# Who (Self) Admits Technical Debt?

Gianmarco Fucci, Fiorella Zampetti
University of Sannio, Italy
{name.surname}@unisannio.it

Alexander Serebrenik
Eindhoven University of Technology,
The Netherlands
a.serebrenik@tue.nl

Massimiliano Di Penta
University of Sannio, Italy
dipenta@unisannio.it

***Abstract*—Self-Admitted Technical Debt (SATD) are comments, left by developers in the source code or elsewhere, aimed at describing the presence of TD, i.e., source code "not ready yet". Although this was never stated in the original paper by Potdar and Shihab, the term SATD might suggest that it refers to a "self-admission" by whoever has written or changed the source code. This paper empirically investigates, using a curated SATD dataset from five Java open-source projects, (i) the extent to which SATD comments are introduced by authors different from those who have done last changes to the related source code, and (ii) when this happens, what is the level of ownership those developers have about the commented source code. Results of the study indicate that, depending on the project, the percentage of SATD admissions introduced or changed without modifying the related source code varies between 0% and 16%, and therefore represent a small, yet not negligible, phenomenon. The level of ownership of those developers is not particularly low, with a median value per project between 10% and 42%. This indicates the possible use of SATD as a different way to perform code review, although this behavior should be considered sub-optimal to the use of more traditional tools, which entail suitable notification mechanisms.**

***Index Terms*—Self-admitted technical debt; Code review; Empirical study**

## I. Introduction

Technical Debt, "not quite right code which we postpone making it right" [5], has been recognized as a major concern both by practitioners and by researchers [1], [8], [11], [13]. Potdar and Shihab have observed that source code comments often contain indications of technical debt such as "FIXME: This is such a gross hack..." and "Ugly, but what else?" [12]. To characterize these comments they have introduced the notion of *Self-Admitted Technical Debt* (SATD).

Although the definition by Potdar and Shihab [12] did not presuppose that TD must necessarily be admitted by the same person who has written the source code, the "self" notion gives the impression that SATD is mostly a self-annotation, having the purpose of being a reminder for themselves and for the others. Indeed, several studies of SATD introduction and removal [12], [3] have focused on modification of the comments rather than the corresponding source code. However, it can still happen that a developer notices a technical debt in the source code written, or recently modified, by somebody else, and decides to leave a comment. The importance of identifying which developer self-admitted TD has been recognized by Sierra et al. [15] and Siegmund [14]. Indeed, keeping track of whoever introduces (and admits) TD could be useful not only for TD management, but also for a retrospective analysis because of the potential unavailability of such developers.

Why can it happen that developers admit TD in somebody else's source code? We conjecture that the introduction or change of a SATD comment without modifying the source code may have multiple purposes. On the one hand, as for the SATD admission explained above, it is a reminder for the general development community. On the other hand, it can also be considered a sort of "code review" performed in a rather unusual way, by commenting on somebody else's source code, instead of relying on code review platforms such as Gerrit.

This paper represents a first, preliminary investigation of the extent to which SATD comments are introduced to comment source code that one has not written, or at least modified lastly. To achieve this goal, we start from known SATD instances in five Java open-source projects of a curated dataset by Maldonado *et al.* [6]. Then, we identify the SATD comment lines in the source code, and we trace them back to their introduction and/or last changes using GIT BLAME. Then, based on the comment location, we attach it to source code elements (methods, blocks, or single statements) and check whether any source code line has also been modified together with the comment. If this is the case, we highlight it as an instance of TD admitted by "somebody else".

It can still happen that one may comment source code not recently modified by them while still having a good knowledge of the source code fragment they are commenting. To this aim, we analyze the level of "ownership" of the comment's authors for the source code fragment. We rely on a definition of ownership by Bird *et al.* [4] based on the proportion of source code changes made by an author.

Results indicate that, in the studied projects, between 0% and 16% of SATD is introduced or changed without modifying the source code. At the same time, developers who authored such commits have, in general, a relatively high ownership, often well above the 5% threshold suggested by Bird *et al.* [4]. Finally, we report multiple examples when developers comment somebody else's code with SATD, and show how this happens for different purposes, *e.g.,* improving the overall system documentation and understandability, pointing out the possible impact of a change somewhere else in the code, or that some source code (*e.g.,* a method) is no longer used after a change elsewhere. In summary, these represent cases in which SATD comments are used as a code review mechanisms, although this trigger further research in this area, in order to ensure appropriate SATD awareness to all the interested developers.

The study dataset is available for replication purposes [10].

## II. Related Work

This section briefly discusses previous literature that has studied the SATD phenomenon with the main goal of understanding it. A broader analysis of SATD literature—covering approaches for SATD detection, comprehension, and repayment—is available in a systematic literature review by Sierra *et al.* [15].

Potdar and Shihab [12] introduce the notion of SATD and conduct the first empirical study aimed at understanding the occurrence of SATD in software projects. The study highlights that SATD is very common, *i.e.,* up to 31% of the studied systems contain at least one SATD.

Multiple authors also investigated what type of TD is usually self-admitted in source code. Alves *et al.* [1] identify 13 types of TD belonging to different software artifacts. Maldonado and Shibab [7] manually classify more than 33 thousand comments and observe that only 5 types of SATD can be found in source code, *i.e.,* design, defect, documentation, requirement, and test. Their results also highlight that the majority of SATD found in the studied projects belong to design debt.

Bavota and Russo [3] empirically evaluate the diffusion of SATD in OSS and its evolution. Their results point out that even if 57% of SATD is actually removed from the code, the removal usually happens far from their introduction, *i.e.,* on average after more than 1000 commits.

While looking at the SATD removal, Maldonado *et al.* [6] survey developers and find that developers mostly remove SATD during bug fixing or while adding new features. Finally, Zampetti *et al.* [17], conduct an in-depth investigation on the removal SATD showing that the majority of SATD removal occurs by chances, even if 33% to 63% of SATD is removed while also changing the affected source code, *e.g.,* by modifying conditional statements or by changing method (API) calls.

To the best of our knowledge, none of the aforementioned papers study the relationship between SATD authorship and source code authorship. At the same time, the attention to who manages SATD [3] and how such an SATD is managed and removed [6], [17] makes our investigation important, not only to better understand SATD practices and possibly relate it to modern code review practices [2].

While not about SATD, very related is the work by Fluri *et al.* [9], who study the co-evolution of source code and comments in software projects. We share the methodology to relate comments (and their changes) to source code elements.

## III. Empirical Study Design

The *goal* of this study is to analyze the authorship of SATD-related comments, to determine whether they have been introduced by a developer who also changed the related source code or, possibly by somebody else that has noticed a likely occurrence of TD in the source code. The *context* consists of data from five open-source projects belonging to an existing, curated SATD dataset by Maldonado *et al.* [6]. We selected five projects for which we could access their history on GitHub.

We aim at addressing the following research questions:

TABLE I
Dataset overview

| Project | SATD Instances (dataset [6]) | Analyzed SATD Instances | # of Authors |
|---|---|---|---|
| Ant | 124 | 116 | 101 |
| ArgoUML | 1145 | 1005 | 54 |
| jFreeChart | 109 | 206 | 26 |
| jMeter | 316 | 278 | 65 |
| jRuby | 462 | 220 | 479 |

- **RQ$_1$:** *Is Technical Debt admitted by developers who changed the affected source code?* We aim at distinguishing between developers that add an SATD comment as part of code modification and those that add an SATD comment to the source code that has been modified by their peers. The latter case might indicate that, rather than "admitting" potential problems in her own source code, a developer is signaling problems introduced in the source code written by somebody else.
- **RQ$_2$:** *What is the proportion of ownership while adding a TD comment into the code?* While a developer could have added an SATD comment adjacent to a code fragment where somebody else did the latest changes, it is still possible that the developer has modified the code fragment in the past. To answer RQ$_2$ we analyze the level of code ownership [4] a developer has on the code fragment where they added a SATD.

### A. Dataset and Data Extraction Methodology

Table I shows, for each project, (i) the number of SATD from the Maldonado *et al.* dataset [6], (ii) the number of SATD comment instances we actually analyze (we explain in the methodology why some instances could not be analyzed), and (iii) the number of unique commit authors for the project.

We start from the dataset described above and available from Maldonado *et al.* [6] replication package. In such a dataset, for each SATD comment, we could access (i) the project name, (ii) the reference commit in which the SATD was detected (and reported in the dataset).

Given an SATD comment and a commit in which it was detected, we use a string-matching approach to identify the source code file(s) and the location(s) where the comment occurs (the same comment may occur in multiple files, and also in multiple locations of the same file).

As shown in Table I, the number of instances in the Maldonado *et al.* differ from those we analyze because we could not match all comments from the dataset onto source code snapshot. On the contrary it can happen, as in the case of jFreeChart, that the same comment from the dataset is matched multiple times, therefore the number of analyzed instances can be greater than the number of those in the dataset.

Then, using GIT BLAME, we identify by whom the SATD comment was changed. We use the `-w` GIT BLAME option to ignore formatting changes. This identifies both changes and additions to SATD comments. For the purpose of our work, we are interested in both cases.

Once we have identified the commit in which the SATD comment has been introduced and/or changed, we check it out and analyze the source code file using SRCML. By relying on the Abstract Syntax Tree (AST) extracted by SRCML we relate comments onto source code elements. To this aim, we use the following heuristics, inspired by previous work on source code and comments co-evolution by Fluri *et al.* [9]:

1) A comment preceding a method signature refers to the whole method.
2) A comment preceding the closing bracket of a method refers to the whole method.
3) A comment preceding a class definition refers to the whole class.
4) A comment preceding the closing bracket of a class definition refers to the whole class.
5) A comment inside a method, if it is not on the same line of a statement terminating by a ";", refers to the source code line immediately following the comment.

The aforementioned analysis allows us to identify the set of source code lines to which a comment refers. By using GIT BLAME on such source code lines, we identify the commit of their last change, and, consequently, their authors.

This leads to three different scenarios, depending on whether the source code line has been modified (i) in the same commit of the SATD comment, (ii) by the same author, but in a previous commit, and (iii) in a previous commit by somebody else. Through a manual analysis of the commit authors' list, we made sure our analysis is not affected by imprecisions due to the presence of multiple aliases for the same author or commits authored by multiple people. No such a case was found. Based on the extracted information, we can address $RQ_1$, by determining how many SATD comments have been added (or modified) by somebody that did not author the last change to any of the lines attached to the SATD comment.

As a final analysis for $RQ_1$, two authors have manually scrutinized all the candidate cases (splitting the analysis between them, after having discussed together the protocol on a small subset of cases) to validate them and to determine the extent to which the SATD comment was added or simply modified, and to identify interesting cases to discuss.

While a developer could have introduced a comment related to lines recently changed by somebody else, it is important to also know the extent to which they "own" such source code, and therefore likely have enough knowledge for discussing it. To determine whether this is the case, given the author of an SATD comment, we compute the ownership [4], *i.e.,* their proportion of past changes to the same file, over the total number of past changes occurred on the file. We consider a coarse-grained definition of ownership (*i.e.,* at file-level) because we assume that if one changes a Java class frequently, they should be knowledgeable enough of its methods.

To address $RQ_2$, we report (i) the ownership distribution for SATD comment authors in form of box-and-whisker plots, and (ii) the number and percentage of cases in which the SATD comment has been introduced by somebody that performed at least $x\%$ of the past changes on that source code file, varying $x \in [5\%, 50\%]$ step by 5.

TABLE II
SATD ADMISSION WITHOUT MODIFYING THE SOURCE CODE.

| Project | Analyzed SATD | Admitted by somebody else | Added | Changed |
|---|---|---|---|---|
| Ant | 116 | 12 (10.34%) | 8 (67%) | 4 (33%) |
| ArgoUML | 1005 | 162 (16.12%) | 130 (80%) | 32 (20%) |
| jFreeChart | 206 | 0 (0%) | 0 (0%) | 0 (0%) |
| jMeter | 278 | 32 (11.51%) | 28 (88%) | 4 (12%) |
| jRuby | 220 | 7 (3.18%) | 7 (100%) | 0 (0%) |

## IV. STUDY RESULTS

This section reports the study results, by addressing the research questions formulated in Section III.

### A. Is Technical Debt admitted by developers who changed the affected source code?

Table II reports the number and percentage of SATD (out of those we analyzed) where the admission has been done by somebody different from the author of the last source code change related to the comment itself. Out of the 229 cases we initially found, the manual analysis removed 16 false positives due to an improper mapping of comments onto source code, leaving with the 213 cases reported in the table.

The percentage of admissions made by "somebody else" varies from 0% of jFreeChart to 16.12% of ArgoUML. Therefore, this is a relatively small, yet non-negligible proportion of cases among those we analyzed. Such a result generally confirms the "common wisdom" according to which developers self-admit TD when they introduce not ready yet source code. At the same time, there is still a percentage of cases (over 10% for three projects out of the five we analyzed) where we observed SATD comments added by somebody else.

TD admission by somebody else never happens for jFreeChart. As shown in Table I, this project has a relatively limited number of authors (26, compared to 54–479 for other projects). It could be possible that, with a small development community, an inter-developer communication through SATD makes less sense, whereas self-admissions are still useful.

As the last two columns show, in most cases the admission by somebody else manifests with the addition of a new SATD comment: this happens in a percentage ranging between 69% (Ant) and 100% (jRuby), with ArgoUML and jMeter being at 80% and 88% respectively.

**$RQ_1$ summary:** There is a non-negligible percentage [0%, 16%] of SATD comments being admitted by somebody different from the developers authoring the related source code lines. Most of them are newly-added SATD comments, while a minority [0%, 31%] concerns comment changes.

### B. What is the proportion of ownership while adding a TD comment into the code?

While, as we have observed in $RQ_1$, developers may admit TD on source code changed by somebody else, these developers

may still be very familiar with the source code they comment. Fig. 1 reports, as boxplots, the distribution of ownership authors of SATD have on the source code they commented when they were not the author of the last change. Note that we do not report and discuss results for jFreeChart because in $RQ_1$ we found no cases of SATD on somebody else's code.

As the boxplots show, the ownership is generally high, *i.e.,* the median value varies from 11% for Ant to 43% for jMeter. The ownership level for jMeter looks particularly high. By looking at its commit log, we found that each source code file is changed, across its evolution, by 1-3 developers (median 1). This looks low, however also ArgoUML exhibits similar values, whereas in Ant and jRuby source code files are modified by up to 6 and 9 authors respectively. Therefore, this is only a partial explanation of the observation, and further investigations on different ways developers collaborate would be required.

Bird *et al.* [4] define a *major contributor* as a developer that has done 5% of the total number of changes involving the software component. In the majority of the cases the developer admitting SATD is a major contributor for the software component showing TD. This result can be confirmed by looking at the percentage of SATD on code changed by somebody else made by authors having different levels of ownership (Fig. 2). While increasing the ownership threshold, we observe a decrease in the percentage of SATD comments introduced by somebody else. However, for jRuby and Ant 37.5% and 23% of SATD are admitted by minor contributors. We conjecture that this may happen as a consequence of the code review process done without relying on ad-hoc tools.

**$RQ_2$ summary:** In most cases, whoever adds or modifies SATD comments on source code lastly changed by somebody else is a major contributor of that source code file. There are cases (*e.g.,* up to 37.5% for jRuby) where the admission comes from a minor contributor.

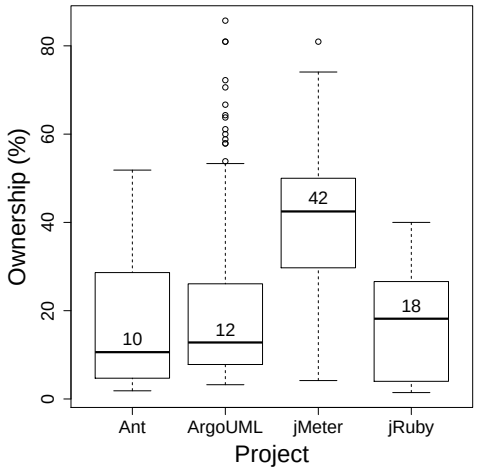


Fig. 1. Ownership of authors who admitted TD on code changed by somebody else, (jFreeChart having no such SATD comments is excluded).

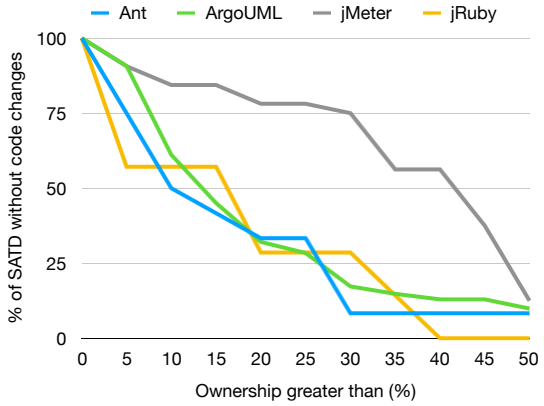


Fig. 2. Percentage of SATD on code changed by somebody else made by authors with ownership greater than X%

### C. Admission for TD Introduced by Somebody Else

In some cases, the SATD has been introduced **during changes related to revisions** to point out possible impact of such changes elsewhere. As an example, in jMeter we found a comment *"// TODO: This method doesn't appear to be used"* being introduced in a commit clearly reporting refactoring action aimed at not changing the behaviour: *"Reformatted to conform with JMeter (Turbine) conventions"*. Looking at the comment body, it is clear that the reviewer is pointing out the presence of a method never used so probably not needed.

There are also cases in which the admission by somebody else is due to the attempt of **fixing a bug**. In this case, the developer looks at the code to fix the bug and incurs in a TD that need to be addressed, or introduces the TD as a consequence of the bug fixing activity. For instance, in jMeter we found a SATD comment *"TODO consider removing this method, and providing method wrappers instead [...]"* being introduced in a commit aimed at fixing a bug "Bug 44022 - Memory Leak when closing test plan". The same happens in ArgoUML, where while trying to address a bug in a change reporting "Issue 5438: Make use of new listener minimal update facility so that we don't unregister all our listeners and [...]" the developer also introduces a SATD comment in a related source code component: *"// TODO: This brute force approach of updating listeners on each // and every event, without checking the event type or any other // information is going to cause lots of InvalidElementExceptions [...]"*. Finally, there are some interesting cases in jRuby where, while dealing with a bug "JRUBY-4071: SystemCallError.new does not create an Errno instance", a developer identifies the presence of a TD inside the code being affected by the bug and reports its presence adding the following SATD comment: *"// FIXME: these descriptions should probably be moved out, // to Constantine project which deals with all platform-dependent constants."*

In ArgoUML, we also found cases where the SATD introduction is due to activities aimed at **improving the overall documentation or code readability**, *e.g.,* the SATD comment: *"// TODO: This is probably an undesirable side effect unless the user // confirms it."* in a change having as message "Minor documentation cleanups". The same happens

also in Ant where a SATD comment: *"// XXX: (John Doe[1]) The comment "if it hasn't been done already" may // not be strictly true. wrapper.maybeConfigure() won't configure the same // attributes/text more than once, but it may well add the children again, // unless I've missed something."* is introduced while adding "JavaDoc comments". The author directly points out the SATD being introduced while reporting the commit message, *i.e., "Note: maybeConfigure implies that calling it twice will have no effect. I have a suspicion that children would be added twice. Search for XXX to find the details."*

Finally, we report one interesting example of comment change from ArgoUML (there are other interesting ones, omitted due to space limits). One SATD comment for a public method says *" We should not use assert on public methods."* This was changed into something that explains the TD better, and actually tells that it should probably not fixed for now: *"should not be using assert here but I don't want to change to IllegalStateException at lead up to a release as I don't know how much testing is done with assert on."*

## V. Threats to Validity

Threats to *construct validity* are mainly related to the measurements we perform. First of all, to address $RQ_1$ and to investigate if the SATD has been modified or added by somebody else we rely on GIT BLAME, which may be subject to imprecisions in tracing changes. Second, we used a set of heuristics to relate comments to source code. Again, it is possible that such heuristics could lead to imprecisions. Also, through a manual analysis, we (i) checked for imprecisions due to multiple author aliases or commits with multiple authors, and (ii) analyzed the results of $RQ_1$, to validate them and distinguish between changes and additions.

Concerning the ownership, as explained in Section III it is measured at file-level, even though an SATD may refer to a method or a statement. However, we assume that a developer frequently changing a file is at least knowledgeable of its content: this is not necessarily true in corner cases.

Threats to *internal validity* concern internal factors to our study that could influence our findings. While our study does not really claim any cause-effect relationship between variables, the link between "commenting somebody else's source code" and performing code review must be interpreted very carefully, because we may not be fully aware of the developers' intent.

Threats to *external validity* concern the generalizability of our findings. This is intendedly a small study on five curated datasets. Thus, we plan to investigate whether the results hold on a larger and more diverse dataset.

## VI. Conclusion and Future Work

We study the extent to which developers introduce or modify SATD comments for source code on which somebody else has performed the last changes. Results from five open-source projects [6] indicate that such a phenomenon concerns up to 16% of the SATD instances. The change is generally made by developers having a high level of ownership on that code.

[1]The name of the developer has been anonymized to protect their privacy.

The findings of this paper indicate that, in general, SATD reflects a "self" admission made when a developer has written (or modified) a piece of source code making it "not ready yet". At the same time, there is a non-negligible number of cases in which SATD should be seen as a code review of source code authored by somebody else. While this does not threaten the general SATD metaphor, it calls for refining its interpretation.

Also, this warns developers for using more appropriate code review mechanisms than source code comments. On the one hand, SATD comments are somewhat more visible than code reviews (*i.e.,* SATD comments are just there, in the source code, whereas code reviews comments could be easily forgotten after a review is closed). On the other hand, it would be desirable to develop better notification mechanisms once TD is admitted, combining the advantages of SATD and code review tools.

Last, but not least, the presence of SATD comments out-of-sync with source code changes calls for approaches aimed at recommending when TD should be admitted [16] and, possibly, to help developers in properly documenting TD.

As future work, we plan to extend this study to a large set of open-source projects.

## References


[1] N. S. R. Alves, L. F. Ribeiro, V. Caires, T. S. Mendes, and R. O. Spínola, "Towards an ontology of terms on technical debt," in *MTD*, 2014, pp. 1–7.

[2] A. Bacchelli and C. Bird, "Expectations, outcomes, and challenges of modern code review," in *ICSE*, 2013, pp. 712–721.

[3] G. Bavota and B. Russo, "A large-scale empirical study on self-admitted technical debt," in *MSR*, 2016, pp. 315–326.

[4] C. Bird, N. Nagappan, B. Murphy, H. C. Gall, and P. T. Devanbu, "Don't touch my code!: examining the effects of ownership on software quality," in *ESEC/FSE*, 2011, pp. 4–14.

[5] W. Cunningham, "The wycash portfolio management system," *OOPS Messenger*, vol. 4, no. 2, pp. 29–30, 1993.

[6] E. da S. Maldonado, R. Abdalkareem, E. Shihab, and A. Serebrenik, "An empirical study on the removal of self-admitted technical debt," in *ICSME*, 2017, pp. 238–248.

[7] E. da S. Maldonado and E. Shihab, "Detecting and quantifying different types of self-admitted technical debt," in *MTD*. IEEE, 2015.

[8] N. A. Ernst, S. Bellomo, I. Ozkaya, R. L. Nord, and I. Gorton, "Measure it? manage it? ignore it? software practitioners and technical debt," in *Foundations of Software Engineering*. ACM, 2015, pp. 50–60.

[9] B. Fluri, M. Würsch, E. Giger, and H. C. Gall, "Analyzing the co-evolution of comments and source code," *Softw. Qual. J.*, vol. 17, no. 4, pp. 367–394, 2009.

[10] G. Fucci, F. Zampetti, A. Serebrenik, and M. Di Penta, "Who (self) admits technical debt?" Aug 2020. [Online]. Available: https://doi.org/10.5281/zenodo.3984829

[11] E. Lim, N. Taksande, and C. Seaman, "A balancing act: what software practitioners have to say about technical debt," *IEEE software*, 2012.

[12] A. Potdar and E. Shihab, "An exploratory study on self-admitted technical debt," in *ICSME*, 2014, pp. 91–100.

[13] C. Seaman and Y. Guo, "Measuring and monitoring technical debt," *Advances in Computers*, 2011.

[14] J. Siegmund, "Program comprehension: Past, present, and future," in *SANER*, vol. 5, 2016, pp. 13–20.

[15] G. Sierra, E. Shihab, and Y. Kamei, "A survey of self-admitted technical debt," *Journal of Systems and Software*, vol. 152, pp. 70–82, 2019.

[16] F. Zampetti, C. Noiseux, G. Antoniol, F. Khomh, and M. Di Penta, "Recommending when design technical debt should be self-admitted," in *ICSME*, 2017, pp. 216–226.

[17] F. Zampetti, A. Serebrenik, and M. Di Penta, "Was self-admitted technical debt removal a real removal?: an in-depth perspective," in *MSR*, 2018, pp. 526–536.