the supporting comment of the SATD does not refer to issues of maintainability or readability of the source code. For example, in Log4j, in commit[7] the SATD comment reads:

2 https://github.com/apache/tomcat/commit/cb69c4a0.
3 https://github.com/apache/camel/commit/2769e150.
4 https://github.com/apache/camel/commit/624bc4a7.
5 https://github.com/apache/camel/commit/1817a41ea9165453.
6 https://github.com/apache/tomcat/commit/fd94c05e.
7 https://github.com/apache/log4j/commit/bffbaa46.

Table 8
Percentages of refactoring actions of different types contributing to SATD removal.

Refactoring type	Camel	Gerrit	Log4j	Tomcat	Overall
Move operation	4.38	44.20	1.85	53.71	49.10
Move attribute	2.77	23.19	7.41	39.10	35.62*
Pull up operation	19.15	1.45	1.85	1.24	2.78*
Pull up attribute	23.88	0.72	0.00	0.70	2.68
Rename Method	11.65	1.45	14.81	0.82	1.83*
Extract and move op.	1.04	0.72	1.85	1.58	1.53
Extract operation	13.03	12.32	22.22	0.05	1.45
Extract class	0.92	3.62	0.00	0.92	0.95*
Rename attribute	3.92	0.00	7.41	0.62	0.93
Rename variable	6.34	0.72	7.41	0.27	0.84
Inline operation	0.58	3.62	7.41	0.64	0.71
Rename parameter	4.61	2.17	14.81	0.12	0.61
Extract variable	2.54	2.17	0.00	0.10	0.33*
Parameterize variable	1.27	3.62	0.00	0.00	0.16*
Rename class	1.27	0.00	0.00	0.04	0.15
Inline variable	1.38	0.00	0.00	0.02	0.14
Replace var. with attr.	0.12	0.00	12.96	0.02	0.10
Move rename class	0.58	0.00	0.00	0.00	0.05
Push down operation	0.46	0.00	0.00	0.00	0.04
Move class	0.12	0.00	0.00	0.01	0.02
Extract superclass	0.00	0.00	0.00	0.01	0.01

"TODO need to work out how to suspend the DocumentChangeListener reFilter temporarily while this bit updates", therefore it refers to how to temporarily suspend a filter during the update, so it is clear that this is not about maintainability or anything like that. Furthermore, although RMiner has detected, for this commit, several refactoring activities, none of them have been performed on source code lines directly affected by the TD.

RQ_2 summary: In 42% of the cases, there is a co-occurrence between SATD removals and refactoring actions on the same source code. However, Only in a small minority of cases (21%) the SATD removal comment refers to maintainability or readability issues, and in 8% the refactoring actions directly address these issues. Finally, for 22% of the total cases, the refactoring is performed as a consequence of the SATD removal.

3.3. What are the types of refactoring actions that co-occur more with SATD removals?

To address RQ_3 we look, with a deeper level of detail, at the refactoring actions occurred in classes where SATD has been removed. Table 8 reports, for each project the percentage of refactoring actions of different types contributing to SATD removals.

The first thing that one can notice from Table 8 is that renaming refactoring actions do not occur so much in classes where SATD has been removed. Although in previous research questions we have removed their effect from the total set of refactoring (because they may be very frequent), and while renaming could in principle be used to address some SATD (*e.g.,* to improve source code readability or make identifiers more consistent with other artifacts or other source code components), in the end, results show that such refactoring co-occurs with a minority of SATD removals.

For Camel, the refactoring types having the largest proportion are, interestingly, Pull Up Attribute ($\simeq$ 24%) and Pull Up Operation ($\simeq$ 19%), *i.e.,* refactoring actions dealing with inheritance. In Gerrit, the most frequents are Move Operation ($\simeq$ 44%) and Move Attribute ($\simeq$ 23%), while in Log4j there is a high percentage of Extract Operation actions ($\simeq$ 22%), besides Tomcat – which is the project having the highest number of refactoring actions being detected, as reported in Table 1 – SATD removal almost exclusively happens through Move Operation ($\simeq$ 54%) and Move Attribute ($\simeq$ 39%).

Table 8 does not list some types of refactoring – Extract subclass, Extract interface, Move source folder, Push down attribute, Rename package, Replace attribute – because no operations of this type were identified in any of the projects under analysis.

Table 8 also highlights the proportion test results. An asterisk indicates that the difference between a proportion and the previous one is statistically significant. More specifically, the differences are statistically significant for the pairs Move Operation & Move Attribute, Move Attribute & Pull up operation, Pull up attribute & Rename Method, Extract Operation & Extract Class, Rename Parameter & Extract Variable, Extract Variable & Parameterize Variable. Overall, the most frequently occurring cases of refactoring seem to be the ones related to moving methods and attributes from one class to another, within the same hierarchy or not.

In addition to that, if we give a broad look at the refactoring types reported in Table 8, and relate them with findings of previous work, we can notice that, except for Tomcat, there is a relatively high percentage of Extract Operation refactorings. These can reflect activities aimed at reducing method size and complexity, and increasing their cohesion. Indeed, previous work reported how structural metrics like LOC and Cyclomatic complexity are good predictors for SATD (Zampetti et al., 2017).

Also, in RQ_1 we noticed how Renaming actions co-occur frequently with SATD removal. However, when restricting our attention only to SATD-affected classes, their percentage remains (for some renaming types) relatively high only for Camel and Log4j. Such renamings, *e.g.,* of methods and operations reflect cases in which previous work has shown how changes to method signatures are part of the SATD removal (Zampetti et al., 2018).

RQ_3 summary: The type of refactorings performed when removing SATD varies from one project to another, however, they mostly concern extracting and moving operations.

3.4. Compared to quality metrics, how does SATD removal explain the presence of a refactoring action?

Table 9 reports the results of the logistic regression mixed-effect model. The top-side of the table reports the model diagnostics (Akaike Information Criterion – AIC (Akaike, 1973), Bayesian Information Criterion – BIC, log likelihood, deviance, and degree of freedom residuals), the scaled residuals, and the random effect (project estimate). The bottom part of the table reports the OR, estimate, standard error, z-value, and p-value for the various factors we considered.

As highlighted in Table 9, all the quality metrics analyzed (CBO, DIT, LOC, LCOM), as well as the presence of SATD removal, have a statistically significant effect on the likelihood that the commit applies a refactoring action. However, by observing the estimates and consequently the OR, a unit increase of the coupling between objects (CBO) increases by about 113 times the odds that the commit applies a refactoring operation. The above result is not surprising considering that, as stated by Fowler (1999), a well-designed project has high cohesion inside each component and low coupling between components. At the same time, commits adding new lines of code have about 70 times more chances of being subject to refactoring operations. Finally, while looking at the SATD removal, we can state that commits in which there is a removal of a previously introduced SATD comment have 1.67 higher odds of involving also refactoring actions. Quite surprisingly, a unit increase of the LCOM does not increase ($OR < 1$) the likelihood that the commit applies also a refactoring action.

Table 9
RQ_4: Results of the logistic regression mixed-effect model.

Diagnostics					
	AIC	BIC	logLik	deviance	df resid.
	9522.0	9603.4	−4754.0	9508.0	835,318
Scaled residuals					
	Min	1Q	Median	3Q	Max
	−0.601	−0.028	−0.022	−0.015	82.435
Random effects					
Groups name	Variance	Std.Dev.			
Project name (Intercept)	0.7768	0.8814			
Quality metrics					
Metric	OR	Estimate	Std. error	*z*-value	*p*-value
(Intercept)	0.00	−7.73	0.43	−18.17	<**0.01**
cbo	113.00	4.73	0.40	11.74	<**0.01**
dit	0.33	−1.12	0.76	−1.48	0.14
lcom	0.27	−1.31	0.59	−2.22	**0.03**
loc	69.83	4.25	0.58	7.31	<**0.01**
SATD	1.67	0.51	0.09	5.50	<**0.01**

RQ_4 summary: Even if quality metrics such as CBO and LOC have a statistically significant effect and a very high OR on the likelihood that the commit contains a refactoring action, the presence of a SATD removal still plays a role, *i.e.,* 1.67 higher odds that the commit contains a refactoring action.

4. Threats to validity

Threats to *construct validity* concern the relationship between theory and observation. Such threats mostly concern possible imprecision in our measurements. While the SATD removals belong to a dataset already used and validated in different papers (Maldonado et al., 2017b; Zampetti et al., 2018), we considered refactoring actions as detected by the RMiner tool. However, Tsantalis et al. (2018) reported for it high precision ($\simeq$ 98%) and recall ($\simeq$ 87%) values. Also, we cannot exclude that the removal dataset could be error prone, although we pruned out some false positives as explained in Section 2.1.

Threats to *internal validity* concern factors internal to our study that can influence the results. In particular, we cannot, in general, claim a cause–effect relationship between refactoring actions and SATD removal. We mitigate this threat by (i) analyzing (RQ_1) refactorings that occurred on SATD-affected source code (same classes and methods), and (ii) by performing a manual validation of such co-occurrences (RQ_2). In RQ_4, our mixed-effect model considered a pool of metrics that have been found to change in the context of refactoring actions (Bavota et al., 2015). Nevertheless, it is possible that there could be other indicators of the need for refactoring which we did not consider.

Threats to *conclusion validity* concern the relationship between experimentation and outcome. We used appropriate statistical tests such as the Fisher's exact test for RQ_2, multiple proportion tests with post-hoc analysis and Benjamini–Hochberg correction in RQ_3, and effect size measure (OR) to support our findings. As for RQ_4, we used a mixed-effect model, in order to account for the random effect of projects. Finally, it is possible that RQ_2 results are due to an agreement by chance. We mitigate this threat by computing Cohen's *k* inter-rater agreement (Cohen, 1960), which resulted to be moderate (0.52).

Threats to *external validity* concern the generalizability of our findings. This study has been conducted on data from four projects from the Maldonado et al. dataset (Maldonado et al., 2017a). Rather than conducting a large study on projects where SATD comments have not been validated, we preferred to rely on a curated SATD dataset. Clearly, further results might be confirmed or contradicted when analyzing other projects.

5. Implications

As explained in Section 2, this study has mainly an observational nature. Therefore, its results may or may not directly impact development practices. Having said that, in the following, we summarize our main findings, from which we derive some conclusions which may impact developers' activities.

First, we found that there is a higher chance for refactoring actions to occur together with SATD removals than with other changes. This means that SATD removals are more likely to co-occur with commits containing other maintainability improvement changes (and specifically refactoring actions) than with other changes not containing refactoring actions. This already indicates that despite previous work indicated the predominance of floss refactoring actions (Murphy-Hill et al., 2012), *i.e.,* of refactoring actions performed with other changes related to bug fixing, enhancement, or feature addition, whenever possible developers dedicate some tasks for cleaning-up code and improving its maintainability. This, among others, includes removing SATD (Maldonado et al., 2017a; Zampetti et al., 2018), performing refactoring actions (Bavota et al., 2015), or improving source code readability (Johnson et al., 2019).

While previous literature suggests that developers perform refactoring as a consequence of other changes, it is not all about floss refactorings: our results highlight, at least for the analyzed projects, the predominance of commits where developers focus on software quality improvement. This implicitly suggests developers regularly plan such activities.

When restricting to cases where SATD removal and refactoring actions co-occur in the same file, we notice how these represent a minority of the actual co-occurrences. Further, when manually inspecting the changes, we found that, while in $\simeq$ 42% of the cases the SATD-affected code and the refactored code overlap, only $\simeq$ 21% of the SATD are maintainability-related, and only in 8% of the inspected cases refactoring actions were performed to resolve maintainability-related SATD. Instead, there is a 22% of cases for which, while refactoring actions are not targeting the problem highlighted in the SATD comment, they are a consequence of its removal, being, therefore, floss refactoring actions (Murphy-Hill et al., 2012).

Moreover, we found that, while also remarked in previous work, SATD removals are rarely documented (Zampetti et al., 2018), there are cases for which the commit message mentions that a refactoring activity has been performed, and it also relates to the SATD comment being removed. On the one hand, this further confirms that co-occurrences dominate over causal-relationship between refactoring and SATD removal, on the other hand, this may also entail further work in the area of automated commit documentation (Jiang et al., 2017; Liu et al., 2018).

Therefore, the obtained results indicate that SATD removal is only in a few cases related to maintainability improvements through refactoring actions. Researchers focusing on developing approaches for TD automated documentation or automated suggestions for TD removal (Zampetti et al., 2018, 2020) must also focus on other actions changing the program's behavior, *e.g.,* changing APIs or pre-and-post conditions.

SATD is much more beyond maintainability improvement, and therefore beyond refactoring. When planning for it, developers should put into account different kinds of actions, including improving code robustness or consider API migrations.

Looking at cases for which the refactoring occurs in the SATD affected source code, we found that the type of refactoring actions varies, with a general predominance of actions aimed at moving methods and attributes, or at extracting methods from long/complex or not cohesive ones. The latter contributes to explaining the results of previous work by Zampetti et al. (2017), where source code metrics were used to predict where SATD should be admitted, and where metrics related to code length and complexity were considered good predictors for TD admission.

Confirming previous literature results, refactoring mainly addresses SATD for those cases in which the source code is considered overly long and complex. Therefore, when SATD is maintainability-related, developers may mainly try to address it by moving code elements through refactoring actions such as move method, pull-up method, or extract method.

6. Related work

This section reports the literature related to (i) detection of TD focusing more on SATD, and (ii) investigation of refactoring effects on SATD and quality metrics.

6.1. Detection of SATD

Potdar and Shihab (2014b) conducted a qualitative analysis on the TD in the source code of open-source projects and observed that developers often "self-admit" the technical debt by inserting comments indicating that the code is temporary and will need to be reviewed in the future. Furthermore, after a manual analysis of the code, they identify 62 different comment schemes indicating SATD. They showed that in software projects SATD is very common and that it is introduced mainly by experienced developers.

Maldonado and Shihab (2015) have developed an approach that allows identifying SATD instances in the code through comments posted by the developers. The proposed approach is based on model matching and classifies the SATD into five types: design, defect, documentation, requirements, and tests. Bavota and Russo (2016) analyzed 159 software projects to study the spread and evolution of self-admitted technical debt and its relationship with software quality. They pointed out that self-admitted technical debt is very widespread and tends to stay long in the code, and also increases over time due to the introduction of new instances that were not resolved by the developers. Similar to previous work by Griffith et al. (2014), they did not find a clear relation between SATD and some software quality metrics, in particular, WMC, CBO, and Buse and Weimer Readability metrics.

Wehaibi et al. (2016) examined the relationship between technical debt and software quality in order to understand whether files with self-admitted technical debt present more defects than files without self-admitted technical debt if the technical variations of the debt allowed introducing future defects and if the technical changes related to the self-admitted debt tend to be more difficult. The results of this study show that although technical debt can have negative effects, its impact is not related to defects, rather it makes the system more difficult to modify in the future.

Maldonado et al. (2017a) conducted an empirical study to examine how much SATD is removed, for how long this SATD remains within a project, and who removes that debt. They highlighted that most of the self-admitted technical debt is removed, that the self-admitted technical debt is mostly self-removed, and that it lasts on average between 82 and 613.2 days in a project before it is removed. They also pointed out that most removals occur due to bug fixes and that there is no formal process to remove the self-admitted technical debt.

Zampetti et al. (2018) conducted a depth quantitative and qualitative study to understand how SATD is removed in the source code. On the one hand, they assessed whether the SATD is accidentally removed and on the other hand to what extent the removal of the SATD is documented. Therefore, they have deepened the study of the relationship between the removal of comments that document SATD and the related changes to the source code, highlighting through their results that a large percentage of removal of SATD comments occurs accidentally when the whole method is removed. Furthermore, the removal of SATD is documented in commit messages only in 8% of cases.

We share with all the aforementioned articles the objective of observing how TD, in particular SATD, is managed to understand if we can consider their actual removal. Furthermore, we rely on the results of Zampetti et al. (2018), because we used their dataset consisting of a set of methods labeled with SATD at the design level and the commit ID in which the SATD was removed.

6.2. Refactoring effects on SATD and quality metrics

Stroggylos and Spinellis (2007) examined whether software metrics are influenced by the developers' refactoring activities. The results indicated a significant worsening of some metrics. In particular, it seems that refactoring has caused an increase in metrics such as LCOM and RFC.

Shatnawi and Li (2011) assessed the effect of refactoring activities on four software quality factors: reusability, flexibility, extensibility, and effectiveness, discovering that not all refactoring activities improve quality factors and even that some make them worse. They also defined a refactoring heuristic that can help developers follow an appropriate refactoring process.

Bavota et al. (2015) conducted an empirical study aimed at investigating whether refactoring activities occur in classes for which some indicators, such as quality metrics or presence of code smells, suggest that refactoring may be necessary. The study conducted on 63 releases of three projects indicated that, often, quality metrics do not show a clear relationship with refactoring.

Some studies have also considered the impact of refactoring on software metrics. In particular, Chaparro et al. (2014) proposed a technique named RIPE (Refactoring Impact PrEdiction) to assess the impact of refactoring operations on source code quality metrics. On a different side, Cinnéide et al. (2012) studied the impact of automatically-recommended refactoring on source code metrics, and found that cohesion metrics agree on only a minority of these refactoring actions.

While the aforementioned studies focused on the relationship between refactoring actions and quality indicators, to the best of our knowledge this is the first paper contributing in terms of studying the relationship between refactoring actions and SATD removal.

6.3. Empirical studies on refactoring

While the focus of this paper is mainly on SATD removal, and how much removal can be achieved through refactoring actions, several authors have investigated refactoring practices. This has been done through various kinds of empirical studies, including field studies in industry (Kim et al., 2012), by contacting open-source developers (Silva et al., 2016), or using Integrated Development Environment usage data (Murphy-Hill et al., 2012). Such studies are orthogonal to our investigation, which focuses, instead, on a specific purpose of refactoring (SATD removal). At the same time, some results of previous research can be related to our findings:

- Two main goals of refactoring are improving source code readability and maintainability (Kim et al., 2012). Both goals relate to some SATD removal, although SATD goes beyond that;
- If looking at some refactoring types that co-occur a lot with SATD removal (*e.g.,* extract method), previous work found that this is mainly done to favor reuse (Silva et al., 2016) than to address a specific SATD. This, once again, stresses the finding that what we are observing is in most cases a co-occurrence and not a cause–effect relationship
- documented cases of commits related to source code refactoring or, in general, improvement, represent cases of root-canal refactoring which is however less common than floss refactoring, *i.e.,* refactoring performed in the context of bug fixing of feature addition/improvement (Murphy-Hill et al., 2012).

7. Conclusions and future work

This paper investigated the extent to which refactoring actions co-occur with SATD removal, and then looked at the extent to which this was just a co-occurrence, or whether the refactoring action actually contributed to the SATD removal. Then, we looked at types of refactoring actions co-occurring more with SATD removals and quantitatively studied whether, compared with quality metrics, SATD removals explain the presence of refactoring actions in a commit.

The study has been performed on data from four open-source Java projects, *i.e.,* Camel, Gerrit, Log4j, and Tomcat, leveraging an already available SATD removal dataset (Maldonado et al., 2017a) and automatically detecting refactorings using an automated tool (Tsantalis et al., 2018).

From a quantitative point of view, results indicate how refactoring actions tend to co-occur more with SATD removals than with other changes, with some refactoring types (*e.g.,* move operations and attributes) being the most frequent. Moreover, the results of the logistic-regression mixed-effect model highlight that the presence of a SATD removal has a significant statistical effect on the likelihood that the commit applies a refactoring action, together with quality metrics, like CBO and LOC, which previous work correlated to refactoring actions (Bavota et al., 2015). This means that, even with a small effect-size, SATD removals statistically correlate with refactoring actions as well as other metrics do.

At the same time, by looking deeper at the refactoring actions, we found that only a minority of them actually contribute to the SATD removal, while the remaining ones are simple co-occurrences, or cases in which the refactoring was a consequence of the SATD removal. This, on the one hand, indicates that refactoring has some contribution in SATD removal, however, the majority is still related to other changes, such as adding/changing conditions or changing APIS (Zampetti et al., 2018, 2020). On the other hand, this highlights the presence of combined software quality activities (sometimes also documented in commit messages) where developers address SATD, apply some refactoring actions, and perform other quality improvement tasks.

Future work aims at extending this work in several directions. In particular, we aim to perform an in-depth quantitative and qualitative analysis of commits aimed at performing general quality improvement activities (*e.g.,* those in which we observed a co-occurrence of refactoring activities and SATD removals), with the aim of determining (i) when they are performed, *e.g.,* together with feature additions or in periods in which the project evolution is fairly stable, and (ii) what other quality characteristics are addressed, including metric profiles, code smells, and static analysis warnings.

CRediT authorship contribution statement

Martina Iammarino: Software, Validation, Formal analysis, Investigation, Data curation, Writing - original draft. **Fiorella Zampetti:** Software, Methodology, Validation, Formal analysis, Data Curation, Writing - original draft. **Lerina Aversano:** Conceptualization, Validation, Formal analysis, Writing - original draft. **Massimiliano Di Penta:** Conceptualization, Methodology, Formal analysis, Writing - original draft.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

Akaike, H., 1973. Information theory and an extension of the maximum likelihood principle. In: 2nd International Symposium on Information Theory.

Alshayeb, M., 2009. Empirical investigation of refactoring effect on software quality. Inf. Softw. Technol. 51, 1319–1326.

Aniche, M., 2015. Java code metrics calculator (CK). Available in https://github.com/mauricioaniche/ck/.

Arnaoudova, V., Eshkevari, L.M., Di Penta, M., Oliveto, R., Antoniol, G., Guéhéneuc, Y., 2014. REPENT: analyzing the nature of identifier renamings. IEEE Trans. Softw. Eng. 40, 502–532.

Bates, D., Mächler, M., Bolker, B., Walker, S., 2015. Fitting linear mixed-effects models using lme4. J. Stat. Softw. 67, 1–48.

Bavota, G., De Lucia, A., Di Penta, M., Oliveto, R., Palomba, F., 2015. An experimental investigation on the innate relationship between quality and refactoring. J. Syst. Softw..

Bavota, G., Russo, B., 2016. A large-scale empirical study on self-admitted technical debt. In: International Conference on Mining Software Repositories. ACM.

Chaparro, O., Bavota, G., Marcus, A., Di Penta, M., 2014. On the impact of refactoring operations on code quality metrics. In: 30th IEEE International Conference on Software Maintenance and Evolution, Victoria, BC, Canada, September 29 - October 3, 2014. pp. 456–460.

Chatzigeorgiou, A., Manakos, A., 2014. Investigating the evolution of code smells in object-oriented systems. ISSE 10, 3–18.

Chidamber, S.R., Kemerer, C.F., 1994. A metrics suite for object oriented design. IEEE Trans. Softw. Eng. (TSE) 20, 476–493.

Cinnéide, M.Ó., Tratt, L., Harman, M., Counsell, S., Moghadam, I.H., 2012. Experimental assessment of software metrics using automated refactoring. In: 2012 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM '12, Lund, Sweden - September 19-20, 2012. pp. 49–58.

Cohen, J., 1960. A coefficient of agreement for nominal scales. Educ. Psychol. Meas..

Cunningham, W., 1992. The wycash portfolio management system. In: Addendum to the Proceedings on Object-Oriented Programming Systems, Languages, and Applications. ACM.

Fisher, R.A., 1962. Confidence limits for a cross-product ratio. Aust. J. Stat..

Fowler, M., 1999. Refactoring: Improving the Design of Existing Code. Addison Wesley.

Griffith, I., Reimanis, D., Izurieta, C., Codabux, Z., Deo, A., Williams, B., 2014. The correspondence between software quality models and technical debt estimation approaches. In: 2014 Sixth International Workshop on Managing Technical Debt. pp. 19–26.

Harrell, Jr., F.E., with contributions from Charles Dupont, many others., 2017. Hmisc: Harrell miscellaneous. URL: https://CRAN.R-project.org/package=Hmisc. r package version 4.0-3.

Hegedüs, P., Kádár, I., Ferenc, R., Gyimóthy, T., 2018. Empirical evaluation of software maintainability based on a manually validated refactoring dataset. Inf. Softw. Technol. 95, 313–327.

Iammarino, M., Zampetti, F., Aversano, L., Di Penta, M., 2019. Self-admitted technical debt removal and refactoring actions: Co-occurrence or more? In: 2019 IEEE International Conference on Software Maintenance and Evolution, ICSME 2019, Cleveland, OH, USA, September 29 - October 4, 2019. IEEE, pp. 186–190. http://dx.doi.org/10.1109/ICSME.2019.00029.

Iammarino, M., Zampetti, F., Aversano, L., Di Penta, M., 2021. An empirical study on the co-occurrence between refactoring actions and self-admitted technical debt removal. http://dx.doi.org/10.5281/zenodo.4628255.

Jiang, S., Armaly, A., McMillan, C., 2017. Automatically generating commit messages from diffs using neural machine translation. In: Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering, ASE 2017, Urbana, IL, USA, October 30 - November 03, 2017. pp. 135–146.

Johnson, J., Lubo, S., Yedla, N., Aponte, J., Sharif, B., 2019. An empirical study assessing source code readability in comprehension. In: 2019 IEEE International Conference on Software Maintenance and Evolution, ICSME 2019, Cleveland, OH, USA, September 29 - October 4, 2019. pp. 513–523.

Kim, M., Zimmermann, T., Nagappan, N., 2012. A field study of refactoring challenges and benefits. In: Proceedings of the 20th International Symposium on Foundations of Software Engineering.

Krippendorff, K., 2012. Content Analysis: An Introduction to its Methodology. Sage.

Lawrie, D.J., Feild, H., Binkley, D.W., 2007. Quantifying identifier quality: an analysis of trends. Empir. Softw. Eng. 12, 359–388.

Liu, Z., Xia, X., Hassan, A.E., Lo, D., Xing, Z., Wang, X., 2018. Neural-machine-translation-based commit message generation: how far are we?. In: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018, Montpellier, France, September 3-7, 2018. pp. 373–384.

Maldonado, da S.E., Abdalkareem, R., Shihab, E., Serebrenik, A., 2017a. An empirical study on the removal of self-admitted technical debt. In: 2017 IEEE International Conference on Software Maintenance and Evolution. IEEE.

Maldonado, E.d.S., Shihab, E., 2015. Detecting and quantifying different types of self-admitted technical debt. In: Managing Technical Debt (MTD), 2015 IEEE 7th International Workshop on. IEEE.

Maldonado, E., Shihab, E., Tsantalis, N., 2017b. Using natural language processing to automatically detect self-admitted technical debt. IEEE Trans. Softw. Eng..

Marcus, A., Poshyvanyk, D., Ferenc, R., 2008. Using the conceptual cohesion of classes for fault prediction in object-oriented systems. IEEE Trans. Softw. Eng. 34, 287–300.

Moser, R., Abrahamsson, P., Pedrycz, W., Sillitti, A., Succi, G., 2007. A case study on the impact of refactoring on quality and productivity in an agile team. In: Balancing Agility and Formalism in Software Engineering, Second IFIP TC 2 Central and East European Conference on Software Engineering Techniques, CEE-SET 2007, Poznan, Poland, October 10-12, 2007, Revised Selected Papers. pp. 252–266.

Murphy-Hill, E.R., Parnin, C., Black, A.P., 2012. How we refactor, and how we know it. IEEE Trans. Softw. Eng. 38, 5–18.

Potdar, A., Shihab, E., 2014a. An exploratory study on self-admitted technical debt. In: Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on. IEEE.

Potdar, A., Shihab, E., 2014b. An exploratory study on self-admitted technical debt. In: International Conference on Software Maintenance and Evolution. IEEE Computer Society.

R Core Team, 2012. R: A Language and Environment for Statistical Computing. ISBN: 3-900051-07-0, URL: http://www.R-project.org.

Shatnawi, R., Li, W., 2011. An empirical assessment of refactoring impact on software quality using a hierarchical quality model. Int. J. Softw. Eng. Appl. 5, 127–150, Cited By 19.

Silva, D., Tsantalis, N., Valente, M.T., 2016. Why we refactor? confessions of github contributors. In: Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2016. pp. 858–870.

Stroggylos, K., Spinellis, D., 2007. Refactoring-does it improve software quality?. In: Proceedings of the 5th International Workshop on Software Quality, WoSQ@ICSE 2007, Minneapolis, Minnesota, USA, May 20, 2007. p. 10.

Szoke, G., Antal, G., Nagy, C., Ferenc, R., Gyimóthy, T., 2017. Empirical study on refactoring large-scale industrial systems and its effects on maintainability. J. Syst. Softw. 129, 107–126.

Tsantalis, N., Mansouri, M., Eshkevari, L.M., Mazinanian, D., Dig, D., 2018. Accurate and efficient refactoring detection in commit history. In: Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018. pp. 483–494.

Tufano, M., Palomba, F., Bavota, G., Oliveto, R., Di Penta, M., De Lucia, A., Poshyvanyk, D., 2017. When and why your code starts to smell bad (and whether the smells go away). IEEE Trans. Softw. Eng..

Wehaibi, S., Shihab, E., Guerrouj, L., 2016. Examining the impact of self-admitted technical debt on software quality. In: Software Analysis, Evolution, and Reengineering (SANER), 2016 IEEE 23rd International Conference on. IEEE.

Yoav, B., Yosef, H., 1995. Controlling the false discovery rate: A practical and powerful approach to multiple testing. J. R. Stat. Soc. Ser. B Stat. Methodol. 57, 289–300, URL: http://www.jstor.org/stable/2346101.

Zampetti, F., Noiseux, C., Antoniol, G., Khomh, F., Di Penta, M., 2017. Recommending when design technical debt should be self-admitted. In: 2017 IEEE International Conference on Software Maintenance and Evolution.

Zampetti, F., Serebrenik, A., Di Penta, M., 2020. Automatically learning patterns for self-admitted technical debt removal. In: 27th IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2020, London, on, Canada, February 18-21, 2020. pp. 355–366.

Zampetti, F., Serebrenik, A., DiPenta, M., 2018. Was self-admitted technical debt removal a real removal?: an in-depth perspective. In: Proceedings of the 15th International Conference on Mining Software Repositories, MSR 2018, Gothenburg, Sweden, May 28-29, 2018. pp. 526–536.

Martina Iammarino is a Ph.D. student in Computer Engineering. She received the master's degree in computer engineering at the University of Sannio in Benevento, Italy. Her research interests include software quality, refactoring, modeling, evaluation and evolution, and software measurement.

Fiorella Zampetti is postdoctoral fellow at the Department of Engineering, University of Sannio, Italy. She received her Ph.D. degree in Information Technology at the University of Sannio, Italy in 2019. She received the Bachelor degree in Computer Science Engineering from University of Sannio in 2012 and the Master degree in Electronic Engineering from University of Sannio in 2015. Her research interests include software maintenance and evolution, mining software repository and empirical software engineering for studying software build systems and software quality. She has authored several publications in top software engineering conferences. She has served/is serving the organizing and program committee of conferences such as SANER'18, MSR'18 challenge, MSR'20 challenge, and ICSME'19 artifact track, ICSE'21 demonstration track.

Lerina Aversano is an associate professor at the Department of Engineering of the University of Sannio Benevento (Italy). She received the Ph.D. in Computer Engineering in July 2003 at the same University and was assistant professor from 2005. She also was a research leader at RCOST – Research Centre On Software Technology – of the University of Sannio from 2005. Her research interests include software maintenance, program comprehension, reverse engineering, reengineering, migration, business process modeling, business process evolution, software system evolution, software quality.

Massimiliano Di Penta is a full professor at the University of Sannio, Italy. His research interests include software maintenance and evolution, mining software repositories, empirical software engineering, search-based software engineering, and service-centric software engineering. He is an author of over 300 papers appeared in international journals, conferences, and workshops. He serves and has served in the organizing and program committees of more than 100 conferences, including ICSE, FSE, ASE, ICSME. He is editor-in-chief of the Journal of Software: Evolution and Processes edited by Wiley. He is in the editorial board of ACM Transactions on Software Engineering and Methodology, and of the Empirical Software Engineering Journal. He has served the editorial board of the IEEE Transactions on Software Engineering.