2022 IEEE International Conference on Software Maintenance and Evolution (ICSME) | 978-1-6654-7956-1/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICSME55016.2022.00048

Evaluating Atoms of Confusion in the Context of Code Reviews

Victoria Bogachenkova*, Linh Nguyen*, Felipe Ebert†, Alexander Serebrenik* Fernando Castor‡,
*Eindhoven University of Technology, The Netherlands, victoria.bogachenkova@gmail.com,
linhnguyenviet21@gmail.com, a.serebrenik@tue.nl
†Jheronimus Academy of Data Science, Tilburg University, The Netherlands, f.ebert@jads.nl
‡Utrecht University, The Netherlands, f.j.castordelimafilho@uu.nl

***Abstract*—Code review is a popular software engineering practice. Success of code reviews can be threatened by confusion experienced by code reviewers. For instance, on the one hand, research has studied the reasons for confusion in code reviews, and on the other hand, it also has analyzed source code patterns, so called "atoms of confusion", that have been shown to lead to misunderstanding in the lab setting. However, to the best of our knowledge, there is no research which tried to investigate the possible cause and effect relationship between atoms of confusion and confusion in code reviews. Another important aspect still not studied is how those atoms of confusion evolve across pull requests. In this emerging results paper, we report an exploratory case study to provide a deeper understanding of atoms of confusion, more specifically, whether atoms of confusion are related to confusion in code reviews and how they persist across pull requests. With the help of an existing tool for the detection of atoms of confusion, and a manual analysis of code reviews comments, we observed that statistical analysis did not show any relationship between atoms of confusion and presence of confusion comments in code reviews. Additionally, we found evidence that atoms of confusion are mostly not being removed in pull requests. Based on the results, we formulate hypotheses on atoms of confusion in the code review context, that should be confirmed or rejected by future studies.**

***Index Terms*—atoms of confusion, confusion, code reviews, pull requests**

I. Introduction

Understanding source code is an essential task of the software development process. Developers can spend more than 50% of their time on program comprehension activities [1], [2]. Code which is difficult or hard to understand can negatively affect not only the development process but also other related tasks, such as code reviews [3]. Recent work has been focusing on one form of accidental complexity, **atoms of confusion**, first discussed by Gopstein *et al.* [4]. These refer to small pieces of code following a pattern that is known to lead to *confusion*, *i.e.*, any situation where the person is uncertain about anything or unable to understand something, among developers and for which there are alternatives that are less confusing.

For example, *post-increment* is one of such atoms of confusion.[1] This atom is present when an increment to one variable occurs in the same statement as an assignment to that variable:

```
V1 = V2++;
```

Semantically equivalent code without confusion would, *e.g.*, split the two operations in two different statements:

```
V1 = V2;
V2 += 1;
```

The main motivation of our study is two-fold: i) we expect that those atoms of confusion are likely to promote confusion among developers also in code reviews (as such task is mainly about program comprehension), and ii) we believe that the trend would be for those atoms to be removed over time during the development process, as they have been shown to be a source for confusion. Even though atoms of confusion have been studied for different programming languages and in different contexts [4]–[8], to the best of our knowledge, there is no literature on atoms of confusion in the code review process. In this work, we conduct an exploratory case study [9] to get further insight into the phenomenon of atoms of confusion by studying whether those atoms are related to confusing code reviews, and how the atoms persist across pull requests (PRs).

Thus, we define our first research question as ***RQ1.*** *What is the relationship between atoms of confusion and confusion in code reviews?*. We analyzed the relationship between the presence of atoms of confusion in pull requests and comments indicating confusion in the corresponding code reviews. The results suggest, contrary to our intuition, there is no relation between atoms of confusion and confusion present in code review discussions. Even though previous research [4] has investigated how project age influences the rate of atoms of confusion and how often bug-fixing commits remove the atoms, there is not study specifically focusing on how atoms evolve across pull requests. Therefore, we define our second research question as ***RQ2.*** *What is the relationship between the presence of atoms of confusion in source code before and after PRs?*. We observe that developers infrequently remove atoms of confusion in pull requests, confirming previous results that projects usually do not dramatically increase or decrease the number of atoms [4]. Finally, as appropriate for an exploratory case study, we formulate hypotheses about atoms of confusion that should be confirmed or rejected by future studies.

[1]The complete list of atoms can be found at [5]. We have not included them due the lack of space.

2576-3148/22/$31.00 ©2022 IEEE
DOI 10.1109/ICSME55016.2022.00048

II. Related Work

Gopstein *et al.* [6] presented 15 atoms of confusion which were obtained from the champion programs of the International Obfuscated C Code Contest.[2] Two controlled experiments were conducted in the study with students and found that those atoms of confusion are harder to understand than the functionally equivalent programs that do not include them. In a follow-up study, Gopstein *et al.* [4] conducted a large-scale repository mining study of C programs and found that the number of atoms in successful projects is large (*e.g.*, in the Linux kernel and Git). Additionally, they found a strong correlation between these atoms and long code comments and bug fixing commits. Langhout and Aniche [5] adapted these atoms for Java programs. By conducting two-phase experiment with students, they could confirm the influence of specific atoms of confusion in Java on the understanding of source code. Medeiros *et al.* [7], by analyzing source code and surveying developers, measured the presence of "misunderstanding patterns" in C projects. They showed that these patterns are frequent in those projects and also that developers consider that some of them can cause confusion. de Oliveira *et al.* [8] used an eye tracker to investigate whether developers misunderstand source code with atoms of confusion. They observed a considerable increase in the time and in the gaze transitions in code snippets with atoms, as well as that the regions that receive most of the eye attention were the regions with those atoms. Finally, Gopstein *et al.* [10] conducted a think-aloud study with students and professional developers to better understand the causes of confusion in code with atoms of confusion. The authors found out that correct hand-evaluations, *i.e.*, predicting the output of a program, do not imply understanding, and incorrect evaluations not misunderstanding.

To the best of our knowledge, no previous work has studied atoms of confusion in the context of code reviews. In particular, we are the first to study the relationship between their presence and confusion as expressed in code review comments, and their evolution across individual pull requests.

III. Methodology

As the goal of this study is to seek new insights and generate new ideas and hypotheses for future research about atoms of confusion in the context of code reviews, we decided to conduct an exploratory case study [9]. This work consists of two parts, each one of them answering one research question. As the primary data of this study, we needed to select GitHub repositories to be analyzed. Our selection criteria is that the repositories should be public repositories and have a large number of contributors (more than 100), a moderate number (1,500+) of closed PRs as well as the majority being Java code files. Not only will these criteria ensure diversity (*i.e.*, different levels of coding experience from contributors) and enough code review discussions during our analysis, this will also ensure that we can detect atoms of confusion in the source code, as our study is based on the study of atoms in the Java programming language [5].

To answer **RQ1**, we considered a single GitHub repository. After thorough investigation, we settled on the openhab-addons[3] repository, which has 300+ contributors and 8,000+ closed PRs. Then, the first two authors manually analyzed a sample of code review discussions from PRs of openhab-addons repository. The manual analysis aimed at identifying comments in which confusion is expressed. As our unit of analysis comprises specific lines of source code, *i.e.*, atoms of confusion, we decided to focus only on the *inline comments* within each PR. The rationale is that inline comments are explicitly linked to specific parts of the source code, and thus those comments are linked to the changed code. This is not possible for general comments of PRs as they cannot easily be linked to specific parts of the code change. For **RQ2**, we considered both openhab-addons and confluentinc-ksql[4]. The latter has 157+ contributors and 6,000+ closed PRs.

To answer the RQs we identify the atoms of confusion in those repositories using a tool[5] based on the Java atoms of confusion by Langhout and Aniche [5]. For **RQ1**, we apply the tool on each PR and record the presence of atoms of confusion after each PR. As for **RQ2**, since we are interested in the presence before and after each PR, we document the presence or absence of atoms before the PR as well. Then, as the goal of **RQ1** is to investigate the link between atoms of confusion and confusion in code reviews and we manually identify comments expressing confusion, we used a sample set of PRs from openhab-addons. At the time of the study, the total number of PRs was 12,100 and therefore we randomly sample 368 PRs including at least one Java file. This sample size gives us a 95% confidence level and a 5% margin of error. To classify whether a code review discussion contains confusion, we employed a cross-labelling method. Firstly, the comments are extracted from the respective PR's code review. Then, the comments are labelled by the first two authors following the approach of Ebert *et al.* [3]. Finally, the first two authors cross-checked the classification by going through the comments once more to make sure they agreed on each other's labelling. For each disagreement, they discussed with each other and decided on the final label together. As the goal is not to provide a gold standard dataset, we do not report on agreement ratio.

Since the process of **RQ2** could be automated completely, we analyse the entire population of closed PRs in both repositories. We exclude PRs that belong to issues (*i.e.*, GitHub API provides issues in the same way it provides PRs, so we need to exclude these issues) and to PRs that do not have Java files.

A. Data analysis

The statistical analyses we performed were similar for both the research questions: we used the Chi-squared [11] and

[2]https://www.ioccc.org

[3]https://github.com/openhab/openhab-addons

[4]https://github.com/confluentinc/ksql

[5]https://github.com/SERG-Delft/atoms-of-confusion-detector

Fisher Exact [12] tests and the odds ratio. Additionally, for **RQ2** we also calculated the risk ratio [13].

To carry out our statistical analysis on the obtained results, we organized the results as contingency matrices. The contingency matrix obtained from the results of **RQ1** is a 2×2 matrix, with rows corresponding to absence/presence of atoms of confusion and columns to absence/presence of confusion comments. The contingency matrix to answer **RQ2** is a 2×2 matrix with rows being how many atoms of confusion there are at the start of the PR: no atoms or some atoms, and columns of how many atoms of confusion after the PR changes. The data set produced in this study with the labeled comments and the classification of the PRs in one of the four categories is publicly available.[6]

Both the Chi-squared and the Fisher exact tests were used to investigate whether there is an association between two categorical variables. For **RQ1** the categorical variables are presence of confusion in comments in code reviews vs presence of atoms of confusion. In **RQ2**, the categorical variables are number of atoms of confusion before the PR vs the number of atoms of confusion after the PR. The odds ratio was calculated to find out the odds that an outcome will occur given a particular exposure compared to the odds of the outcome occurring in the absence of that exposure. The exposure in **RQ1** is the presence of atoms of confusion at the end of PR review. The exposure in **RQ2** is the presence of atoms of confusion at the start of the PR, specifically the number of the atoms. The risk ratio is used to carry out further analysis on the results from **RQ2**. With this measure we are able to investigate how much more or less likely it is that there are atoms of confusion at the end of a PR review if there are atoms before the PR is reviewed.

IV. Results

In this section, we present the results of our study by answering each research question (Sections IV-A and IV-B),

A. *RQ1. What is the relationship between of atoms of confusion and confusion in code reviews?*

To answer this research question, we measured the following metrics: the number of inline confusion comments and the number of atoms of confusion within the PRs. Both metrics are based on a random sample of 368 PRs from the openhab-addons repository. As explained in Section III, we created the contingency matrix, shown in Table I, with all of the results. There appears to be little to no relationship between atoms of confusion and confusion in code review comments. In order to properly verify it we ran two statistical analyses on the results, *i.e.*, the Fisher Exact and Chi-squared tests. The results of the statistical tests show there is no relationship between the presence of atoms of confusion in pull requests and confusion expressed in code review comments. The Chi-squared test resulted in a p-value of 0.498 and the Fisher Exact test resulted in a p-value of 0.532.

[6]https://figshare.com/s/9503e287fc189e92139b

	Confusion comments present	No confusion comments present
No atoms present	96	88
Some atoms present	90	95

TABLE I
CONTINGENCY MATRIX FOR **RQ1**.

At the beginning of the PR	At the end of the PR: No atoms of confusion present	At the end of the PR: Atoms of confusion present
No atoms present	554	401
Some atoms present	1287	3014

TABLE II
CONTINGENCY MATRIX FOR **RQ2**: OPENHAB-ADDONS.

B. *RQ2. What is the relationship between the presence of atoms of confusion in source code before and after PRs?*

To answer this research question, we collected the following metrics: the number of atoms of confusion at the start of the PR and the number of atoms of confusion at the end of the PR. Both of the metrics are based on all of the PRs in the openhab-addons repository as well as all the PRs in the confluentinc-ksql repository. Table II shows the contingency matrix for the openhab-addons repository, and Table III for confluentinc-ksql. The cells in these tables show numbers of files affected by PRs and whether they have atoms of confusion or not before and after the PR.

On both of the contingency matrices we did further analysis by running the Chi-squared, Fisher Exact tests, and calculated both the odds ration and the risk ratio. The results can be seen in Table IV.

From the results shown in Tables II and III as well as the tests along with the ratios, we can see both repositories following the same pattern: the majority of the pull requests from these repositories maintains the presence of atoms after the request. In other words, if a file involved in a PR contains atoms of confusion before the PR, it is likely to contain atoms of confusion after the PR as well.

By taking a more holistic look, we recognise another trend manifested in both repositories: the presence, or lack thereof, of atoms of confusion in a PR has a higher chance of remaining the same before and after each PR. This means that if there

At the beginning of the PR	At the end of the PR: No atoms of confusion present	At the end of the PR: Atoms of confusion present
No atoms present	180	73
Some atoms present	18	2720

TABLE III
CONTINGENCY MATRIX FOR **RQ2**: CONFLUENTINC-KSQL.

Test	**openhab-addons**	**confluentinc-ksql**
Chi-squared test	$7.24 \cdot 10^{-61}$	< 0.00001
Exact Fisher test	$3.51 \cdot 10^{-58}$	$6.11 \cdot 10^{-205}$
Odds ratio	3.23	366.26
Risk ratio	1.66	3.44

TABLE IV
STATISTICAL RESULTS FOR **RQ2**. THE TOP TWO ROWS PRESENT THE P-VALUES FOR THE TWO TESTS. THE BOTTOM ROWS PRESENT THE RATIOS.

	Decreases	Remains	Increases
No atoms at the start	0	554	399
Some atoms at the start	1690	1196	1415

TABLE V
RESULTS FOR **RQ2**: OPENHAB-ADDONS.

	Decreases	Remains	Increases
No atoms at the start	0	180	73
Some atoms at the start	403	1041	1294

TABLE VI
RESULTS FOR **RQ2**: CONFLUENTINC-KSQL.

is a number of atoms existing in the files involved in a pull request before the PR, it is more probable that there will still exist some atoms in these files after that PR. The same applies to the absence of these atoms: if there are no atoms in the files of a pull request before the request, it is more likely that after the pull requests no new atoms are added as well.

We can put a magnifying glass over this result by discriminating cases where the number of atoms in a file decreased, stayed constant, or increased, after each PR. Tables V and VI provide data considering these three scenarios for the two analyzed projects. The tables show that the same trend can be observed for both files: if a file originally had no atoms, it is more likely that it will remain atom-less after a PR. In addition, if there are some atoms present before a pull request in the involved files, it is more likely that the number of atoms in these files will either stay the same or increase and less likely that this number will decrease. The tables also show nuance when considering cases where there is a decrease in the number of atoms. For confluentinc-ksql, less than one in every six PRs caused the number of atoms to decrease. In openhab-addons, more than one in every three PRs had that effect.

V. DISCUSSION

The results for **RQ1** suggest that there is no relationship between the presence of atoms of confusion in source code and the presence of confusion in code review comments. We believe there might be some explanations for this. Firstly, it is possible that the developers are indeed confused with the code that has the presence of atoms. However, they might not be aware of this internal confusion, or they might express it in ways other than code review comments, for example direct communication with the author of the PR, or not at all. Therefore, we cannot measure this confusion in the code review comments in such scenarios. Secondly, this could happen due to the difference between the lab setting where the atoms of confusion were developed and the field setting. The original study [6] involved students who might not have a great command of programming languages and/or might have gotten confused by the bigger picture, *i.e.*, they might have been confused from multiple pieces of code shown to them consecutively. However, in our study, we investigate repositories which include multiple contributors, the majority of whom are professionals who might not be confused by these small atoms of confusion due to their experience and familiarity with programming and the project itself.

Regarding the results for **RQ2**, we found evidence that the atoms of confusion are usually not being removed in pull requests. This result confirms the findings of previous research [4] on atoms of confusion, but only limited to this context. For example, Tufano *et al.* [14] conducted a large empirical study and found that about 80% of code smells survive in the system, Businge *et al.* [15] found that 44% of the 512 analyzed Eclipse third-party plug-ins depend on "bad" APIs and that developers continue to use these "bad" APIs. Piantadosi *et al.* [16] observed that only a minority of the commits aim to make code more readable. These results provide evidence that developers are usually not inclined to remove or change *problems* in the code.

As for the introduction of atoms, we believe that the developers introduce them unknowingly, due to a number of reasons. Firstly, it is possible that these developers are unaware of the concept of atoms of confusion. As mentioned above, this concept was developed in a lab setting and therefore might not be popular amongst developers. This could subsequently lead to them not recognising the confusing patterns of code and adding more atoms - either by following these patterns for consistency or adding new types of atom due to their lack of knowledge about this concepts - rather than trying to resolve them. Secondly, it is possible that the developers might have indeed been confused by the atoms at first. However, rather than addressing this problem by either resolving them themselves or making the original authors aware of this and asking them to resolve it, they might spend time rereading the code and trying to understand the code, possibly with the help of others in real life or online. The consequence of this is that they will now regard these coding patterns as understandable and continue using them. Finally, the level of proficiency of the developers from these projects could have affected this increase in number of atoms in the repositories. Experiments where atoms of confusion have been shown to cause confusion involved mostly students. However, the presence of atoms of confusion might have a limited impact on the ability of professional developers to understand code, to a point where they do not even consider it something to be improved in a system under review.

A. Proposed Hypotheses

As befitting an exploratory case study [9], we propose three hypotheses about atoms of confusion, based on the results of this study:

H1. *Atoms of confusion are not perceived as confusing for experienced developers.* Future research should try to confirm whether the atoms of confusion, which so far have been confirmed with students, in different contexts, are indeed causing misunderstanding for experienced developers. It might be the case that those atoms are so small and simple that they are not a problem at all in "real-life".

H2. *Developers are unknowingly introducing atoms of confusion within the PRs.* There seems to be the case that

developers are inadvertently introducing atoms within the PRs. Future research could investigate whether an automated support suggesting developers some version without atoms of confusion is beneficial for the project and the developer's comprehension of the code.

H3. *Developers are aware of the bad effect of atoms of confusion, but are still willing to live with the consequences.* Another possibility is that developers are consciously willing to not remove these atoms and live with them in the project. This is a conjecture similar to the work of Businge *et al.* [15] where they believe developers are consciously continuing to use bad APIs, even though knowing they are volatile and unsupported.

Beyond the specific hypotheses presented above, we believe that a replication of this study with a broader set of repositories will bring full understanding of the role played by atoms of confusion. We believe that such understanding will shed new light on the development and comprehension tasks of developers.

VI. Threats to validity

Construct Validity*:* We inherit this threat from of Ebert [17] as *confusion* itself is an abstract, difficult-to-measure concept, which means this keyword - and expression-based scheme might have some inaccuracy. In order to combat this, the first two authors cross-checked the labelling of each other and in cases of disagreement, they would discuss with each other to come to a final label together.

Internal Validity*:* General comments were not included our analysis, but they may include confusion. However, since they do not have a direct link to specific parts of the code, it is not be feasible to check their relationship with atoms. Differently, inline comments do allow such analysis. Another threat is the subjectivity of our comment classification process. None of the two labellers are involved in the analyze projects. Therefore we might have made misinterpretations when reading review comments due to a lack of knowledge about the culture and guidelines of the projects.

External Validity*:* for **RQ1**, we only analysed a sample of PRs from one Java repository. Further studies are needed. As for **RQ2**, we only used two Java repositories. This means that the results only pertain to these specific projects. In order to confirm the results from our study, replication to other projects is necessary.

VII. Conclusion

The omnipresence of code reviews calls for careful consideration of mistakes, oversights, and confusion from the developers when reviewing code. In this paper, we describe the first exploratory case study to understand if there is a connection between the presence of confusion in code reviews and atoms of confusion. We also investigated how pull requests affect the presence of atoms in the files of two different projects. We collected data about pull requests and analysed them for atoms and confusion comments. Our initial results show that there is no relationship between atoms of confusion and comments expressing confusion in code reviews. Additionally, based on the two analyzed repositories, we observe that a PR that impacts files that does not include atoms is more likely to not introduce atoms in these files. In a similar vein, PRs impacting files that already have atoms are more likely to maintain or increase the number of atoms. Based on our findings, we also propose new hypotheses about atoms of confusion for future research.

Acknowledgments

We are very grateful to Mauricio Aniche, Jorge Romeu and Pavlos Makridis for their great work on the topic of atoms of confusion and also for developing and making the tool for detecting atoms of confusion available to us.

References

[1] R. Minelli, A. M. and, and M. Lanza, "I know what you did last summer: An investigation of how developers spend their time," in *ICPC*, 2015, pp. 25—-35.

[2] X. Xia, L. Bao, D. Lo, Z. Xing, A. E. Hassan, and S. Li, "Measuring program comprehension: A large-scale field study with professionals," *TSE*, vol. 44, no. 10, pp. 951–976, 2018.

[3] F. Ebert, F. Castor, N. Novielli, and A. Serebrenik, "Confusion in code reviews: Reasons, impacts, and coping strategies," in *SANER*, 2019, pp. 49–60.

[4] D. Gopstein, H. H. Zhou, P. Frankl, and J. Cappos, "Prevalence of confusing code in software projects: atoms of confusion in the wild," in *MSR*, 2018, pp. 281–291.

[5] C. Langhout and M. Aniche, "Atoms of confusion in java," in *ICPC*, 2021, pp. 25–35.

[6] D. Gopstein, J. Iannacone, Y. Yan, L. DeLong, Y. Zhuang, M. K.-C. Yeh, and J. Cappos, "Understanding misunderstandings in source code," in *ESEC/FSE*, 2017, pp. 129—139.

[7] F. Medeiros, G. Lima, G. Amaral, S. Apel, C. Kästner, M. Ribeiro, and R. Gheyi, "An investigation of misunderstanding code patterns in c open-source software projects," *EMSE*, vol. 24, no. 4, pp. 1693—1726, 2019.

[8] B. de Oliveira, M. Ribeiro, J. A. S. da Costa, R. Gheyi, G. Amaral, R. de Mello, A. Oliveira, A. Garcia, R. Bonifácio, and B. Fonseca, "Atoms of confusion: The eyes do not lie," in *SBES*, 2020, pp. 243––252.

[9] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *ESE*, pp. 131–164, 2009.

[10] D. Gopstein, A.-L. Fayard, S. Apel, and J. Cappos, *Thinking Aloud about Confusing Code: A Qualitative Investigation of Program Comprehension and Atoms of Confusion*, 2020, pp. 605—-616.

[11] K. Pearson, "X. on the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 50, no. 302, pp. 157–175, Jul. 1900.

[12] P. Sprent, *Fisher Exact Test*. Springer Berlin Heidelberg, 2011, pp. 524–525.

[13] P. Cummings, "The Relative Merits of Risk Ratios and Odds Ratios," *Archives of Pediatrics Adolescent Medicine*, vol. 163, no. 5, pp. 438–445, 05 2009.

[14] M. Tufano, F. Palomba, G. Bavota, R. Oliveto, M. D. Penta, A. De Lucia, and D. Poshyvanyk, "When and why your code starts to smell bad (and whether the smells go away)," *TSE*, vol. 43, no. 11, pp. 1063–1088, 2017.

[15] J. Businge, A. Serebrenik, and M. G. Brand, "Eclipse api usage: The good and the bad," *Software Quality Journal*, vol. 23, no. 1, p. 107–141, 2015.

[16] V. Piantadosi, F. Fierro, S. Scalabrino, A. Serebrenik, and R. Oliveto, "How does code readability change during software evolution?" *EMSE*, vol. 25, no. 6, pp. 5374–5412, 2020.

[17] F. Ebert, "Understanding confusion in code reviews," Ph.D. dissertation, Federal University of Pernambuco, Recife, Brazil, 2019. [Online]. Available: https://felipeebert.github.io/post/phd-2019/phd-2019.pdf