As we split our research into two different approaches, I have highlighted the portions and analyses that I wrote for the sake of clarity.

On the Relationship Between Technical Debt, Refactorings, and Bug Fixes

Gabby Albrecht & Meredith Diamond

Introduction: Motivation and Problem

For many programmers working with version control systems, committing changes often involves refactoring or bug fixes. On the surface level, these two kinds of code improvements do not appear to have any correlation. Bug fixing is self-explanatory, referring to the alteration of a program in order to handle an existing glitch or bug. Refactoring relates to any form of restructuring code so as to improve aspects such as readability and implementation while preserving the original functionality. In an ideal world, bug fixing and refactoring should have no correlation to one another, as each one handles a different kind of programming problem. However, the idea of whether the two share a relationship has been considered often in the world of software engineering and research. Even though refactoring seeks to maintain functionality, it is reasonable to believe that it could cause bugs as well, simply due to the fact that it deals with the physical augmentation of code.

Technical debt is another common phenomenon that comes with software development. It is defined as the implied additional work that must be done afterward when developers opt for the easier, quicker solution, rather than the optimized one. In other words, technical debt is the result of, at times, subpar code, and the effort required to clean it up. Refactoring and technical debt are already known to have a relationship with each other (1), and refactoring is often the method of resolving debt because it involves improving implementation. This then brings up the question, if refactoring and bug fixing is already speculated to have a correlation, then do technical debt and bug fixing have some sort of relationship as well? Considering this, a problem arises too: many developers have concerns about refactoring code for the fear that doing so will bring about bugs (2). However, refactoring is also a solution when trying to improve technical debt. With these conflicting results, how can developers be sure whether their refactoring is doing more harm than good? Or is the correlation, if any, between refactorings and bugs small enough to be dismissed?

In this paper, we aim to explore the relationship between technical debt, refactorings, and bug fixes through multiple methods of analysis, with the hope that doing so will provide helpful insight into the way each could possibly affect the other. The data we will be utilizing comes from the 2022 MSR Mining Challenge, hosted by the Mining Software Repositories conference. In short, the challenge is an annual loosely-guided research project, in which a dataset is provided along with a set of potential research questions. This particular year's dataset comes from SmartSHARK data, a collection of in-depth information ranging from various version control systems. The smaller set, which we will be using, covers over 40 gigabytes of commit messages, code clones, refactorings, pull requests, and several other related fields. For the purposes of this project, however, we are only concerned with commit messages. In the following sections, we will be walking through our methods of analysis, evaluations of the resulting data, and a discussion of both our and other researchers' conclusions.

Approach

1st Approach

Our first thought when approaching these questions was seeing how often commits about refactoring and bugs occur together. We began by parsing through all the commits in the SmartSHARK data to find the commits with messages. From there, we were able to begin parsing for specific words in order to categorize each commit by whether it contained references to bug fixes, refactoring, or neither.

Originally, we parsed only for “refactor” and “bug”. This returned 15,004 commits with messages containing the word “bug” and 6,994 commits with messages containing the word “refactor”. Only 288 of those commits contained both. We chose not to parse for the word “debug” because those commits were already being identified by the keyword “bug”.

Next we questioned whether the commits following those we had already parsed, contained the other word. Would a commit containing “refactor” be commonly followed by a commit containing “bug”? Unfortunately, the SmartSHARK data does not have a way to identify if commits are part of the same project. We could check for the same author ID or committer ID, but that would discount all commits which were made by other team members. If SmartSHARK organized commits by project, we would be able to simply look at the following commit in the list. Instead, we separated all of the data into lists of commits sharing the same committer ID.

Because of this, our data does not represent the instances where one person pushes their refactoring code, and someone else attempts to push their code, but now there are bugs caused by some miscommunication in the team. We can only look at preexisting bugs and when they were fixed. We also cannot tell if while one person works on refactoring, another is working on debugging. If we were to gather our own data to study, we would include a project ID so we could include more variables.

From the commits of the same committer ID, we checked each commit and the one that follows immediately after for our keywords. In the future, we would separate each committer’s messages into lists organized by the date of the commit. With our limited storage and time during this project, we unfortunately could not do this. However, if we were to continue this research, we would take these steps to make our data even more accurate.

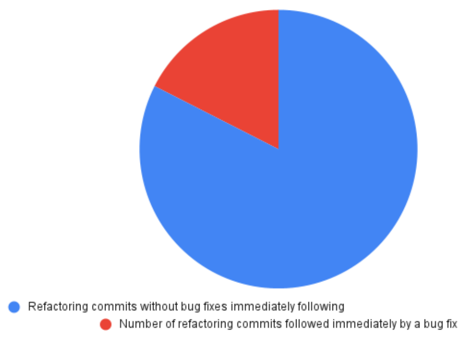
Of course, we also run into problems with language and vocabulary. Refactoring has many synonyms, and we cannot rely on committers who only use the words “refactor” and “bug”. Therefore, we added terms to our parser which refer to the same acts. To our “refactor” list, we added “rearrange”, “restructure”, “streamline”, “redesign”, and “alter”. We also added configurations of the words where they were all caps or the first letter was capitalized. With the added words, our parser was able to decrease the loss of data that we faced originally.

To the “bug” list, we added “fix”. Committers often refer to the exact problem or bug they fix without saying the word “bug.” Adding “fix” to the list more than quintupled the number of commits selected. With now 80,221 bug fixing commits detected, the sampled data encapsulates several more practices. Our parser now detects 8,637 refactoring commits.

There are several more margins for error using this parsing method. What happens if a commit message references a bug without fixing it? What if a message addresses something caused by refactoring? With such large quantities of commits, we cannot go through each individually to ensure we have classified each message correctly. Our next steps would be to refine our parser to do that work for us.

We now have lists of the data of our interest. From here, we collected the sum of projects, the sum of refactoring commits, and the sum of bug fix commits that follow those refactoring commits. We also collected the sum of refactorings and bug fixes which follow and vice versa for each project. Our data looks at, in total, 4,128 unique committers. With this information at our disposal, we can begin to analyze.

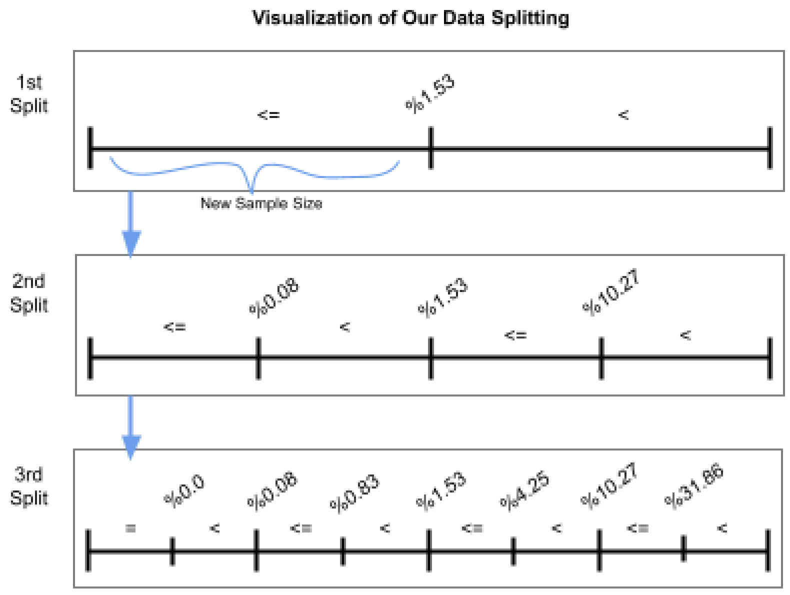
We also analyzed commit messages which contained references to both refactoring and bug fixes. With a total of 1,245 commit messages containing both, we were able to discern the frequency in each project that refactoring and bug fixes occurred concurrently.



2nd Approach

Another method of analysis we used focused on seeing if there was an absolute positive trend with refactorings and bug fixes. With this method, we wanted to split projects into different groups based on the frequency of refactoring-related commits. As the occurrence of refactorings went up, would the amount of bug fix-related commits as well? As mentioned before, since the SmartSHARK dataset did not provide a way to connect commits to the same project, we had to opt for linking commits by shared committer ID; in other words, commits made by the same person. While this means the results of our approach are less definitively accurate, in a way we still get to see trends across different projects. For the most part, committers will be unique to their project, and often projects have one or a few people who comprise the majority of commits for that project.

To begin, we parsed through every commit, storing them in a Python dictionary by shared committer ID. In addition, for each entry in the dictionary, we calculated the average percent of commits that contained “refactor” and “bug fix.” This gave us the commits from 4128 different people to analyze. Then, using the average percent of refactorings stored in each dictionary entry, we found that a total average of 1.53% of **all** commits made per committer ID had references



to refactoring.

With this average, we started by splitting the data into two groups: Those with less than or equal to 1.53% of commits with refactorings, and those with more. Then, in each of those two groups, we once again found the average of commits per committer ID that mentions refactoring. We split on these averages as well, and now had a total of four groups. Finally, we performed a third split using the same methods as before, and were left with eight groups of data. For each process, such as finding a value to split on or dividing up groups, we developed functions that would complete such calculations for us. Since the commits were arranged in a dictionary, and we included extra information about how many commits were refactoring- or bug-related, it became rather easy to access and compute the data we needed.

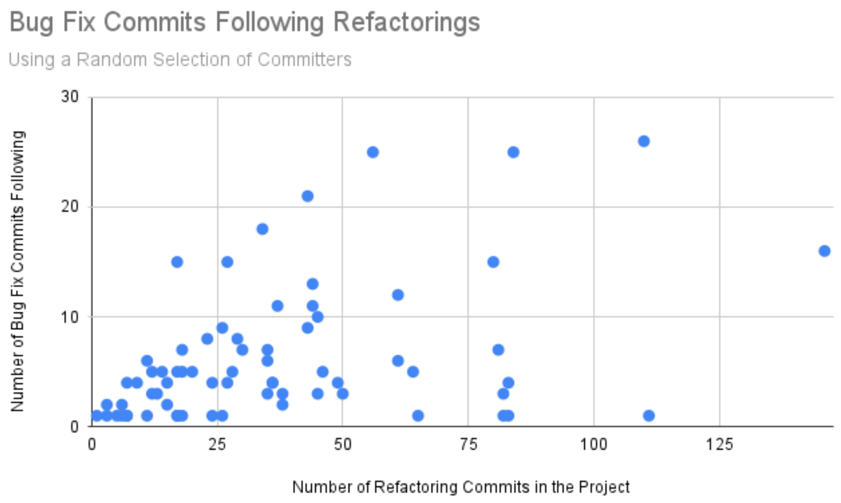
Of course, several improvements could be made to our parser. Ideally, with a more robust algorithm, we would continue splitting the data even further, such that each subgroup would have a sample size of 0 to 3 groups of commits to observe. The idea is that with each split, we get to see a trend of whether or not the amount of bug fixes increases or decreases, or has no distinct trend at all, as the concentration of refactorings goes up. More splits means a better defined trend, and in the future we would take care to implement an algorithm that could do such.

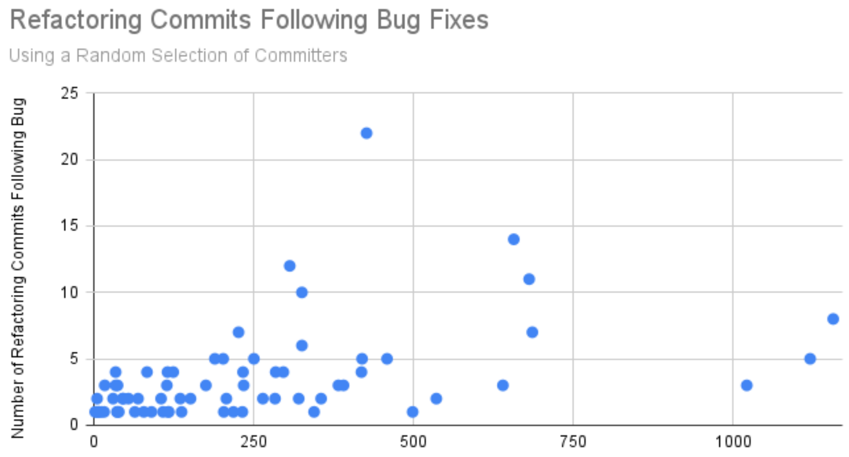
For now, we had eight subgroups to observe, and for each we computed the average percentage of commits per committer ID that contained the phrases ‘bug’ and ‘fix.’ We also took note of the sample size of each group, as this could help provide some insight about the frequency of refactorings and bug fixes appearing together. Some sets had over 3000 committer IDs, while others had less than 100.
In our evaluation section, we will take care to discuss the ramifications of these varied sample sizes and an in-depth analysis of the data.

Evaluation

Results from 1st Approach

According to our first approach, refactoring and bug fixes often happen together, but neither necessarily causes the other. Refactoring happens directly after 1.5% of bug fix commits. However, bug fixes happen immediately after 17% of refactoring commits. This is not unsurprising considering the nature of each task. As one goes back to clean up code, it is likely they might try to then fix parts near where they streamlined. However,



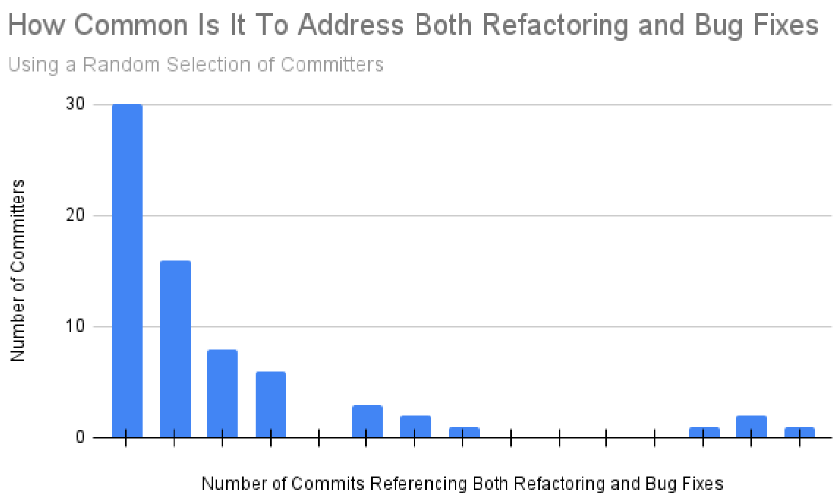


bug fixes do not lend themselves to easy refactorings.

456 unique committers refactored in one commit and committed bug fixes immediately after. That means over 10% of committers prefer to do their refactoring first, and then address the bugs in their code. 426 unique committers committed in the opposite order. So while refactoring happens less often after bug fixes than vice versa, about the same number of people use each method.

These committers perform these tasks at different rates. While it seems as the number of refactoring commits increases, so does the number of bug fix commits which follow. However, no matter the number of bug fixes, it seems that refactoring happens immediately most often after only 1-10 bug fix commits.

1,245 commits contain references to both refactoring and bug fixes. That is only about 1.4% of commits containing references to either refactoring or bug fixes. However, if you look at the percentage of refactoring commits, that is 14.4% of the commits. Only 386 unique committers of the 4128 performed refactoring and bug fixes in the same commit. It is much more common in these projects to commit only once or twice after refactoring and fixing bugs at the same time.



There is a clear correlation between the two acts. However, as we know, correlation does not equal causation. There is no data present which implies that one might fix a bug because of the refactoring one just applied. There is also no data which implies that refactoring happens because of bug fixes. It definitely helps the process, but it does not cause anything. The order and pattern in which one refactors and fixes bugs is up to the individual. While there is a similar number of times each pattern we looked at occurred in the data, looking at the data as a whole, it seems that refactoring and then fixing bugs is more common. This makes sense considering cleaner code lends itself to solving bugs a little bit easier.

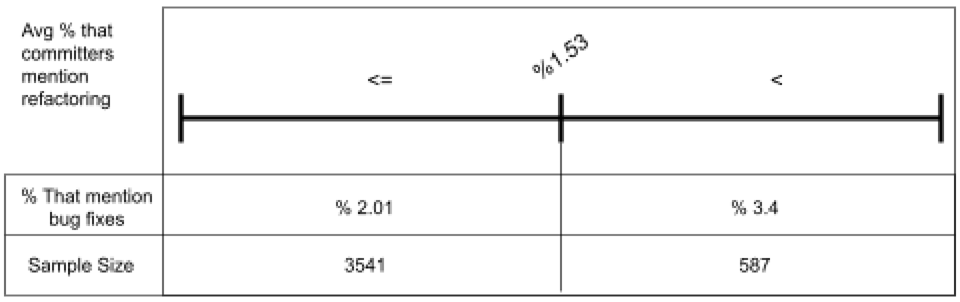
Results from 2nd Approach

Our second approach of data collection provided a clearer positive trend between occurrence of refactorings and those of bug fixes. Splitting the data into smaller subsets allowed us to better see this relationship. Again, our first split of the data began by sorting different committers and their commits into two groups: those that mentioned “refactor” less than or equal to %1.53 of the time, and those who mentioned the phrase more.

In the first subgroup, under this split, there was a sample size of 3541 different committers, while over the split, there were 587 committers. Even in this first split, we can see that the data is favoring the side of less mentions of refactoring. This makes sense, because often, the majority of commits for a project will not necessarily be about refactoring, and rather there is usually a variety. Thus, on average most projects, or at least most committers, will not be spending a majority of their time on refactoring.

Now in each group, we looked at average percent of times committers mentioned "bug" and "fix" within a single commit. We found that in the lower split, 2.01% of commits were related to bug fixes, and in the higher split, 3.4% were related. The seemingly obvious trend is that as the appearance of refactorings increases, so does the appearance of bug fixes: from the first subgroup to the other, bug fixes increase by almost 60%. It is interesting that this is the case as well, because the group over the split has a smaller sample size, and yet is far more dense in refactorings and bug fixes than the 3541 committers on the lower side of the split.

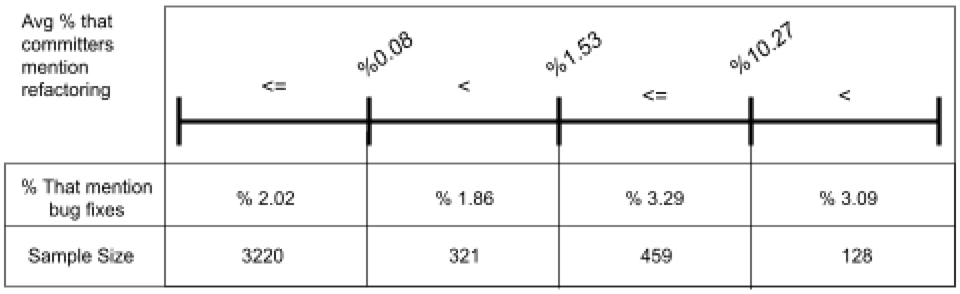
1st Split



Avg % that committers mention refactoring	<= %1.53	<
% That mention bug fixes	% 2.01	% 3.4
Sample Size	3541	587

Our second split divided the full dataset into four subsections. You can see in the figure below the different sample sizes and percentages for each group. Once again, the majority of committers do not reference both refactorings and bug fixes very often. 3220 committers mentioned refactoring less than or equal to to 0.08% of the time on average, and within that groups only 2.02% mentioned bug fixes. On the higher side of the data, committers that referenced refactorings in 10.27% of their commits also referenced bug fixes 3.09% of the time. Between the lower and higher end, there was about a 67% increase in bug fixes.

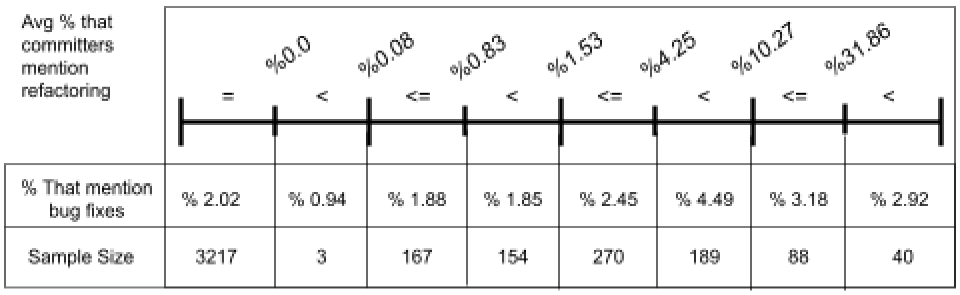
2nd Split



Avg % that committers mention refactoring	<= %0.08	< %1.53	<= %10.27	<
% That mention bug fixes	% 2.02	% 1.86	% 3.29	% 3.09
Sample Size	3220	321	459	128

In our final split, we were left with eight subgroups. The figure below shows our findings for this segment of analysis. The table shows, somewhat surprisingly, that there are 3217 committers that never produced a refactoring commit. This could be due to several circumstances, however. For example, there may be some people that only committed a handful of times, and it makes sense that their commits would lack variety. It could also mean that some of these people were not assigned the role to

3rd Split

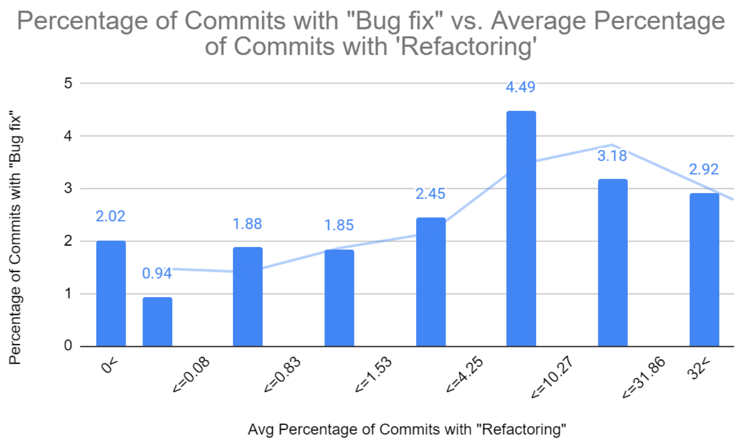


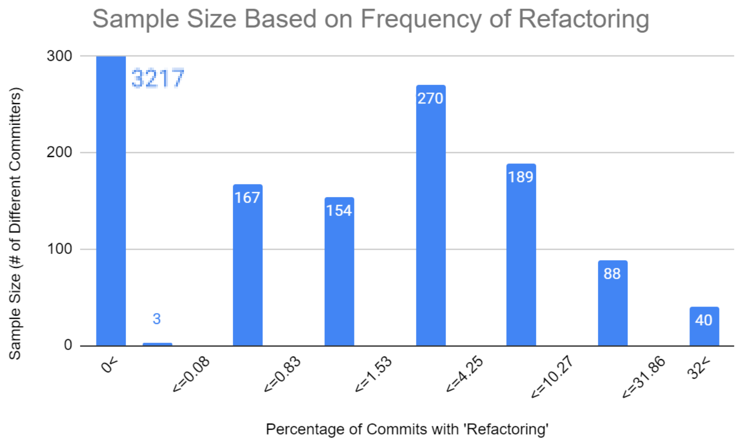
Avg % that committers mention refactoring	= %0.0	< %0.08	<= %0.83	< %1.53	<= %4.25	< %10.27	<= %31.86	<
% That mention bug fixes	% 2.02	% 0.94	% 1.88	% 1.85	% 2.45	% 4.49	% 3.18	% 2.92
Sample Size	3217	3	167	154	270	189	88	40

refactor code. Another issue is that in this particular approach, we neglected to include synonyms of the word "refactor," and doing so would likely alter these results. Nonetheless, bug fixes are still present in these commits, and it is suffice to say that bug fixes will always exist in the absence of refactorings.

The following two figures to the right provide a better visual representation of the tabled data. You can see that generally, as the presence of refactorings increases, so does the occurrence of bug fixes. It is not a perfect incline however; the highest presence of bug fixes happens between 4.25% to 10.27% of refactorings, and declines slightly from there on. We could potentially interpret this as a sign of committers performing "irresponsible" refactoring. It makes sense that frequent refactoring done on a program could induce bugs, as it involves moving around and re-implementing code, which by human error is bound to result in mistakes.

However, this chart could give insight to the skill of different developers. On one hand, frequent refactoring could suggest that code that was previously written is frequently subpar. On the other hand, it could suggest that the developers are great at being proactive with solving technical debt. Neither conclusion can be made for certain, but nonetheless there is certainly some sort of positive relationship between refactoring and bug fixes.





The second chart shows the wide range of sample sizes resulting from each data split. From as high as 3217 to a miniscule pool of 3, it is clear that the variation in sample size could skew the accuracy of the results. Yet it does give a sense of how little of a problem bug-inducing refactorings may be. As discussed before, the majority of committers seem to not deal much with refactorings. Yet bugs still exist. On the higher end of the spectrum, where refactorings take up 32% or more of all commits, there are only 40 different committers, and bugs were only slightly more prominent. So while refactorings appear to be inducing bugs, it may not happen as often as we think.

Discussion

The results of our research seem intuitive to a veteran programmer. It makes sense that when one cleans up spaghetti code, they might solve a bug or two on the way. However, one might not consider how others' strategy when attacking these tasks differs from one's own.

Many further questions were raised by the answers we discovered in our research. How many committers use multiple patterns for refactoring or fixing bugs? Continuing in our research, we will compare committer ids between each pattern to see if there is any overlap. Do these tasks just happen to occur together or is there intention behind each programmer's choice for when to complete what task?

Going forward, we would aim to gather data that also contains a project ID field, so that we could look at all committers' contributions, and not just one specific person. How does this data change when multiple people are coding at the same time? Are tasks split differently, or are refactoring and bug fixes done simultaneously? It is possible that refactoring may cause problems for other developers who are coding simultaneously. By only looking at individual committers' data, we lose insight into programming as a team. When developing a project, programmers are almost always on a team. We cannot truly understand how refactoring and bug fixes relate to each other if we cannot also view them in the group setting.

We have already mentioned several elements which could threaten the accuracy of our results. If we could organize commits by project and date, we would be able to understand the intricacies of the relationship between refactoring and bug fixes better.

There are several possibilities for error in our parser. The first is the possibility of our parser incorrectly categorizing commit messages. To combat this, if we were to continue this study, we would refine the parser to determine the difference between mentions of bugs or refactorings and the actual acts of refactoring and fixing bugs. Our parser also only looks for certain words and certain capitalizations of those words. If someone were to randomly capitalize letters in their commit messages, our parser would not categorize that commit correctly. Developers also use many different words for the general acts we have been referring to as refactoring and bug fixes. Although we attempted to combat this by adding additional terms which refer to these tasks, there is still a margin for error.

Thankfully, while there are many areas of improvement in our study, several other researchers have made an effort to try and determine the relationship between refactorings and bug fixes with their own insightful conclusions. In 2020, with a powerful and accurate miner, a group of Italian researchers took to a database of 103 repositories and sought to answer the question: "*Are refactoring-related commits more likely to induce fixes than other commits?"* (2). With their parser, they were able to separate commits by those that mentioned refactoring and those that didn't, and determine the likelihood of these commits inducing a bug fix later on. Their conclusions found that, yes, there exists a relationship between the two, and "commits with refactoring always have significantly higher odds to induce a fix than other changes" (2).

Refactorings and bug fixes will always be a part of any software engineering process in some shape or form. It makes sense why developers care so much about how the two affect each other, because they are both such integral parts of code improvement. While most of our results and others' conclusions suggest that the two share a connection, they may not be so overly dependent on each other. Obviously, bug fixes aren't going away, and there will always be technical debt that must be resolved through refactoring. While refactoring can cause bugs, it is arguably more pertinent to refactor and resolve debt, and simply deal with the occasional bug. Overall, our research has given us a greater base understanding of the topic, and an opportunity to dive deeper into all of its intricacies.

References

1. Zabardast, Ehsan, et al. "Refactoring, Bug Fixing, and New Development Effect on Technical Debt: An Industrial Case Study." *IEEE Xplore*, 28 August 2020
2. Di Penta, Massimiliano, et al. "On the Relationship between Refactoring Actions and Bugs: A Differentiated Replication" *arxiv.org*, 24 September 2020
3. Peruma, Anthony, et al. "Refactoring Debt: Myth or Reality? An Exploratory Study on the Relationship Between Technical Debt and Refactoring." *arxiv.org*, 10 March 2022