Qualitative Analysis

Round 3, August 02 – September 10, 2021

Topics

Method of Analysis

General. We adopted a non-probability sampling, *purposive sampling*, to select refactoring-inducing and non-refactoring-inducing PRs, according to specific criteria (that emerged from the previous round), for an in-depth investigation of their review comments and discussion while cross-referencing their detected refactoring edits. We also validated these refactorings by checking for false positives and false negatives. It is worth clarifying that we will follow that strategy until we reach a point of data saturation (when no new information emerges) [Miles and Huberman, 1994], through rounds of analysis. Accordingly, at each round, we will examine a purposive sample fitting a valuable scenario to the current aim of the analysis. We chose that sampling strategy because it provides us gaining an in-depth understanding of data by exploring scenarios suitable at each step, in line with emergent patterns or ideas [Patton, 2014]. Then, each analyst apart examined all sample's PRs. Next, one analyst checked all individual judgments in order to achieve a concluding judgment concerning the sample. As decision criteria, we considered the agreement of responses by at least two analysts. It is worth clarifying that, in such subjective decision-making, we considered the refactoring-inducement in settings where review comments either explicitly suggested refactoring edits or left any actionable recommendation that induced refactoring.

Sample 3. In the third purposive sample, to address RQ_3 to RQ_5, we considered 13 refactoring-inducing PRs from 36 ones, presenting refactoring edits that change code design (*high-level refactorings*), in order to explore whether persist the emergent patterns from sample one and sample two (which comprise less complex PRs). In specific, we took into account the following high-level refactoring types: *Pull Up Method*, *Pull Up Attribute*, *Push Down Method*, *Push Down Attribute*, *Move Class*, *Move and Rename Class*, *Move and Rename Attribute*, *Extract Superclass*, *Extract Interface*, *Extract Class*, and *Extract Subclass*. Those 13 refactoring-inducing PRs embrace a diversified setting of refactoring edits of distinct types, including only one type of refactoring (e.g., dubbo #3654) and a mix of refactoring types (e.g., incubator-iceberg #183). To deal with

RQ_1 and RQ_2, we examined a randomly selected sample of 13 non-refactoring-inducing PRs that present ten review comments – the median value of the number of review comments in 13 refactoring-inducing PRs.

Sample 3

- 13 refactoring-inducing PRs
- 13 non-refactoring-inducing PRs
- Number of review comments: 327
- Number of discussing comments: 275
- Number of subsequent commits: 126
- Number of refactoring edits: 209

Descriptive Statistics of Samples

It is worth noting, as displayed in Table 1 and Table 2, that the magnitude of both refactoring-inducing and non-refactoring-inducing PRs increases in the following order: *sample 2 < sample 1 < sample 3,* when considering the features analyzed.

Table 1 – Descriptive Statistics (Refactoring-Inducing PRs)

Sample	Feature	Average	SD	Median	IQR
Sample 1	*Number of reviewers*	2	0	2	0
	Number of review comments	5	0	5	0
	Number of subsequent commits	2.4	1.2	2	1.2
	Time to merge	3.2	6.1	1	2.2
	Number of file changes	8.2	8.7	5	7
	Number of added lines	93.1	170.5	31	56.7
	Number of deleted lines	43.3	58.3	13.5	39.5
	Number of refactoring edits	5.7	7.2	2.5	5
Sample 2	*Number of reviewers*	1.8	0.6	2	0.5
	Number of review comments	4	4.3	2	2
	Number of subsequent commits	1	0	1	0
	Time to merge	3.5	6.6	1	3
	Number of file changes	1.7	0.6	2	1
	Number of added lines	35.4	63.5	6	21.5
	Number of deleted lines	26.9	56.4	6	13
	Number of refactoring edits	1	0	1	0
Sample 3	*Number of reviewers*	3.1	1.1	3	1
	Number of review comments	15.1	13.7	10	14
	Number of subsequent commits	5.8	3.6	5	6
	Time to merge	11.5	14.8	6	8
	Number of file changes	26.5	34.6	16	22

	Number of added lines	550	1198.8	143	332
	Number of deleted lines	297.9	490.7	104	281
	Number of refactoring edits	16.1	18.6	10	12

Table 2 – Descriptive Statistics (Non-Refactoring-Inducing PRs)

Sample	Feature	Average	SD	Median	IQR
Sample 1	*Number of reviewers*	2	0	2	0
	Number of review comments	5	0	5	0
	Number of subsequent commits	1.4	0.5	1	1
	Time to merge	2.5	4	0.5	3
	Number of file changes	2.9	2	2	2
	Number of added lines	17.5	19.2	8.5	24.5
	Number of deleted lines	16.6	19.2	9	24.5
Sample 2	*Number of reviewers*	2.1	0.6	2	0
	Number of review comments	4.8	4.2	4	4
	Number of subsequent commits	1	0	1	0
	Time to merge	2.9	2.3	3	4
	Number of file changes	2.1	1.7	1	1
	Number of added lines	115.1	293	2	46
	Number of deleted lines	107.5	295.3	3	8
Sample 3	*Number of reviewers*	2.8	0.9	3	1
	Number of review comments	10	0	10	0
	Number of subsequent commits	3.8	1.9	3	2
	Time to merge	10.9	23.2	2	7
	Number of file changes	7.7	4.6	6	4
	Number of added lines	47.2	48.5	23	40
	Number of deleted lines	29.9	39.4	18	16

Concluding Remarks

(1) As shown in Table 3, *RefactoringMiner* presented precision 99.6% and recall 98.3% for refactoring detection in 66 refactoring-inducing PRs.

Table 3 – Results of the manual validation of refactoring edits mined by RefactoringMiner 1.0 (all samples)

Sample	TP	FP	FN	Precision	Recall
Sample 1	66	0	2	100%	97.06%
Sample 2	9	1	2	90%	81.8%
Sample 3	208	0	1	100%	99.5%
All samples	***283***	***1***	***5***	***99.6%***	***98.3%***

In specific, *RefactoringMiner* 1.0 mistakenly detected an edit as a *Rename Method* (kafka #6565, sample 2), while it did not detect instances of *Extract Variable* (commons-text #39, sample 1), *Extract Method* (hadoop #942, sample 1), *Inline Variable* (dubbo #3185, sample 2), *Method Attribute* (beam #6261, sample 3), and *Rename Method* (kafka #7132, sample 2) refactorings.

(2) When examining refactoring edits, we consider the arguments provided by Fowler, to explain that *high-level refactorings* are structured in terms of *low-level refactorings* [Fowler, 2000], in order to classify them. To clarify, creating a class, changing the type of attributes, renaming a method, and moving methods between classes are examples of low-level refactorings. High-level refactorings are more complex edits that structurally impact code design; thus, creating a hierarchical inheritance and moving and renaming a class between packages are examples of such high-level refactorings. For instance, kafka #4735 presents *Change Type* instances due to an *Extract Superclass* one.

It is noteworthy that we mined refactoring edits by using *RefactoringMiner* 1.0, a version available in September 2019. In such a version, 40 distinct types of refactoring may be detected (listed in Table 4). Given that, we classified the types of refactoring into low-level and high-level edits in light of their impact on code design, based on technical descriptions provided in [Fowler, 2000]. Accordingly, high-level refactorings denote edits that either impact the code structure (*Extract Superclass*, *Extract Interface*, *Extract Class*, and *Extract Subclass*) or require a higher number of checking in order to preserve the code behavior (*Pull Up Method*, *Pull Up Attribute*, *Push Down Method*, *Push Down Attribute*, *Move Class*, *Move and Rename Class*, and *Move and Rename Attribute*).

Table 4 – Classification of the refactorings detectable by RefactoringMiner 1.0

Low-level refactorings	**High-level refactorings**
Extract Method	*Pull Up Method*
Inline Method	*Pull Up Attribute*
Rename Method	*Push Down Method*
Move Method	*Push Down Attribute*
Move Attribute	*Extract Superclass*
Rename Class	*Extract Interface*
Extract and Move Method	*Move Class*
Rename Package	*Move and Rename Class*
Extract Variable	*Extract Class*
Inline Variable	*Extract Subclass*
Parameterize Variable	*Move and Rename Attribute*
Rename Variable	
Rename Parameter	
Rename Attribute	
Replace Variable with Attribute	

Replace Attribute (with Attribute) *Merge Variable* *Merge Parameter* *Merge Attribute* *Split Variable* *Split Parameter* *Split Attribute* *Change Variable Type* *Change Parameter Type* *Change Return Type* *Change Attribute Type* *Extract Attribute* *Move and Rename Method* *Move and Inline Method*	

As shown in Table 5, low-level refactorings happen more often, since we identified those edits in 229/288 (79.5%) refactoring-inducing PRs (64/68 in sample 1, 11/11 in sample 2, and 154/209, in sample 3). In sample 1, 4/68 (5.9%) of refactoring-inducing PRs, in which refactoring edits were induced by code review, present instances of *Move Class* (2) and *Move and Rename Class* (2). In sample 3, 30/140 (21.4%) of refactoring-inducing PRs, in which refactoring edits were led by authors, present instances of *Extract Class* (4), *Extract Superclass* (2), *Move Class* (3), *Pull Up Attribute* (1), *Pull Up Method* (5), *Push Down Attribute* (8), and *Push Down Method* (7). Also, 25/69 (36.2%) of refactoring-inducing PRs, in which refactoring edits were induced by code review, present instances of *Extract Interface* (1), *Extract Superclass* (2), *Move Class* (4), *Move and Rename Class* (1), *Pull Up Attribute* (3), *Pull Up Method* (10), *Push Down Attribute* (2), and *Push Down Method* (2).

In addition, from Table 1 and Table 2, we know that the magnitude of refactoring-inducing PRs increases in the following order: sample 2 < sample 1 < sample 3. Therefore, we realized that as PR magnitude increases, the number of high-level refactorings also increases (Table 5, 0 < 4 < 55).

Remark: Since sample 2 < sample 1 < sample 3, we realize that as PR magnitude increases the number of low-level refactorings decreases whereas high-level ones increases. Accordingly, we conjecture that PRs that comprise larger magnitudes, such as more than two file changes and ten added/deleted lines (Table 1, sample 2), are more likely to get code high-level refactorings at code review time. Regarding the trigger for refactoring edits, we perceive that in smaller PRs, such as from sample 1 and sample 2, code review induces more low-level and high-level refactorings than the PR authors. Nevertheless, this rule changes in larger PRs, such as from sample 3, in which we identify PR authors led more low-level and high-level refactoring edits. In addition, we realize that a single review comment may induce several refactoring instances, as occurs in flink #8222 (sample 3), which pre-

sents one *Pull Up Attribute* and five *Pull Up Method* instances induced by only one review comment.

Table 5 – Refactoring-inducing PRs by level of refactorings (all samples)			
Sample	**Refactoring inducement**	**Low-level refactorings**	**High-level refactorings**
Sample 1	*Code review*	45/68 (66.2%)	4/68 (5.9%)
	Author	19/68 (27.9%)	none
Sample 2	*Code review*	9/11 (81.8%)	none
	Author	2/11 (18.2%)	
Sample 3	*Code review*	46/209 (22%)	23/209 (11%)
	Author	108/209 (51.7%)	32/209 (15.3%)
All samples	***Code review***	**99/288 (34.4%)**	**27/288 (9.4%)**
	Author	**130/288 (45.1%)**	**32/288 (11.1%)**
Total		***229/288 (79.5%)***	***59/288 (20.5%)***

(3) We identified refactoring edits led only by authors in 9/37 (24.3%) of refactoring-inducing PRs, comprising 3/13 (23.1%) in sample 1 (beam #4460, flink #7971, samza #1030), 2/11 (18.2%) in sample 2 (incubator-pinot #479, kafka #5423), and 4/13 (30.8%) in sample 3 (beam #6261, dubbo #3654, kafka #6657, usergrid #102). Moreover, as we can see in Table 6, 28/37 (75.7%) of refactoring-inducing PRs are due to code review, being the refactoring edits due to both code review and led by the PRs' authors in 8/28 (28.6%) PRs.

Table 6 – Inducement by code review in refactoring-inducing PRs (all samples)				
Sample	**Inducement by code review**	**PRs**	**(also) Led by the author**	**PRs**
Sample 1	10/13 (76.9%)	dubbo # 3299, commons-text #39, flink #9143, fluo #837, hadoop # 942, incubator-iceberg #254, kafka #5194,	3/10 (30%)	dubbo #2279, flink #7970, flink #7945
Sample 2	9/11 (81.8%)	beam #4407, beam #4458, brooklyn-server #1049, dubbo #3185, kafka #5784, kafka #7132, samza #1051, servicecomb-java-chassis #346 , tomee #275	0/9 (0%)	
Sample 3	9/13 (69.2%)	cloudstack #2071, cloudstack #3454, kafka #4757, kafka #5590	5/9 (55.5%)	flink #8222, incubator-iceberg #119, incubator-iceberg #183, kafka #4735, servicecomb-java-chassis #678
All samples	***28/37 (75.7%)***		***8/28(28.6%)***	

Conclusion: We realize a decrease in the proportion of refactoring-inducing PRs due to code review and an increase in the proportion of refactoring-inducing PRs led by PRs' authors as the complexity of PRs increase (Table 6). Thus, it seems that refactoring-inducing PRs tend to be likely due to code review in smaller PRs (in terms of code churn, number of file changes, and number of subsequent commits, as we can see in Table 1).

(4) Table 7 and Table 8 respectively present the kinds of refactoring edits and targets addressed in refactoring-inducing PRs analyzed. As we can see in Table 7, *Change Type* (36/68, 63/209) and *Rename* (21/68, 38/209) instances are the most common kinds of refactoring identified in samples 1 and 3, whereas *Extract* and *Rename* instances are the most frequent in sample 2 (in which PRs are smaller). Such a high occurrence of *Change Type* in sample 1 and sample 3 may be explained due to a few instances of low-level refactorings compose high-level ones, as we argued in (2).

Table 7 – Kinds of refactoring edits in refactoring-inducing PRs (all samples)

Sample	Refactoring led by authors		Refactoring induced by code review	
Sample 1	*Change Type*	*11*	*Change Type*	*25*
			Extract	*2*
			Move	*2*
	Rename	*8*	*Move and Rename*	*2*
			Rename	*13*
			Replace	*5*
	19/68 (27.9%)		49/68 (72.1%)	
Sample 2	*Change Type*	*1*	*Change Type*	*1*
			Extract	*3*
	Extract	*1*	*Inline*	*1*
			Rename	*3*
			Split	*1*
	2/11 (18.2%)		9/11 (81.8%)	
Sample 3	*Change Type*	*32*	*Change Type*	*31*
	Extract	*12*	*Extract*	*7*
	Extract and Move	*18*	*Extract and Move*	*3*
	Inline	*1*	*Move*	*4*
	Merge	*1*	*Move and Rename*	*1*
	Move	*20*	*Pull Up*	*13*
	Pull Up	*6*	*Push Down*	*4*
	Push Down	*15*	*Rename*	*6*
	Rename	*32*		
	Replace	*2*		
	Split	*1*		
	140/209 (66.9%)		69/209 (33.1%)	

When dealing with refactoring targets, we can explore the presence of *top-level code structures* (package, interface, class, and attribute) and *low-level* ones (method, parameter, variable). Accordingly, we note that sample 3 presents a higher proportion of top-level code structures as refactoring targets than sample 1 and sample 2 (Table 8), potentially explained by the higher number of refactorings that change code design in sample 3 (Table 5).

Table 8 – Targets of refactoring edits in refactoring-inducing PRs (all samples)			
Sample	**Target**	**Refactoring led by authors**	**Refactoring induced by code review**
Sample 1	*Class*	*1/19 (5.3%)*	*6/49 (12.2%)*
	Attribute	*3/19 (15.8%)*	*5/49 (10.2%)*
	Method	*5/19 (26.3%)*	*18/49 (36.7%)*
	Parameter	*7/19 (36.8%)*	*1/49 (2.1%)*
	Variable	*3/19 (15.8%)*	*19/49 (38.8%)*
Sample 2	*Class*	*none*	*1/9 (11.1%)*
	Attribute		*2/9 (22.2%)*
	Method		*4/9 (44.5%)*
	Parameter		*none*
	Variable	*2/2 (100%)*	*2/9 (22.2%)*
Sample 3	*Package*	*none*	*1/69 (1.5%)*
	Interface		*1/69 (1.5%)*
	Class	*10/140 (7.1%)*	*7/69 (10.1%)*
	Attribute	*51/140 (36.4%)*	*9/69 (13.0%)*
	Method	*55/140 (39.3%)*	*30/69 (43.5%)*
	Parameter	*6/140 (4.3%)*	*19/69 (27.5%)*
	Variable	*18/140 (12.9%)*	*2/69 (2.9%)*

Remark: Regardless of PR magnitude and refactoring inducement (either induced by code review or led by the author), it seems that bottom-level code structures (method, parameter, and variable) tend to be likely refactoring targets than top-level ones (package, interface, class, attribute). We can partially explain such a pattern due to high-level refactorings are defined in terms of low-level ones.

(5) As shown in Table 9, 3/37 (8.1%) refactoring-inducing PRs present self-affirmed refactorings. Such edits are explicit in the commit message by using keywords like "Refactor…", "Mov…", "Renam...".

Conclusion: Self-affirmed refactorings are rare in three samples, since only 3/37 (8.1%) of refactoring-inducing PRs present self-affirmed refactorings (Table 9).

Table 9 – Number of PRs containing self-affirmed refactorings in their commits (all samples)		
Sample	**Number of PRs**	**PRs**
Sample 1	1/13 (7.7%)	flink #7971
Sample 2	0/11 (0%)	
Sample 3	2/13 (15.4%)	beam #6261, usergrid#102
All samples	***3/37 (8.1%)***	

(6) After three rounds of analysis, we notice that both refactoring-inducing and non-refactoring-inducing PRs comprise the three primary types of changes (adaptive, corrective, and perfective), as indicated in Table 10. Our classification follows the descriptions given by [Swanson, 1976] and the identifica-

tion method provided by [Mockus and Votta, 2000]. To clarify, we explored PR descriptions and commit messages by searching for keywords that could denote the type of changes; for instance, keywords such as *"fix"* and *"correct"* indicate corrective changes.

Conclusion: It seems that there is no relationship between the type of change and refactoring-inducement (Table 10) since we realize a slightly similar proportion of the types of changes in both refactoring-inducing and non-refactoring-inducing PRs, except for corrective changes in non-refactoring-inducing PRs (almost 45% of the PRs). This scenario may denote that, when dealing with corrective changes, PRs tend to do not treat refactoring concerns. Moreover, we explored all samples in order to identify any pattern related to types of refactorings against types of changes (Table 11), however, no one emerged. To exemplify, we identified *Change Type* instances in refactoring-inducing PRs consisting of perfective changes, in which refactoring edits were led by the authors, from sample 1 and sample 2; however, such a pattern did not remain in sample 3. We also perceived perfective changes in refactoring-inducing PRs, in which refactorings were induced by code review in sample 3, but such a setting does not appear in sample 1 and sample 2.

Table 10 – Type of changes by category of PRs (all samples)

Sample	Category	Type of change		
		Adaptive	***Corrective***	***Perfective***
*Sample 1**	*Refactoring-inducing PRs*	3/13 (23.1%)	4/13 (30.8%)	5/13 (38.5%)
	Non-refactoring-inducing PRs	1/7 (14.3%)	4/7 (57.1%)	2/7 (28.6%)
*Sample 2***	*Refactoring-inducing PRs*	4/11 (36.4%)	4/11 (36.4%)	2/11 (18.2%)
	Non-refactoring-inducing PRs	3/9 (33.3%)	5/9 (55.5%)	none
*Sample 3****	*Refactoring-inducing PRs*	7/13 (53.8%)	3/13 (23.1%)	2/13 (15.4%)
	Non-refactoring-inducing PRs	4/13 (30.8%)	4/13 (30.8%)	4/13 (30.8%)
All samples	***Refactoring-inducing PRs***	***14/37 (37.8%)***	***11/37 (29.7%)***	***9/37 (24.3%)***
	Non-refactoring-inducing PRs	***8/29 (27.6%)***	***13/29 (44.8%)***	***6/29 (20.7%)***

Adaptive: *refactoring-inducing PRs – flink #7945, hadoop #942, samza 1030 (sample 1); beam #4458, incubator-pinot #479, samza #1051, servicecomb-java-chassis #346 (sample 2); beam #6261, flink #8222, incubator-iceberg #119, kafka #4735, kafka #4757, kafka #5590, servicecomb-java-chassis #678 (sample 3); and non-refactoring-inducing PRs – incubator-iotdb #342 (sample 1); beam #5785, beam #6050, beam #7696 (sample 2); dubbo #3447, dubbo #4208, incubator-iotdb #67, servicecomb-java-chassis #744 (sample 3).*

Corrective: *refactoring-inducing PRs – flink #7970, flink #7971, flink 9143, fluo 837 (sample 1); beam #4407, brooklyn-server #1049, kafka #5784, kafka #7132 (sample 2); incubator-iceberg #183, kafka #6657, usergrid #102 (sample 3); and non-refactoring-inducing PRs – brooklyn-server #411, dubbo #4870, flink #91, flink #4055 (sample 1); dubbo #3317, flink #9451, kafka #5219, kafka #6565, kafka #6818 (sample 2); cloudstack #2706, cloudstack #3276, fluo #929, kafka #6298 (sample 3).*

Perfective: *refactoring-inducing PRs – beam #4460, commons-text #39, dubbo #3299, incubator-iceberg #254, kafka #5194 (sample 1); dubbo #3185, kafka #5423 (sample 2); cloudstack #2071, cloudstack #3454 (sample 3); and non-refactoring-inducing PRs – beam #5772, kafka #5111 (sample 1); cloudstack #2553, dubbo #3184, kafka #6438, servicecomb-java-chassis #691 (sample 3).*

** 1/13 (7.7%) of refactoring-inducing PRs presented both adaptive and corrective changes (dubbo #2279).*

*** 1/11 (9.1%) of refactoring-inducing PRs and 1/9 (11.1%) of non-refactoring-PRs presented both corrective and perfective changes (tomee #275, tinkerpop #524).*

**** 1/13 (7.7%) of refactoring-inducing PRs and 1/13 (7.7%) presented both corrective and perfective changes (dubbo #3654, cloudstack #2714).*

Table 11 – Kinds of refactoring by type of change in refactoring-inducing PRs				
Sample	**Refactoring inducement**	**Type of change**		
		Adaptive	***Corrective***	***Perfective***
*Sample 1**	*Code review*	Extract Move and Rename** Rename	Change Type Rename	Change Type Extract Move** Replace
	Author	Change Type Rename	Change Type Rename	Change Type
*Sample 2****	*Code review*	Extract Rename	Extract Rename Split	Inline
	Author	Extract	none	Change Type
*Sample 3*****	*Code review*	Change Type Extract** Extract and Move Merge Move Move and Rename** Pull Up** Push Down** Rename Replace	Extract** Pull Up**	Change Package Move** Push Down** Split
	Author	Change Type Extract** Extract and Move Rename	Change Type Extract** Extract and Move Inline Move** Pull Up** Push Down** Rename	none

** In dubbo #2279, consisting of both adaptive and corrective changes, we found Change Type and Rename instances.*
*** Those instances comprise edits that change code design.*
**** In tomee #275, consisting of both corrective and perfective changes, we found a Change Type instance.*
***** In dubbo #3654, consisting of both corrective and perfective changes, we found a Move instance (Move Class).*

(7) A repository's code review bot is easily detectable since it left a comment in the PR, including the bot commands performed. For instance, the Apache *flinkbot* checks the PR description, whether a PR needs attention from a specific reviewer, the architecture, and the overall code quality. As a whole, only 6/37 (16.2%) of refactoring-inducing PRs and 1/29 (3.4%) of non-refactoring-inducing PRs ran a code review bot (Table 12).

Conclusion: There is no relationship between running a code review bot and refactoring-inducement since we found both refactoring-inducing and non-refactoring-inducing PRs that ran code review bots (Table 12).

Table 12 – Presence of code review bot in PRs (all samples)				
Sample	**Refactoring-inducing PRs**	**PRs**	**Non-refactoring-inducing PRs**	**PRs**
Sample 1	5/13 (38.5%)	flink #7970, flink #7971, flink #7945, flink #9143, hadoop	none	

		#942		
Sample 2	none		1/9 (11.1%)	flink #9451
Sample 3	1/13 (7.7%)	flink #8222	none	
All samples	**6/37 (16.2%)**		**1/29 (3.4%)**	

(8) In three rounds of analysis, we found both refactoring-inducing and non-refactoring-inducing PRs presenting none PR description (Table 13). We performed such an action to explore some potential particularity of the PR descriptions' content that could impact in the refactoring-inducement.

Conclusion: There is no relationship between PR description and refactoring-inducement, based on Table 13. Specifically, Sample 2, which comprises smaller PRs than the other two samples, presents a higher proportion of PRs containing none PR description. We speculate that such a setting is due to the lesser complexity of the PRs.

Table 13 – Presence of none PR description in PRs (all samples)

Sample	Refactoring-inducing PRs	PRs	Non-refactoring-inducing PRs	PRs
Sample 1	2/13 (15.4%)	commons-text #39, hadoop #942	1/7 (14.3%)	flink #4055
Sample 2	4/11 (36.4%)	beam #4458, incubator-pinot #479, kafka #5423, tomee #275	4/9 (44.4%)	beam #5785, beam #7696, kafka #5219, kafka #6565
Sample 3	2/13 (15.4%)	servicecomb-java-chassis #678, usergrid #102	0/13 (0%)	
All samples	***8/37 (21.6%)***		***5/29 (17.2%)***	

(9) We found self-affirmed minor PR (in which, a title or a description self-declares a minor PR) in both refactoring-inducing and non-refactoring-inducing PRs (Table 14). Moreover, we found self-affirmed minor review comments (in which, a reviewer declares a review comment as a minor) in both refactoring-inducing and non-refactoring-inducing PRs (Table 15). We performed such an exploration in order to investigate potential patterns that could emerge from that self-affirmations; for instance, would they be present only in non-refactoring-inducing PRs? In specific, we observed self-affirmed review comments that induced instances of *Rename* (flink #7945 in sample 1; kafka 5784 in sample 2), *Split* (brooklyn-server #1049 in sample 2), *Inline* (dubbo #3185 in sample 2), and *Extract* (kafka #4735 in sample 3).

Conclusion: There is no relationship between self-affirmed minor PR/review comments and refactoring-inducement since we observed self-affirmed minor PRs/review comments in both refactoring-inducing and non-refactoring-inducing PRs (Table 14, Table 15), including self-affirmed minor review comments inducing refactoring edits.

Table 14 – Presence of self-affirmed minor PRs (all samples)

Sample	Refactoring-inducing PRs	PRs	Non-refactoring-inducing PRs	PRs
Sample 1	1/13 (7.7%)	kafka #5194	2/7 (28.6%)	brooklyn-server #411, kafka #5111

Sample 2	1/11 (9.1%)	kafka #5423	0/9 (0%)	
Sample 3	1/13 (7.7%)	kafka #5590	1/13 (7.7%)	kafka #6438
All samples	***3/37 (8.1%)***		***3/29 (10.3%)***	

Table 15 – Presence of self-affirmed minor review comments (all samples)				
Sample	**Refactoring-inducing PRs**	**PRs**	**Non-refactoring-inducing PRs**	**PRs**
Sample 1	1/13 (7.7%)	flink #7945	2/7 (28.6%)	brooklyn-server #4111, flink #91
Sample 2	3/11 (27.3%)	brooklyn-server #1049, dubbo #3185, kafka #5784	2/9 (22.2%)	beam #5785, flink #9451
Sample 3	3/13 (23.1%)	kafka #4735, kafka #4757, kafka #6657	0/13 (0%)	
All samples	***7/37 (18.9%)***		***4/29 (13.8%)***	

(10) By observing the age of the PRs (difference between the date of repository's creation and date of PR creation, in number of years), we realize that newer refactoring-inducing PRs present refactoring edits that change code design (Table 16). We carried out such an action to explore some potential particularity of refactoring edits against repositories maturity.

Table 16 – Age of PRs (number of years)			
Sample	**Statistic**	**Refactoring-inducing PRs**	**Non-refactoring-inducing PRs**
Sample 1	*Average*	6	4.8
	SD	2.6	1.8
	Median	6.2	5.4
	IQR	4.7	3
	Outliers	none	none
Sample 2	*Average*	5.4	5.3
	SD	3.3	1.8
	Median	5.7	5.5
	IQR	3.5	2.9
	Outliers	12.9	none
Sample 3	*Average*	4.9	5.6
	SD	2.6	2.7
	Median	5.2	6.6
	IQR	4.4	4.8
	Outliers	none	none

Conclusion: It seems that refactoring edits changing code design is more likely to happen in newer refactoring-inducing PRs, maybe denoting their repositories' code evolution.

(11) Intending to investigate the experience of PR authors and reviewers, we explored their participation in GitHub profiles. It is worth emphasizing that we considered the number of contributions instead of contributions itself due to a high number of them, which expresses a time-intensive task, as we can see in Table 17. Note that reviewers present experience in code review as from 2016[1]; accordingly, there may be more experienced reviewers than noted. Not all PRs' participants indicate their Apache's roles; thus, Table 18 presents numbers inferred from examining the contributors' profile. We elucidate that we did not include PRs with no review in Java code to calculate the contributors' experience.

From that, it seems that authors of refactoring-inducing PRs are less experienced than non-refactoring-inducing PR authors (Table 17), and that reviewers of refactoring-inducing PRs are slightly more experienced than reviewers of non-refactoring-inducing PRs (Table 17, Table 18). We also noted that authors of refactoring-inducing PRs are more experienced than reviewers of refactoring-inducing PRs.

Conclusion: We conjecture that authors of non-refactoring-inducing PRs develop less problem-prone code and, therefore, less refactoring-prone code, because they are more experienced developers.

Table 17 – Stats of PRs' participants by category (all samples)

Sample	Category	Stats	Author	Reviewer
Sample 1	*Refactoring-inducing PRs*	Average	668.9	2801.6
		SD	853.3	3417.6
		Median	424	1414
		IQR	794	3698
		Outliers	3127	14442
	Non-refactoring-inducing PRs	Average	5375.6	2907.6
		SD	10982.5	4358.2
		Median	511	1373
		IQR	12862.5	1228
		Outliers	none	12678
Sample 2	*Refactoring-inducing PRs*	Average	758.2	956.1
		SD	547.5	974.9
		Median	1032	438
		IQR	1024.5	1909
		Outliers	3699, 28964	10853, 14688
	Non-refactoring-inducing PRs	Average	3137	1645.5
		SD	4058.6	1311.3
		Median	1674.5	1639

1 We believe this is due to the inclusion of more contributing activities on GitHub in 2016, available in https://github.blog/2016-05-19-more-contributions-on-your-profile/.

		IQR	3499	2765
		Outliers	11965	none
Sample 3	*Refactoring-inducing PRs*	Average	723.5	2683.1
		SD	1447.2	4637.7
		Median	99	912
		IQR	697	2543.5
		Outliers	5254	14216, 15197, 18995
	Non-refactoring-inducing PRs	Average	2091.1	2432.4
		SD	3114.8	4872.1
		Median	380.5	828
		IQR	3752	1190
		Outliers	none	5808, 18728, 18866

Table 18 – Apache roles of PRs' participants by category (all samples)

Sample	Category	Author	Reviewer
Sample 1	*Refactoring-inducing PRs*	2 committers 1 commiter/PMC	5 PMC 2 committers 3 commiters/PMC 1 PMC 'incubating'
	Non-refactoring-inducing PRs	1 committer/PMC	1 PMC 2 committer/PMC
Sample 2	*Refactoring-inducing PRs*	1 committer 1 commiter/PMC	4 PMC 3 commiters/PMC
	Non-refactoring-inducing PRs	3 committer/PMC	4 PMC 4 committers/PMC
Sample 3	*Refactoring-inducing PRs*	none	1 committer 10 PMC 3 commiters/PMC
	Non-refactoring-inducing PRs	none	1 committer 14 PMC 1 committers/PMC

Research Questions

- **RQ_1 How are refactoring-inducing and non-refactoring-inducing PR's review comments characterized?**

Refactoring-inducing PRs

Table 19 summarizes the emerged characteristics of code reviews in refactoring-inducing PRs, in which refactoring edits were submitted entirely by the authors. In specific, review comments directly address **code aesthetics**; they present **questions on simpler issues regarding code logic** in contrast to those in PRs, which refactorings were induced by code review, sometimes giving reasons; and/or

they provide **suggestions on code logic**. The questions concern a few issues not related to potential refactorings, and the authors give answers by providing clarifications properly. The suggestions are direct using terms such as *"could we use...?"*, *"use..."*, *"can be replaced with..."*.

Table 19 – Characteristics of review comments in refactoring-inducing PRs, in which refactorings were led by the authors (all samples)	
Characteristic	**Examples/PRs**
Addressing code aesthetics	• *Code format* (flink #7971 in sample 1)
Questioning simple issues on code logic	• *Access by name or index* (beam #4460 in sample 1) • *Error in conditional statement* (kafka #6657 in sample 3) • *Treatment of empty values* (samza #1030 in sample 1) • *Method calls* (questioning the effect of a call at beam #6261 in sample 3)
Suggesting improvements to the code	• *Adding case tests* (kafka #5423 in sample 2, incubator-pinot #479 in sample 3) • *Method calls* (proposing a method replacement in dubbo #3654, kafka #6657 in sample 3) • *Adding code documentation* (beam #6261 in sample 3) • *Change the content of test files* (usergrid #102 in sample 3) • *Use of assertion* (incubator-pinot #479 in sample 2)

Table 20 indicates the emerged characteristics of reviews in refactoring-inducing PRs, in which refactoring edits were induced only by code review. From examining those PRs, we observed that review comments **question** about **code logic**, **suggest improvements** to the code, and provide **warnings** on good development practices.

Uncertain review comments (like *"I'm wondering if..."*) are not welcome; for instance, a reviewer wondering whether another strategy is appropriate to deal with code logic (flink #9143 in sample 1 and kafka #7132 in sample 2) led to discussion and no effect.

In specific, the author of dubbo #3299 ignored a review comment containing an **inline code** (for treating a variable status), whereas a review comment suggesting (*"what do you say...?"*) an inline code as an alternative (for simplifying a code) was embraced by the author of dubbo #3185. We speculate that such scenarios may affect code ownership, so explaining the behavior of PR authors. Thus, it seems that using inline code is appreciated only when it is a suggestion, but not when it is a proposal of the reviewer to change the original code.

Table 20 – Characteristics of review comments in refactoring-inducing PRs, in which refactorings were induced by code review (all samples)	
Characteristic	**Examples/PRs**
Questioning issues on code logic	• *Use of specific types* (dubbo #3299 in sample 1) • *Method calls* (a doubt at fluo #837 in sample 1, a question on the use a single instance to distinct arguments at beam #4407 in sample 2, a question on a method call as a substitute for other ones at kafka #4757 and kafka #5590 in sample 3) • *Method signature* (*"Any reason to…?"* at commons-text #39 in sample 1)
Suggesting improvements to the code	• *Refactoring* (flink #9143, hadoop #942, incubator-iceberg #254, kafka #5194 in sample 1; beam #4458, kafka #5784, kafka #7132, samza #1051,

	servicecomb-java-chassis #346, tomee #275 in sample 2; cloudstack #2071, cloudstack #3454, kafka #4757, kakfa #5590 in sample 3) • *Adding code documentation* (due to obfuscated name at fluo #837 in sample 1; to deal with a moving refactoring at kafka #5194 in sample 1; due to an code update at bem #4407 in sample 2; asking for comments in English at servicecomb-java-chassis #346 in sample 2) • *Adding case tests* (flink #9143, kafka #5194 in sample 1, beam #4458 in sample 2) • *Adding an exception handling* (kafka #5590 in sample 3) • *Discard a method*, by providing explanations regarding the code design (incubator-iceberg #254 in sample 1) • *Issues on UI layout* (cloudstack #3454 in sample 3)
Warning on good development practices	• *Code conventions* (brooklyn-server #1049 in sample 2)

In refactoring-inducing PRs, comprising refactoring edits both led by the authors and induced by code review, the review comments present characteristics that already emerged from exploring the refactoring-inducement in refactoring-inducing PRs (Table 21). Uncertain review comments remain in such a scenario, where the sentences include terms as *"Looks like..."* and *"not sure if..."* at incubator-iceberg #183 in sample 3.

Table 21 – Characteristics of review comments in refactoring-inducing PRs, in which refactorings were led by the authors and induced by code review (all samples)

Characteristic	Examples/PRs
Addressing code aesthetics	• *Indentation and code format* (incubator-iceberg #119, incubator-iceberg #183 in sample 3)
Questioning issues on code logic	• *Use of specific types* (dubbo #2279 in sample 1; kafka #4735 in sample 3) • *Method calls* (missing a method implementation at dubbo #2279 in sample 1, doubt on return type of a method at flink #8222 in sample 3, generalization of methods at incubator-iceberg #183 in sample 3, the content of a method output at kafka #4735 in sample 3, optimization in a method invocation at servicecomb-java-chassis #678 in sample 3)
Suggesting improvements to the code	• *Refactoring* (flink #7945 in sample 1; flink #8222, incubator-iceberg #119, incubator-iceberg #183, kafka #4735, servicecomb-java-chassis in sample 3) • *Adding an exception handling* (dubbo #2279, flink #7970 in sample 1) • *Adding code documentation* (to make consistent with other classes at flink #8222 in sample 3, to update a method description after a change at incubator-iceberg #119 in sample 3) • *Adding case tests* (flink #8222, incubator-iceberg #119 in sample 3)
Warning on good development practices	• *Casting problems* (flink #7970 in sample 1) • *Use of methods in tests* (incubator-iceberg #183 in sample 3) • *Use of global instances* (servicecomb-java-chassis #678 in sample 3)

Based on Table 19-21, we realize that review comments deal with more complex issues regarding code logic and concerns on good development practices when code reviewing induces refactoring edits.

Non-refactoring-inducing PRs

Firstly, it is worth mentioning that, even in Java repositories, PRs can present a few non-Java files (e.g., Scala and Python code), sometimes resulting in no review comment in Java files. Such characteristic is common in 8/29 (27.6%) of non-refactoring-inducing PRs examined (beam #5772, flink #4055, incubator-iotdb #342 in sample 1; beam #5785, beam #7696, flink #9451 in sample 2; cloudstack #2706, kafka #6298 in sample 3).

We found review comments including inline code as a proposal for substituting the author's code ("should we change..."), so obtaining no effect (dubbo #4870, kafka #5111 in sample 1). However, such a structure of review comment is welcomed when the author asks for support from the reviewer, as it occurs at fluo #929 in sample 3. Therefore, it seems that using inline code is appreciated when it is a suggestion, but not when the reviewer proposes to change the original code, as we previously argued.

Review comments in non-refactoring-inducing PRs directly address **minor issues on code logic**, through **questions** and **suggestions** (Table 22). To clarify, those minor issues denote those subjects that have no impact related to refactoring, such as fixing an error in conditional statements and using the correct range of variable's values.

Usually, the authors are able to give answers and arguments to the reviewers, even in the presence of discussions, so resulting in no effect on the code (brooklyn-server #411, dubbo #4870, kafka #5111 in sample 1; beam #7696, dubbo #3317, kafka #6818, tinkerpop #524 in sample 2; cloudstack #2553, cloustack #2714, cloudstack #3276, dubbo #3184, incubator-iotdb #67, servicecomb-java-chassis #691, servicecomb-java-chassis #744 in sample 3).

Table 22 – Characteristics of review comments in non-refactoring-inducing PRs (all samples)

Characteristic	Examples/PRs
Examining code aesthetics	• *Extra blank lines, code format* (dubbo #4208, fluo #929 in sample 3)
Addressing issues on code logic	• *Error in conditional statements* (kafka #6818 in sample 2; fluo #929, servicecomb-java-chassis #691 in sample 3) • *Exception handling* (brooklyn-server #411 in sample 1; cloudstack #2553, servicecomb-java-chassis #691 in sample 3) • *Method calls* (using another version of an overloaded method at beam #6050 in sample 2, doubt concerning a specific method invocation at dubbo #3447, dubbo #3184, servicecomb-java-chassis #691, servicecomb-java-chassis #744 in sample 3; suggestion of a method overrinding at cloudstack #2714 in sample 3) • *Reverting of changes* (kafka #5111 in sample 1; cloudstack #3276, cloudstack #2553, dubbo #3447, incubator-iotdb #67 in sample 3) • *Removing of code fragments* (dubbo #4870 in sample 1; dubbo #3317, kafka #5219, kafka #6818 in sample 2; dubbo #3184, servicecomb-java-chassis #744 in sample 3)
Suggesting improvements to the code	• *Adding assertion* (dubbo #3447 in sample 3) • *Adding case tests* (kafka #6818 in sample 2; cloudstack #2714, cloudstack #3276, fluo #929 in sample 3) • *Adding code documentation* (brooklyn-server #411 in sample 1;

	servicecomb-java-chassis #691) • *Fixing code documentation* (beam #6050 in sample 2; dubbo #4208 in sample 3) • *Alternatives to code fragments* (kafka #5111 in sample 1; beam #6050, tinkerpop #524 in sample 2; cloudstack #3276, servicecomb-java-chassis #691 in sample 3) • *Alternatives to error/output messages* (dubbo #3317 in sample 2; fluo #929, incubator-iotdb #67, servicecomb-java-chassis #691 in sample 3) • *Fixing a typo* (flink #91 in sample 1) • *Use of better argument values* (kafka #6565 in sample 2; kafka #6438 in sample 3)

- **RQ_2 What are the differences between refactoring-inducing and non-refactoring-inducing PRs, in terms of review comments?**

We realized that review comments in refactoring-inducing and non-refactoring-inducing PRs are different concerning the following criteria:

- **Issues addressed**. Based on Table 19–22, code review addresses **code aesthetics**, **code logic**, and **improvements** in both refactoring-inducing and non-refactoring-inducing PRs. From that, we realize that review comments concern minor issues on code logic and suggestions of minor improvements to the code in non-refactoring-inducing PRs than in refactoring-inducing PRs. In order to clarify, reviewers ask questions regarding issues of minor scope in non-refactoring-inducing PRs (e.g., doubt on a version of an overloaded method), whereas they deal with major issues (e.g., doubt on the return type of a method) that can induce refactoring, in refactoring-inducing PRs. Also, reviewers provide suggestions of minor improvements to the code (e.g., alternatives to output messages) in non-refactoring-inducing PRs, whereas directly suggest refactoring (e.g., alternative to a method signature) in refactoring-inducing PRs. The same patterns emerged when comparing review comments in refactoring-inducing PRs, where refactoring edits were led by the author (Table 19), and non-refactoring-inducing PRs (Table 22). In such a context, we found questions on minor issues on code logic in non-refactoring-inducing PRs (e.g., a doubt concerning a method call) than in refactoring-inducing PRs (e.g., a question on the effect of a method call); and suggestions of minor improvements in non-refactoring-inducing PRs (e.g., proposing the use of “A”, ”B”… instead of “X”, “Y”… as a method arguments) than in refactoring-inducing PRs (e.g., suggesting a method replacement). In specific, we found **warnings on good practices of development** in refactoring-inducing PRs that, in turn, may induce refactoring (Table 20–21).

- **Discussion**. We found reviewing discussion in 12/37 (32.4%) refactoring-inducing PRs and in 10/21 (47.6%) non-refactoring-inducing PRs. We also detected reviewing discussion in two non-refactoring-inducing PRs (dubbo #3447 and dubbo #3184), in which reviewers agree with each other when addressing code issues. In refactoring-inducing PRs, reviewing

discussion tends to arise as from either warning on good practices of development (authors may require explanations due to no knowledge of such practices, as it occurs in kafka #4757) or questions/opinions concerning code logic, such as addressing potential failures and alternatives to specific decisions making (e.g., change a package name to meet project standards may break external plugins making use of a service, as it happens in servicecomb-java-chassis #678). We realize that authors properly provide clarifications for questions/opinions of reviewers. The reviewing discussion in non-refactoring-inducing PRs clearly fits a pattern: everything begins with a review comment suggesting some change to the code, followed by direct arguments from the author to refuse such a suggestion. As a result, the author wins the discussion.

- **Structure of the content**. We realize that, usually, review comments are more polite in refactoring-inducing PRs than in non-refactoring-inducing PRs. They consist of sentences that incorporate expressions like *"Maybe we should..."*, *"Should we...?"*, and *"How about...?"*, so triggering the authors to think better concerning the code under review, which tends to result in refactoring. Furthermore, we identify a lot of review comments that impose a change in non-refactoring-inducing PR through using terms such as *"...should be..."*, which tends to make no effect.

- **Experience of reviewers**. By counting the number of contributions of authors and reviewers (Table 17), we realize that review comments are submitted by reviewers more experienced than authors in refactoring-inducing PRs, whereas the opposite happens in non-refactoring-inducing PRs. Accordingly, the experience of authors of non-refactoring-inducing PRs reflects in both characteristics of their code and their ability to provide clarifications and win discussions.

- **RQ_3 How do reviewers suggest refactorings in review comments in refactoring-inducing PRs?**

In refactoring-inducing PRs, the suggestions of refactoring are usually **polite and direct** that, in turn, are well-understood by PRs' authors, using terms such as *"Maybe…", "Should this be…?", "How about renaming…?", "...should be moved into…?", "...can/could be replaced with…"*, *"...you can replace the code…"*, *"should we use…?"*, *"I think we should..."*, *"I think it would be better..."*, *"this could also be..."*. We conjecture that the experience of reviewers against authors, in refactoring-inducing PRs, might explain such a pattern. Also, sometimes, the reviewers provide a **rationale** for their suggestions, such as performance at commons-text #39 in sample 1; for separating a class by functionality at flink #7945 in sample 1; naming consistency at kafka #5784 in sample 2; change type to avoid boxing/unboxing at tomee #275 in sample 2; changing an outdated service at cloudstack #2071

in sample 3; security-related issues at cloudstack #3454 in sample 3; for preserving of orders and naming consistency at flink #8222 in sample 3; to deal with efficiency issues and to avoid duplication at incubator-iceberg #119 in sample 3; adding a utility function at kafka #4757 in sample 3; explaining the need for an interface at kafka #5590 in sample 3. In brief, the purpose of a rationale is possibly to let the author know that an adjustment is not necessary for the code operation, but to bring more benefits, such as readability or maintain a coding standard among the other repository's developers.

Review comments, in the form of warning on good practices of development, may induce refactoring and provide **explanations and examples** (using structures of the code under analysis, e.g., a class, a method) of how to deal with problematic points (e.g., casting problems at flink #7970 in sample 1, single responsibility of methods at incubator-iceberg #119 in sample 3, the use of methods in tests at incubator-iceberg #183 in sample 3, issues related to global instances at servicecomb-java-chassis #678 in sample 3).

- **RQ_4 Do refactoring suggestions justify the reasons?**

In 17/37 (45.9%) of the refactoring-inducing PRs, the reviewers provide reasons that induced refactoring edits; nevertheless, the refactoring type is not explicitly pointed, except for a few *Rename*, *Move*, and *Extract* instances. In addition, even suggestions of low-level refactorings may provide reasons. Indeed, only review comments in flink #7945 (sample 1) and cloudstack #2071, incubator-iceberg #119, incubator-iceberg #183 (sample 3) submit reasons for high-level refactoring instances.

Reasons are **contextualized sentences** regarding problematic situations, structured like alerts (e.g., obfuscated name in fluo #837, sample 1, and constants format in brooklyn-server #1049, sample 2), sometimes accompanied by **explanations using examples** (e.g., code's structure under analysis in flink #7970 and kafka #5194, sample 1), as it occurs in cloudstack #2071, cloudstack #3454, flink #8222, incubator-iceberg #119, incubator-iceberg #183, kafka #4757, kafka #5590 (sample 3). In other cases, reasons begin with a **question on code logic** (*"Is this type over-kill?"*, *"Can we…?"*) assisted by examples (dubbo #3299, flink #7945 in sample 1; kafka #5784, tomee #275 in sample 2); in others, reviewers provide a **direct explanation** as in dubbo #2279 (informing about a class useless).

- **RQ_5 What is the relationship between refactoring recommendations and actual refactorings in refactoring-inducing PRs?**

When the reviewers get straight to the point, refactoring edits are always done. We identify such a pattern in 28/28 (100%) of refactoring-inducing PRs, in which refactoring edits were induced by

code review. For instance, in kafka #5194, a reviewer says *"The interface belongs in the package of the providers. Should be org.apache.kafka.common.config.provider.ConfigProvider."*, so inducing a Move Class instance. Other direct review comments include explanations using examples in the code under analysis (fluo #837, flink #7970) and reasons (commons-text #39, flink #7945, incubator-iceberg #254 in sample 1; kafka #5784 in sample 2; cloustack #2071, cloudstack #3454, flink #8222, incubator-iceberg #119, incubator-iceberg #183, kafka #4757 in sample 3).

Nevertheless, when there is space for discussion, refactorings may not be done. We observed **discussions** due to uncertain review comments in five refactoring-inducing PRs, in which refactoring edits were induced by code review (flink #9143 in sample 1; beam #4458, kafka #7132 in sample 2; kafka #4735, kafka #5590 in sample 3) and in one refactoring-inducing PR, in which refactoring edits were led by the author (kafka #6657 in sample 3). Uncertain review comments, as cited, include expressions like *"wondering whether..."*, *"there were discussions to..."*, *"I am just wondering if..."*, *"I'm not sure..."*, and *"As far as I remember..."*. To reinforce such a pattern, as we previously argued, the proportion of reviewing discussion is higher in non-refactoring-inducing PRs than in refactoring-inducing PRs. For instance, in dubbo #4870 (sample 1), a reviewer suggests a method change (*"should we..."*) by indicating an inline code, nonetheless, PR's author presents an argument and a counterexample, so winning the discussion.

Suggestions

As a proposal for the next sample, we suggest an exploration of distinct sequences of refactoring edits (of different types of refactoring) in PR commits.

Notes

- **Apache roles[2]:**
  - **Contributor** is a developer who contributes to a project in the form of code or documentation.
  - **Committer** is a contributor who has write access to the code repository.
  - **PMC member** is a committer who has write access to the code repository and can agree and approve/disapprove the changes.

- **Repositories timeline (contribution activities):**

  - flink – Dec 2010
  - cloudstack – Aug 2010
  - usergrid – Oct 2011
  - dubbo – Jun 2012
  - kafka – Dec 2012

2 https://apache.org/foundation/how-it-works.html#roles

- fluo – Jun 2013
- beam – Dec 2014
- incubator-iotdb – May 2017
- servicecomb-java-chassis – May 2017
- incubator-iceberg – Dec 2017

Refactoring-inducing PRs:

- **beam 6261**: 2018 | commit merge | 1 mention after PR merge | no bot | adaptive changes (adding a new test) | firstly, a contributor suggested running a test suit | **refactorings led by PR's author** | 1 self-affirmed refactoring | review comments on adding documentation regarding a new class (after self-affirmed refactoring), on a method call (answered by the author), and questioning concerning adding new tests | 14 (+1) refactoring edits | presence of floss refactoring
  - 2º subsequent commit: *2 Change Attribute Type* | *2 Extract Class* | *3 Move Method* | *5 Move Attribute* | *(false negative: 1 Move Attribute)* | *(Move instances are due to Extract instances)*
  - 5º subsequent commit: *1 Extract Method* | *1 Extract and Move Method*
  - 14 subsequent commits, 33 file changes, 413 added lines, 285 deleted lines, 6 days to merge
  - Author, contributions since 2018 (97)
  - Reviewer, contributions since 2012 (1125), reviews since 2016

- **cloudstack 2071**: 2017 | commit merge | initial refactoring edits | no bot | perfective changes (enhancing a service) | **refactorings led by code review** | a direct review comment questioning a package name, providing a reason | a direct review comment suggesting a package rename ("What about using...")? caused discussion (reviewer won the discussion by providing a rationale) | refactored code in the last commit led to needs for clarification | 5 refactoring edits
  - 1º subsequent commit: *1 Rename Package* | *2 Move Class* | *(Move instances due to Rename Package instance)*
  - 2º subsequent commit: *2 Move Class*
  - 2 subsequent commits, 16 file changes, 16 added lines, 16 deleted lines, 4 days to merge
  - Author, contributions since 2012 (5,254)
  - Reviewer, contributions since 2009 (15,197), reviews since 2016
  - Reviewer, contributions since 2008 (3,414), reviews since 2016
  - Reviewer, contributions since 2013 (319), reviews since 2017
  - Reviewer (Apache Cloudstack PMC), contributions since 2015 (300), reviews since 2016

- **cloudstack 3454**: 2019 | commit merge | initial refactoring edits | no bot | perfective change (enhancing a service) | **refactorings induced by code review** | a direct review comment (induced refactorings) suggested moving changes to another class, providing security-related reasons | review comments (no induced refactorings) addressed UI layouts | 5 refactoring edits

    - 1º subsequent commit: *1 Split Parameter* | *2 Push Down Attribute* | *2 Push Down Method*
    - 3 subsequent commits, 7 file changes, 47 added lines, 38 deleted lines, 11 days to merge
    - Author, contributions since 2018 (99)
    - Reviewer (Apache Cloudstack PMC), contributions since 2013 (77), reviews since 2017
    - Reviewer (Apache Cloudstack PMC), contributions since 2009 (18,995), reviews since 2016
    - Reviewer (Apache Cloudstack PMC), contributions since 2015 (935), reviews since 2016
    - Reviewer, contributions since 2009 (347), reviews since 2019

- **dubbo 3654**: 2019 | commit merge | initial refactoring edits | 1 mention after PR merge | no bot | perfective (optimization) and corrective (fixing faults) changes | **refactoring led by the author** | review comments on a method call, a simple code logic (providing a reason), and need for a new test | 1 refactoring edit | presence of floss refactoring

    - 2º subsequent commit: *1 Move Class*
    - 3 subsequent commits, 6 file changes, 97 added lines, 17 deleted lines, 5 days to merge
    - Author, contributions since 2014 (116)
    - Reviewer, contributions since 2016 (187), reviews since 2018
    - Reviewer (Apache Dubbo PMC), contributions since 2013 (751), reviews since 2018

- **flink 8222**: 2019 | commit merge | initial refactoring edits | 1 external reference after PR merge | flinkbot | adaptive changes (adding a new feature) | **refactorings induced by code review and the author** | review comment (induced refactoring) questioned the use of list ("Should we use …?"), providing reason on the preservation of order | review comments (inducing refactoring) questioned the use of list, providing no reason | review comments in form of doubts were solved by the author | a review comment on the potential need for cloning a class induced *Pull Up Attribute* and *Pull Up Method* instances | review comments on adding test, adding comments | a lot of *Change Type* instances are due to a review comment addressing issues on class name consistency | a review comment concerning no need for an interface (providing reason) caused all inheritance-related refactorings | 55 refactoring edits

    - 1º subsequent commit: *1 Change Return Type* | *1 Change Attribute Type* | *2 Change Parameter Type* (*Change Parameter Type* instances and a *Change Return Type* are due to *Change Attribute Type* instance)
    - 2º subsequent commit: *1 Extract Variable* | *2 Change Return Type*

- 3º subsequent commit: *1 Rename Class* | *7 Change Return Type* | *4 Extract Method* | *6 Rename Method* | *2 Rename Parameter* | *5 Extract Variable* | *2 Change Variable Type* | *1 Merge Parameter* | *2 Change Attribute Type* | *12 Change Parameter Type* | *1 Pull Up Attribute* | *5 Pull Up Method* (*Change Type* instances are due to class name consistency)
- 5 subsequent commits, 29 file changes, 605 added lines, 391 deleted lines, 6 days to merge
- Author, contributions since 2012 (369)
- Reviewer, contributions since 2009 (1,814), reviews since 2016
- Reviewer (Apache Flink and Beam Committer), contributions since 2014 (201), reviews since 2017
- Reviewer (Apache Flink Committer and PMC), contributions since 2013 (889), reviews since 2016
- Reviewer, contributions since 2011 (83), reviews since 2018
- Reviewer (Apache Flink PMC), contributions since 2011 (284), reviews since 2016

- **incubator-iceberg 119**: 2019 | commit merge | 2 external references after PR merge | no bot | adaptive changes (adding new tests) | This PR addresses all comments from PR #111 | **refactorings induced by code review and the author** | review comment, in form of warning, on code logic (e.g., suggesting a better parameter value) | review comments on code aesthetics and need for documentation | review comment suggested moving test classes, providing a reason (to avoid duplication) | review comment on using of a data random generator caused discussion and no changes | review comment suggested the use of a set of values (induced refactoring), providing a reason | a direct review comment caused a method restructuration, in form of a warning (good development practice) | 11 refactoring edits

  - 3º subsequent commit: *1 Move and Rename Class (by code review)* | *2 Rename Attribute (by the author)* | *1 Change Attribute Type (by the author)*
  - 4º subsequent commit: *1 Extract Method(by code review)*
  - 5º subsequent commit: *1 Change Parameter Type (by the author)* | *1 Extract Method (by the author)*
  - 6º subsequent commit: *1 Change Return Type (by code review)* | *1 Rename Parameter (by code review)* | *1 Change Parameter Type (by code review)* | *1 Rename Method (by code review)*
  - 6 subsequent commits, 18 file changes, 294 added lines, 330 deleted lines, 11 days to merge
  - Author, contributions since 2009 (614)
  - Reviewer, contributions since 2009 (4,010), reviews since 2016

- **incubator-iceberg 183**: 2019 | commit merge | 1 external reference after PR merge | no bot | corrective changes (fixing faults) | **refactorings induced by code review and the author** | review comments, in form of doubts ("Looks like...", "not sure if..."), caused no change | a review comment, in form of a warning regarding good practices of programming, concerning tests (providing reason), induced inheritance-related refactorings | 7 refactoring edits

  - 1º subsequent commit: *1 Extract Superclass (by code review)* | *3 Pull Up Method (by code review)* | *2 Pull Up Attribute (by code review)* | *1 Change Parameter Type (by the author)*

- 2 subsequent commits, 5 file changes, 143 added lines, 92 deleted lines, 1 day to merge
- Author, contributions since 2012 (86)
- Reviewer, contributions since 2009 (4,085), reviews since 2016
- Reviewer, contributions since 2013 (113), reviews since 2016

- **kafka 4735**: 2018 | commit merge | 2 external after PR merge | no bot | adaptive changes (adding documentation) | **refactorings induced by code review and the author** | review comments on typos and better output sentences | a direct review comment suggesting inheritance-related refactoring | self-affirmed minor review comments | 17 refactoring edits

  - 1º subsequent commit: *1 Extract Variable (by the author)*
  - 3º subsequent commit: *13 Change Attribute Type (by the author)* | *2 Change Variable Type (by the author)* | *1 Extract Superclass (by code review) (Change Type instances are due to Extract Superclass edit)*
  - 5 subsequent commits, 8 file changes, 96 added lines, 104 deleted lines, 29 days to merge
  - Author, contributions since 2015 (7)
  - Reviewer, contributions since 2013 (59), reviews since 2017
  - Reviewer, contributions since 2010 (4,910), reviews since 2016
  - Reviewer, contributions since 2015 (1,737), reviews since 2016

- **kafka 4757**: 2018 | commit merge | 1 external reference after PR merge | no bot | adaptive changes (adding a new feature) | an extensive discussion, assisted by rationales and examples using a few code structures, regarding how to deal with common configuration keys for clients caused refactoring | **refactorings induced by code review** | an extensive discussion, assisted by rationales and examples using a few code structures, regarding how to deal with common configuration keys for clients caused refactoring | a direct review comment on adding a config to ease later changes, providing a reason | a direct review comment, suggesting adding a utility function, inspired *Extract* and *Extract and Move* instances | self-affirmed minor review comments | 10 refactoring edits | presence of floss refactoring

  - 3º subsequent commit: *1 Extract Method*
  - 4º subsequent commit: *2 Rename Parameter*
  - 5º subsequent commit: *1 Extrac Method*
  - 7º subsequent commit: *1 Extract Method* | *3 Extract and Move Method*
  - 8º subsequent commit: *2 Push Up Method*
  - 9 subsequent commits, 29 file changes, 207 added lines, 94 deleted lines, 15 days to merge
  - Author, contributions since 2013 (80)
  - Reviewer, contributions since 2007 (14,216), reviews since 2016
  - Reviewer, contributions since 2011 (2,009), reviews since 2016
  - Reviewer (Apache Kafka PMC), contributions since 2015 (853), reviews since 2016

- **kafka 5590**: 2018 | commit merge | 3 external references after PR merge | no bot | adaptive changes (adding a new feature) | self-affirmed minor PR | **refactoring induced by code review** | an extensive discussion regarding the PR's purpose, involving the author and four reviewers, provided rationales and a few examples that emphasize the usefulness of such a PR | the refactoring-inducement happened due to a reviewer's explanation on need for an interface | minor review comments on code logic | 2 refactoring edits

    - 1º subsequent commit: *1 Extract Interface*
    - 4 subsequent commits, 7 file changes, 81 added lines, 222 deleted lines, 1 day to merge
    - Author, contributions since 2011 (1,719)
    - Reviewer, contributions since 2012 (50), reviews since 2017
    - Reviewer (Apache Kafka Committer and PMC), contributions since 2015 (2,531), reviews since 2016
    - Reviewer (Apache Kafka PMC), contributions since 2010 (1,151), reviews since 2016

- **kafka 6657**: 2019 | commit merge | 1 external reference after PR merge | no bot | corrective (fixing faults) | **refactoring led by the author** | minor review comment on the use of a specific object and timestamp range (causing discussion) | self-affirmed minor review comments | 1 refactoring edit

    - 2º subsequent commit: *1 Pull Up Attribute*
    - 4 subsequent commits, 7 file changes, 80 added lines, 49 deleted lines, 3 days to merge
    - Author, contributions since 2013 (44)
    - Reviewer (Apache Kafka Committer and PMC), contributions since 2015 (3,041), reviews since 2016
    - Reviewer (Apache Kafka PMC), contributions since 2010 (1,577), reviews since 2016

- **servicecomb-java-chassis 678**: 2018 | commit merge | initial refactoring edit | no bot | An empty checklist for contributions | No PR description | adaptive changes (adding a new feature) | **refactorings induced by code review and led by the author** | review comments addressed good practices of programming (warnings), typos, and questions on code logic (e.g., concurrency issues – inducing refactoring), in which arguments and code example were used as support | 27 refactoring edits

    - 1º subsequent commit: *1 Change Attribute Type (by code review)*
    - 3º subsequent commit: *2 Replace Variable with Attribute* | *1 Change Variable Type* | *1 Extract Class* | *3 Move Attribute* | *6 Push Down Method*
    - 4º subsequent commit: *1 Rename Attribute*
    - 5º subsequent commit: *1 Rename Method*
    - 6º subsequent commit: *2 Rename Attribute* | *2 Change Attribute Type*
    - 7º subsequent commit: *2 Change Variable Type* | *2 Rename Variable*

- 8º subsequent commit: *1 Rename Variable* | *1 Rename Parameter* | *1 Change Attribute Type*
- 9 subsequent commits, 47 file changes, 586 added lines, 369 deleted lines, 54 days to merge
- Author, contributions since 2017 (16)
- Reviewer, contributions since 2016 (741), reviews since 2017
- Reviewer (Apache Servicecomb PMC), contributions since 2017 (445), reviews since 2017

- **usergrid 102**: 2014 | commit merge | init refactoring edit | no bot | No PR description | self-affirmed refactorings in five commits | corrective changes (fixing fault) | **refactorings led by the author** | a minor review comment concening the content of a file | 54 refactoring edits

  - 1º subsequent commit: *1 Move Class* | *1 Extract Class* | *2 Push Down Attribute* | *2 Move Attribute*
  - 2º subsequent commit: *1 Extract Class*
  - 5º subsequent commit: *8 Rename Method*
  - 6º subsequent commit: *4 Change Attribute Type* | *4 Rename Method* | *1 Rename Attribute* | *6 Push Down Attribute* | *1 Pull Up Method*
  - 7º subsequent commit: *1 Change Return Type* | *1 Move Attribute*
  - 8º subsequent commit: *10 Extract and Move Method*
  - 10º subsequent commit: *1 Move Class* | *1 Rename Attribute* | *1 Inline Variable* | *1 Extract Superclass* | *4 Pull Up Method* | *1 Push Down Method* | *2 Move Method*
  - 10 subsequent commits, 133 file changes, 4485 added lines, 1866 deleted lines, 3 days to merge
  - Author, contributions since 2010 (904)
  - Reviewer, contributions since 2013 (154), reviews since 2014

Non-Refactoring-Inducing PRs:

- **dubbo 3447**: 2019 | commit merge | no bot | adaptive changes (adding a new test) | direct review comments regarding code logic (e.g. need for reverting a deleted line) and use of assertion

  - 2 subsequent commits, 2 file changes, 21 added lines, 19 deleted lines, 0 days to merge
  - Author, contributions since 2014 (112)
  - Reviewer, contributions since 2016 (815), reviews since 2019
  - Reviewer, contributions since 2015 (1,333), reviews since 2018
  - Reviewer (Apache Dubbo Committer), contributions since 2015 (177), reviews since 2018
  - Reviewer (Apache Dubbo PMC), contributions since 2017 (445), reviews since 2018

- **cloudstack 2706**: 2018 | commit merge | no bot | corrective changes (fixing faults) | no review comments on Java files

  - 5 subsequent commits, 6 file changes, 10 added lines, 8 deleted lines, 8 days to merge

- Author, contributions since 2009 (16,802)
- Reviewer, contributions since 2013 (720), reviews since 2017
- Reviewer, contributions since 2012 (5,841), reviews since 2017
- Reviewer, contributions since 2010 (1,602), reviews since 2016

- **cloudstack 3276**: 2019 | commit merge | no bot | corrective changes (fixing faults) | review comments on adding a test, on code/business logic (providing reasons), and questioning to revert a change (answered by the author)

  - 8 subsequent commits, 18 file changes, 167 added lines, 116 deleted lines, 25 days to merge
  - Author, contributions since 2018 (18)
  - Reviewer (Apache Cloudstack Committer and PMC), contributions since 2009 (18,728), reviews since 2016
  - Reviewer (Apache Cloudstack PMC), contributions since 2015 (581), reviews since 2016

- **dubbo 3184**: 2019 | commit merge | initial refactoring edit | no bot | perfective changes (enhancement) | review comments on changing method calls due to their impact on business logic, in form of suggestions and providing reasons | review comment questioned code logic (answered by the author)

  - 3 subsequent commits, 5 file changes, 5 added lines, 18 deleted lines, 1 day to merge
  - Author, contributions since 2016 (1,107)
  - Reviewer, contributions since 2018 (85), reviews since 2018
  - Reviewer, contributions since 2016 (828), reviews since 2017

- **dubbo 4208**: 2019 | commit merge | 1 mention after PR merge | no bot | adaptive changes (adding a new test) | minor review comments on code aesthetics

  - 3 subsequent commits, 9 file changes, 123 added lines, 8 deleted lines, 2 days to merge
  - Author, contributions since 2014 (1,318)
  - Reviewer, contributions since 2014 (286), reviews since 2017
  - Reviewer (Apache Dubbo PMC), contributions since 2011 (846), reviews since 2017

- **kafka 6298**: 2019 | commit merge | 1 external reference after PR merge | no bot | corrective changes (fixing faults) | no review comments on Java files

  - 5 subsequent commits, 10 file changes, 69 added lines, 25 deleted lines, 1 day to merge
  - Author, contributions since 2015 (7,569)
  - Reviewer, contributions since 2015 (1,698), reviews since 2016

- **servicecomb-java-chassis 744**: 2018 | commit merge | no bot | no PR description, a fullfilled checklist for contributions | adaptive changes (adding a new feature) | review comments suggested code simplification and questioned decisions (e.g., why using a specific object)

    - 2 subsequent commits, 7 file changes, 59 added lines, 119 deleted lines, 4 days to merge
    - Author, contributions since 2018 (136)
    - Reviewer (Apache Servicecomb PMC), contributions since 2017 (508), reviews since 2017
    - Reviewer (Apache Servicecomb PMC), contributions since 2016 (828), reviews since 2017

- **incubator-iotdb 67**: 2019 | commit merge | initial refactoring edits | no bot | adaptive changes (adding documentation) | minor review comments suggested improvements to a few output sentences and questioned parameter values, providing an image (answered by the author)

    - 4 subsequent commits, 6 file changes, 8 added lines, 6 deleted lines, 2 days to merge
    - Author, contributions since 2016 (625)
    - Reviewer (Apache Iotdb PMC), contributions since 2016 (530), reviews since 2017
    - Reviewer (Apache Iotdb PMC), contributions since 2011 (285), reviews since 2017
    - Reviewer (Apache Iotdb PMC), contributions since 2014 (2,888), reviews since 2017

- **cloudstack 2553**: 2018 | commit merge | no bot | perfective changes (updating/enhancement) | minor review comments suggested using a different parameter for exception errors (giving an example of a method call) | discussion on how to handle log messages

    - 6 subsequent cohttp://twitter.com/homemmits, 16 file changes, 22 added lines, 24 deleted lines, 8 days to merge
    - Author, contributions since 2015 (109)
    - Reviewer (Apache Cloudstack PMC), contributions since 2013 (577), reviews since 2017
    - Reviewer (Apache Cloudstack PMC), contributions since 2012 (5,808), reviews since 2017

- **fluo 929**: 2017 | commit merge | no bot | no PR description | corrective changes (fixing faults) | review comments suggested adding a test and changing conditions and error messages in the code | reviewer provided a code example to answer one author's question and reasons to change and detailed instructors to improve the code

    - 5 subsequent commits, 6 file changes, 53 added lines, 18 deleted lines, 1 day to merge
    - Author, contributions since 2011 (18)
    - Reviewer, contributions since 2011 (3,330), reviews since 2016

- **servicecomb-java-chassis 691**: 2018 | commit merge | no bot | perfective/refactoring changes (self-affirmed PR) | minor review comments questioned the use of an exception class and code

logic | review comments suggested code simplification, adding documentation, and changing one exception output sentence, besides improvements (providing reason), e.g., to avoid conflicts

- 1 subsequent commit, 7 file changes, 23 added lines, 15 deleted lines, 4 days to merge
- Author, contributions since 2012 (97)
- Reviewer, contributions since 2016 (828), reviews since 2017
- Reviewer, contributions since 2017 (508), reviews since 2017

- **cloudstack 2714**: 2018 | commit merge | no bot | corrective (fixing faults) and perfective (enhancement) changes | review comments questioned the adding of tests and a method overriding | discussion regarding what exceptions should be considered in tests

  - 3 subsequent commits, 4 file changes, 35 added lines, 7 deleted lines, 1 day to merge
  - Author, contributions since 2012 (6,392)
  - Reviewer (Apache Cloudstack PMC), contributions since 2013 (522), reviews since 2017
  - Reviewer (Apache Cloudstack PMC), contributions since 2013 (341), reviews since 2016
  - Reviewer (Apache Cloudstack PMC), contributions since 2009 (18,866), reviews since 2016
  - Reviewer (Apache Cloudstack PMC), contributions since 2015 (921), reviews since 2016

- **kafka 6438**: 2019 | commit merge | 1 external reference | no bot | perfective changes (improvement) | PR description declares minor code style improvements | self-affirmed minor PR | author provided clarifications on his decisions | the most review comments comprises non-Java files (only one on a Java file)

  - 3 subsequent commits, 4 file changes, 19 added lines, 6 deleted lines, 85 days to merge
  - Author, contributions since 2015 (7,592)
  - Reviewer, contributions since 2011 (2,280), reviews since 2016
  - Reviewer, contributions since 2011 (832), reviews since 2018

References

Fowler, M. *Refactoring: Improving the Design of Existing Code*. Boston, MA, USA: Addison-Wesley. ISBN: 0-201-48567-2, 2000.

Miles, M. B. and A. M. Huberman. *Qualitative Data Analysis: An Expanded Sourcebook.* Second Edition. Singapore: Sage Publications, 1994.

Mockus, A. and Votta, L. G. "Identifying Reasons for Software Changes Using Historic Databases". In *Proceedings of the International Conference on Software Maintenance.* Washington, USA: IEEE Computer Society, 2000.

Patton, M. Q. *Qualitative Research & Evaluation Methods: Integrating Theory and Practice.* Fourth Edition. Singapore: Sage Publications. ISBN: 9781412972123, 2014.

Swanson, E. B. “The Dimensions of Maintenance”. In *Proceedings of the 2nd International Conference on Software Engineering*, 492–497. Washington, USA: IEEE Computer Society, 1976.