Rationale Dataset and Analysis for the Commit Messages of the Linux Kernel Out-of-Memory Killer

Mouna Dhaouadi  
DIRO, Université de Montréal  
Montréal, Canada  
mouna.dhaouadi@umontreal.ca

Bentley James Oakes  
GIGL, Polytechnique Montréal  
Montréal, Canada  
bentley.oakes@polymtl.ca

Michalis Famelis  
DIRO, Université de Montréal  
Montréal, Canada  
michalis.famelis@umontreal.ca

ABSTRACT

Code commit messages can contain useful information on why a developer has made a change. However, the presence and structure of rationale in real-world code commit messages is not well studied. Here, we detail the creation of a labelled dataset to analyze the code commit messages of the Linux Kernel Out-Of-Memory Killer component. We study aspects of rationale information, such as presence, temporal evolution, and structure. We find that 98.9% of commits in our dataset contain sentences with rationale information, and that experienced developers report rationale in about 60% of the sentences in their commits. We report on the challenges we faced and provide examples for our labelling.

CCS CONCEPTS

• Software and its engineering → Requirements analysis; Maintaining software; Documentation.

KEYWORDS

developer rationale, dataset, Linux kernel, commit messages

1 INTRODUCTION

To be effective in evolving software systems, developers who desire to propose code changes have to deeply understand the system and its behavior. Developing this understanding is not easy, yet it is crucial to grasp the rationale behind the decisions that shaped the system to its current state. Communities of software developers may put in place norms to encourage documenting this rationale, to describe why each change was made. In modern software development, such information is often contained in the commit message that accompanies a proposed code change that is submitted in a shared version control repository, such as Git.

Although several researchers have discussed the importance of rationale [7], and proposed various approaches for structuring and extracting it [5, 10, 11, 16, 20, 26, 35], there are scant details in the literature about the characteristics of rationale in real-world systems. In fact, only few researchers have attempted to develop a deep understanding of developers’ rationale in Open Source Software (OSS), e.g., by studying chat messages or email archives [2, 27]. To the best of our knowledge, there is no prior work studying developer’s rationale in the code commit messages of open source projects, specifically their structure. We expect that the results of such studies could be used a) to better understand the characteristics of rationale (such as its presence, structure and evolution over a project’s lifetime), b) as first steps towards setting rationale documentation guidelines to improve commit messages to justify the commits for OSS commits in the future, which would reduce the developer’s need to access an issue tracker (avoiding inconsistencies or traceability errors), and c) as baselines for automated solutions (e.g. GitHub pull request bots) to identify rationale in software.

To fill this gap, we propose to study developers’ rationale in the commit messages of an open source project. Our main research questions is: What are the characteristics (presence, impact factors, evolution, structure) of how rationale information appears in collaborative open-source commit messages? Specifically, we report on our creation of an annotated, high-quality rationale dataset for the Out-of-Memory Killer (OOM-Killer) component of the Linux kernel project. We have previously argued that this project is well suited for studying rationale [10]. In this work, we systematically annotate the content of these commit messages, using a categorization of sentences as Decision, Rationale and Supporting Facts [11]. We then analyze quantitatively the resulting dataset to characterize rationale in the OOM-Killer subsystem. Specifically, we follow the empirical standards of a repository mining study [24].

We previously proposed Kantara [10], an end-to-end automatic rationale extraction and management pipeline, and evaluated it on some example commits from the Linux OOM-Killer component. Our then-partial labelling of the OOM-Killer dataset is completed here, as a stepping stone to future work on automatic extraction [9]. We also described how this rationale information can be represented using an ontologically-based knowledge graph, along with some initial reporting and visualization functionalities of Kantara [11].

The contributions of this paper are twofold: 1) a high-quality dataset of labelled commits of the OOM-Killer component, and 2) an analysis of the dataset of seven research questions (see Section 4.2) that touch on topics ranging from the frequency of the presence of rationale, the factors that impact it, its temporal evolution, to
the structure of commit messages. The dataset and the source code used to perform our analysis are publicly available.1

The remainder of this paper is organized as follows. We present the OOM-Killer subsystem in Section 2, and discuss our work on creating the labelled dataset of the OOM commits in Section 3. Then, we describe the obtained dataset and analyze it in Section 4. We introduce the threats to validity we encountered in Section 5, overview related work in Section 6, and conclude in Section 7.

2 LINUX OUT-OF-MEMORY KILLER SUBSYSTEM

The Linux kernel is a large open-source project that is being continuously developed collaboratively since 1991. Its main collaboration channel is the Linux Kernel Mailing List (LKML), which contains code patches and discussions. Since 2005, code patches for the Linux kernel have been structured as Git commits. Table 2 shows an example of a commit message. Developers are encouraged to explain the motivation for the commit and explain its impact on the kernel in the commit message.2 This community practice makes the Linux kernel commits a rich repository of rationale information.10

The Out-Of-Memory Killer (OOM-Killer) subsystem is a Linux module that is in charge of freeing up the memory to prevent a system crash when all the available memory has already been allocated. It embodies one approach to handling OOM problems.15 It refers to the system’s future state.15

Thus, we have applied purposeful sampling.30 We have selected a component (the OOM-Killer) with high-quality justification in commit messages as a critical case, i.e., a case where rationale of high quality is expected to be present and identifiable. Absence of rationale in this case would signify deeper problems for rationale identification in general.

3 DATASET CREATION

This section discusses labelling the rationale information of all commits of the OOM subsystem, as well as the final data set structure.

3.1 Commit Pre-processing

We obtained the initial set of commits downloading the available commit history for the oom_kill.c file which contains the C source code of the OOM Killer module.3 This initial set contained 418 commits since the Linux development moved to Git on 2005-04-16 up until 2022-09-27 (the commit date of the latest commit we pulled). We excluded Git merge commits, as they do not typically contain informative commit messages. In fact, in this particular module, we have 13 merge commits that restate previous commits (e.g., “Pull updates from X”4), contain no description of the changes (e.g., inapplicable pre-processing error or bad sentences (i.e., does not contain English sentences)5 or summarize the changes in a non-informative manner (e.g., “updates”8). Therefore, removing them does not present a threat to validity. Only one merge commit contains informative description as it fixes some conflicts.7 Although removing this specific commit presents a small threat, we chose to do so for consistency reasons.

We then performed pre-processing of the messages of the remaining 404 non-merge commits. For each message, we removed the meta-data at the end of the message, such as the tags Signed-off-by and Suggested-by, that are not relevant for the study of rationale. We also removed URLs, references to other resources, and call traces using regular expressions. We then split the message to sentences, keeping only sentences with: a) more than three characters, and b) that are not source code. We used heuristics to detect whether a sentence is source code, like the existence of keywords or symbols such as git, sed or echo. We chose these keywords by manually investigating the data.

Since the data was not properly formatted, the pre-processing step was challenging and only partially successful. For instance, entries such as ”BUG: scheduling while atomic: rsyslogd/1422/0x00000000 INFO: lockdep is turned off” were not removed. To overcome this limitation, we will continue cleaning the data during the labelling, using the Inapplicable label (see Section 3.2.2). The pre-processed sentences could be also be corrected manually but that would be time consuming and error-prone.

3.2 Sentence Labelling Procedure

3.2.1 Piloting. We continue the categorization of Decision, Rationale and Supporting Facts from [11], as shown in Table 1. We note that a sentence can have multiple labels, or no labels. To develop a shared understanding regarding the meaning of the labels we will be using, we performed five iterations of piloting rounds and consolidation meetings during which the three annotators independently annotated 33 randomly-chosen commits in total. We elaborated a common protocol to consistently label the dataset based on the diverse cases we encountered during our piloting rounds.

3.2.2 Codebook. As shown in Table 1, a Decision provides information about the state of the system after the patch is applied, i.e., it refers to the system’s future state. Rationale is the reason why a decision is taken, such as a value judgement about undesirable behavior. Supporting Facts are bits of information in a sentence where a developer discusses the currently existing state of the system, at

Table 1: Labelling codebook

<table>
<thead>
<tr>
<th>Label</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decision</td>
<td>An action or a change that has been made, including a description of the patch behaviour</td>
</tr>
<tr>
<td>Rationale</td>
<td>Reason for a decision or value judgment</td>
</tr>
<tr>
<td>Supporting Facts</td>
<td>A narration of facts used to support a decision</td>
</tr>
<tr>
<td>Inapplicable</td>
<td>Pre-processing error or bad sentences (i.e., does not contain English sentences)</td>
</tr>
</tbody>
</table>

---

1 https://enode.org/record/10063089
3 https://github.com/torvalds/linux/commit/132b7931ad56f1863b55540c0814d76b
4 https://github.com/torvalds/linux/commit/5c6aebec9ac2e4347189759f6dfb60cad5c4e3c9b6
5 https://github.com/torvalds/linux/commit/35ce8ae9ae2e471f92759f9d6880eab42cc1c3b6
7 https://api.github.com/repos/torvalds/linux/git/commits/b52723c5607f7684c2c0c075f86f86da0d7fb6d0
8 https://github.com/torvalds/linux/commit/2b828925652340277a889cbc11b2d0637f7cdaf7
9 https://api.github.com/repos/torvalds/linux/git/commits/28e92f990337b8b4c5fdec47667f8b96089c503e
the moment before they propose a change. The Inapplicable label is used by the annotators to identify noise in the data that was not successfully filtered by the pre-processing step due to the lack of uniform formatting and the presence of (pseudo-)code.

Table 2 reproduces a commit from the dataset, along with a classification for each sentence. The first sentence (the summary phrase of the commit) is labelled as a Decision as it states the patch’s change. The second sentence is labelled as a Supporting Facts as it presents the currently existing state of the system, and the third sentence is labelled as Rationale as it motivates this commit.

The codebook we adopted is data-driven; we came to it after several discussions and consolidation meetings. However, this categorization is preliminary and used as a first-order classification. For example, although we labelled the sentence “A future optimization could be to put sched_entity and sched_rt_entity into a union.” as Decision, it could also fit under a Suggestion category. We plan to extend this codebook in the future to include other components of commit message rationale (e.g. Goal, Need, Benefit, etc. [1]).

3.2.3 Protocol. To systematically create the dataset, we developed this protocol for annotating the commit message sentences:

(1) We do not separate a sentence from its context; when in doubt, we look at the patch code to better understand it.

(2) Past changes count as Supporting Facts. Decision and Rationale, however, usually concern the present (current commit), e.g., the sentence “The reason this check was added was the thought that, since only the OOM disabling code would wait on this queue, wakeup operations could be saved when that specific consumer is known to be absent.” counts as Supporting Facts. In contrary, the sentence “It’s better to extract this to a helper function to remove all the confusion as to its semantics.” counts as Decision and Rationale.

(3) Future intent counts as rationale. e.g., the sentence “Moreover, this will make later patch in the series easier to review.” counts as Rationale.

(4) We consider terse value judgment language (e.g., “fix” or “cleanup”) to imply the presence of rationale and descriptions of decisions, even if it is low quality. e.g., the sentences “mm/oom_kill.c: fix vm_oom_kill_table[] idfjeffery.” and “Unify it.” count both as Decision and Rationale.

(5) It is possible that no label is applicable. e.g., the sentence “This patch introduces printk ratelimiting specifically for the oom killer.” (6) Examples can also be labelled with one of the categories. e.g., the sentence “In the following example, abuse_the_ram is the name of a program that attempts to iteratively allocate full oom memory until it is stopped by force.” counts as Supporting Facts. However, the sentence “The value is added directly into the badness() score so a value of -500, for example, means to discount 50% of its memory consumption in comparison to other tasks either on the system, bound to the mempolicy, in the cpuset, or sharing the same memory controller.” should be labelled the same even though pre-processing produced them as separate items.

(7) If a sentence was mistakenly cut during pre-processing, we label all the parts with the same labels, e.g., these two parts of the same sentence: “oom_reaper used to rely on the oom_lock since e2fe1456433 (“oom_reaper”) and “close race with exit-
task”).” should be labelled the same even though pre-processing produced them as separate items.

(8) If pre-processing has produced an item that mixes code (e.g., log or trace or source code) with a valid English sentence (e.g., not heading, for example: “current message”), we ignore the code and label the item based on the sentence. e.g., when labelling the item “oom-kill: constraint=CONSTRAINT_NONE, nodemask=null, cpuset=1, mems_allowed=0-1, task=panic, pid=10737, uid=0 An admin can easily get the full oom context at a single line which makes parsing much easier.”, we disregard the non-English part and apply a label based only on the part starting with “An admin can easily...”.

3.2.4 Labelling. We conducted the labelling of the 366 remaining commits (i.e., excluding those used for the piloting) in batches, over a period of six months. In all, we annotated 2333 sentences, where sentences could have multiple labels. While labelling independently, we held regular meetings to discuss problematic sentences and resolve discrepancies. The purpose of the meetings was to agree on how to remove any potential misunderstanding or confusion regarding the commit messages. For sentences where we could not establish complete agreement or that contained ambiguous language, we assigned labels by taking the union of all opinions.

To compute inter-rater agreement, we use Fleiss Kappa [12] since we are in the presence of more than two annotators. Table 3 shows the number of batches, their sizes (number of commits) and the Fleiss Kappa per category. Fleiss Kappa was computed after the consolidation meetings. As shown in Table 3, there is generally a stronger consensus about the Decision and Supporting Facts categories than the Rationale category. Despite the low kappa values for individual categories in some batches (e.g. batch 6, 9 or 10), overall...
<table>
<thead>
<tr>
<th>Batch</th>
<th>Size</th>
<th>Decision</th>
<th>Rationale</th>
<th>Supporting Facts</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>20</td>
<td>0.596</td>
<td>0.514</td>
<td>0.489</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td>0.733</td>
<td>0.487</td>
<td>0.596</td>
</tr>
<tr>
<td>3</td>
<td>20</td>
<td>1.0</td>
<td>0.933</td>
<td>1.0</td>
</tr>
<tr>
<td>4</td>
<td>40</td>
<td>0.899</td>
<td>0.929</td>
<td>1.0</td>
</tr>
<tr>
<td>5</td>
<td>20</td>
<td>0.928</td>
<td>0.796</td>
<td>0.738</td>
</tr>
<tr>
<td>6</td>
<td>20</td>
<td>0.861</td>
<td>0.80</td>
<td>0.40</td>
</tr>
<tr>
<td>7</td>
<td>20</td>
<td>1.0</td>
<td>0.853</td>
<td>0.863</td>
</tr>
<tr>
<td>8</td>
<td>20</td>
<td>0.865</td>
<td>0.498</td>
<td>0.666</td>
</tr>
<tr>
<td>9</td>
<td>20</td>
<td>0.644</td>
<td>0.261</td>
<td>0.799</td>
</tr>
<tr>
<td>10</td>
<td>20</td>
<td>0.286</td>
<td>0.2</td>
<td>0.280</td>
</tr>
<tr>
<td>11</td>
<td>20</td>
<td>0.796</td>
<td>0.865</td>
<td>0.598</td>
</tr>
<tr>
<td>12</td>
<td>20</td>
<td>0.777</td>
<td>0.707</td>
<td>0.665</td>
</tr>
<tr>
<td>13</td>
<td>20</td>
<td>0.925</td>
<td>0.498</td>
<td>0.850</td>
</tr>
<tr>
<td>14</td>
<td>20</td>
<td>0.569</td>
<td>0.343</td>
<td>0.603</td>
</tr>
<tr>
<td>15</td>
<td>20</td>
<td>1.0</td>
<td>0.932</td>
<td>0.731</td>
</tr>
<tr>
<td>16</td>
<td>20</td>
<td>0.918</td>
<td>0.555</td>
<td>0.859</td>
</tr>
<tr>
<td>17</td>
<td>20</td>
<td>0.822</td>
<td>0.775</td>
<td>0.733</td>
</tr>
<tr>
<td>18</td>
<td>6</td>
<td>0.593</td>
<td>0.326</td>
<td>0.175</td>
</tr>
</tbody>
</table>

| All   | 366  | 0.748    | 0.603     | 0.648            |

The kappas are considered good (> 0.6) for the three categories. This indicates strong agreement considering the subjective nature of rationale [7], and demonstrates the high quality of the dataset.

The main challenge we faced during the labelling was understanding the commit messages, such as due to implicit or ambiguous language. As some commit messages are written by (presumed) non-native authors, it was hard to read the tone of the sentences, especially past/future tenses. For example, it is hard to distinguish whether “The oom_reaper end then simply retries if there is at least one notifier which couldn’t make any progress in blockable mode” is a statement before or after the patch. Most commit authors also needed technical understanding. Although we tried our best to interpret what the sentence contains, it is still possible that we misunderstood some of the commit messages. To mitigate any potential bias, we took an inclusive labelling approach. We consider as future work involving Linux developers to obtain a more reliable labelling and a finer-grained analysis.

3.3 Dataset Structure

After removing 99 sentences that were flagged as Inapplicable by at least two annotators, we end up with 2234 sentences. We consider the final classification of these sentences as the union of the labels of the three annotators. We use a comma separated values (CSV) tabular format where each entry (line) refers to a sentence for easy analysis. An example of the dataset structure is shown in Table 4.

4 DATASET DESCRIPTION AND ANALYSIS

4.1 Dataset Description

Here, we describe the distribution of sentences over the agreed-upon labels. We also overview the most used words in each category.

4.1.1 Sentences distribution. Among the 2234 sentences in the dataset, 18 did not fit in any category. Figure 1 shows the distribution of the remaining 2216 labelled sentences over the identified categories. The distribution in Figure 1 shows that the three categories are not clear-cut and that there is a substantial overlap between them. In fact, among the 366 commits in question, only two commits had all their sentences classified with only one label (one of them is the commit presented in Table 2). This could be explained by our inclusive labelling for disagreement. We also expected sentence-level labelling to lead to multi-category classification. In the future, we will investigate further categories, and whether phrase-level labelling can reduce overlap.

Furthermore, we observe a large intersection between Rationale and the other categories. One interpretation of this is that it indicates the subjective nature of rationale. Another is that rationale is often present in the same sentence to motivate decisions and to employ supporting facts. We show examples of such multi-labelled sentences in Table 5. In it, we show a commit from our dataset, along with a colour-coded multi-label classification for each sentence, using the same colour scheme as in Figure 1. As an example of the labelling, the second sentence contains a supporting fact (“... now that there is a separate function for ‘fullmm’ flushing”) that reinforce the rationale behind this patch (“The ‘start’ and ‘end’ arguments to tlb_gather_mmu() are no longer needed”). The third sentence is labelled as both Decision and Rationale as it states not only the patch’s change (“Remove the .. and update all callers”) but also the motivation behind this change, expressed as a value judgment (“unused arguments”).

4.1.2 Frequent words. To better understand how developers express themselves in a large-scale open source project, we extract the most prominent words in each of the categories. We augmented the built-in list of stop words of the wordcloud python library to be removed with the following list of words: OOM, mm, memory, killer, kernel, victim, task, linux, thread, process, system, patch, oom_kill, memc, as they have no added value for our interpretation. We do not consider multi-labelled sentences as we are most interested in discovering the distinct characteristics of each category.

In Figure 2a, we visualize the most frequent words in the 431 sentences labelled as Decision only. Action verbs are very common in how developers phrase their decisions (e.g. “add”, “remove”, “use”, “introduce”, “change”). In Figure 2b, we visualize the most used words in the 49 sentences labelled as Rationale only. We notice that the (“make”, “might”) are the most important. Also, verbs that imply value judgment (e.g. “fixes”, “cleanup”) and positive connotations adjectives (e.g. “useful”, “easier”) are frequent. Finally, we can notice the reference to the future (e.g. “will”, “later”). This indicates that developers tend to express the future positive impact of their patches. In Figure 2c, we visualize the most used words in the 397 sentences labelled as Supporting Facts only. We note the presence of context-specific words (e.g. “kill”, “killed”, “tasks”, “node”). One of the examples is the use of “flushing” in the commit messages that reinforce the rationale behind the patch.

Figure 1: Distribution of the sentences in our OOM dataset

Figure 2: Distribution of the most frequent words in each category.
Table 4: Example of the dataset structure with one entry

<table>
<thead>
<tr>
<th>Column</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>commit number</td>
<td>4</td>
</tr>
<tr>
<td>commit ID</td>
<td>C_kwD0ACN7MtoAKGEoxOWNhZDA20T010TdYjc5YeZIY29H4YTE3YTYmYJhNWFN2Q5MGU</td>
</tr>
<tr>
<td>author name</td>
<td>Andrew Morton</td>
</tr>
<tr>
<td>recommender name</td>
<td>akpm</td>
</tr>
<tr>
<td>message</td>
<td>mm/oom_kill.c: fix vm_oom_kill_table[]</td>
</tr>
<tr>
<td>message_preprocessed</td>
<td>mm/oom_kill.c: fix vm_oom_kill_table[]</td>
</tr>
<tr>
<td>Decision</td>
<td>yes</td>
</tr>
<tr>
<td>Rationale</td>
<td>yes</td>
</tr>
<tr>
<td>Supporting Facts</td>
<td>no</td>
</tr>
</tbody>
</table>

Table 5: An example commit with multi-labelled sentences from our dataset, using the same colour scheme as in Figure 1.

<table>
<thead>
<tr>
<th>Sentence</th>
<th>Labelling</th>
</tr>
</thead>
<tbody>
<tr>
<td>tlb: mmu_gather: Remove start/end arguments from mmu_gather_mmu()</td>
<td>Rationale, Supporting Facts</td>
</tr>
<tr>
<td>The 'start' and 'end' arguments to mmu_gather_mmu() are no longer needed now that there is a separate function for flushing. Retrieve the unused arguments and update all callers.</td>
<td>Decision</td>
</tr>
</tbody>
</table>

average rationale density = \( \frac{\sum_{commits, rationale density}}{\text{number of commits that contain rationale}} \)

Our data set has an average rationale density of 61.43%, in other words rationale information is present in about 60% of a commit message. These results suggest that Linux developers support their decisions with a lot of rationale information, expressing it in a rather detailed way. This finding might serve as a rule of thumb guideline for writing commit messages.

Result 1 – Presence of rationale: Commit messages almost always contain rationale information. On average, around 60% of the message contains rationale information.

4.2 Dataset Analysis

We present seven research questions (RQs) on four themes: (a) the presence of rationale (RQ1, RQ2), (b) the factors that impact it (RQ3, RQ4), (c) its temporal evolution (RQ5, RQ6), and (d) the structure of commit messages from the point of view of rationale (RQ7).

4.2.1 Presence of Rationale. Here, we discuss rationale information abundance (RQ1) and amount (RQ2) in the commit messages.

RQ1. How many commits contain rationale? To answer this question, we compute the rationale % as follows:

\[
\text{rationale density}\% = \frac{\text{number of commits that contain rationale}}{\text{total number of commits}}
\]

We consider that a commit contains rationale if at least one of its sentences is labelled as Rationale. In our dataset, 98.9% of the commits contain at least one sentence with rationale information. This suggests that rationale is almost always described.

RQ2. How much of the commit contains rationale? To answer RQ2, we define the commit-level rationale density metric:

\[
\text{rationale density} = \frac{\text{number of sentences labelled as Rationale}}{\text{total number of sentences in a commit}}
\]

Rationale density is thus the percentage of sentences that contain rationale in a commit. We compute this metric for all the commits. Then, we compute the average rationale density as follows:

4.2.2 Factors impacting rationale. Here, we present analyses about the possible dependencies between the size of the commit (RQ3) and the developers experience (RQ4), and the rationale density.

RQ3. Does the quantity of rationale reported depend on the commit message size? The commit message size refers to the number of sentence entries obtained after preprocessing the commit message. To answer RQ3, first, we do a normality test [8] on the distribution of the rationale density of the commits of our dataset. Results indicate a non-normal distribution (\( p_{value} = 0.02 < 0.05 \)). The normality test also indicates a non-normal distribution for the commit message sizes (\( p_{value} = 1.73e^{-55} < 0.05 \)). Since both distributions are not normal, we use Spearman’s rank correlation coefficient [22] to discover whether or not there is a correlation between the commit message size and its rationale density. Results indicate that there is no correlation between them (\( p_{value} = 2.45e^{-10} < 0.05, R = -0.32 \)).

In Figure 3, we show the rationale density values versus the commit message size (i.e., number of sentences in a commit). The figure shows that the majority of the commits have fewer than 15 sentences, and that a lot of the short commits (fewer than 6 sentences) have a high rationale density (> 0.6). The figure also
shows that as a commit becomes longer, it tends to have between 40% to 60% of its sentences containing rationale information.

**RQ4. Does the quantity of rationale reported depend on the developer experience?** We consider the number of commits authored an indication of the developer’s experience. We count the number of commits per author, as well as the average rationale density per author (i.e., we compute the mean of the rationale density of the commits of each author). The normality test result indicates that the number of commits per author is not a normal distribution ($p_{value} = 4.45e^{-33} < 0.05$). Spearman’s rank correlation coefficient between the number of commits per author and the average rationale density per author indicates that the test is not significant ($p_{value} = 0.635 > 0.05$, $R = -0.05$).

We visualize the average rationale density per author along with the number of commits per author in Figure 4. We identify four regions in the Figure: three for authors with few commits (< 16) and one for authors with many commits (> 16), separated by the vertical dashed line. In fact, only five developers have written more than 16 commits. All the other developers have written fewer than 16 commits, and most of them fewer than 10 commits. More experienced developers’ commits have a consistent rationale density near 60%. This may indicate a guideline for the other developers to target. We further investigate these top contributors and their rationale densities in Section 4.2.3.

In this analysis, we refer to the authors by their initials for ethical reasons [13]. The three regions for the authors with few commits are separated by the two dashed horizontal lines pointing the densities of 0.2 and 0.7. Among these authors, three authors had a very low rationale density (< 0.2). Specifically, authors M.K. and P.E. had a rationale density of 0 as each wrote only one commit that did
not contain any rationale\textsuperscript{23,24}. Author B.S. wrote two commits\textsuperscript{25,26} but their rationale density was low (0.125) as one of them did not contain any rationale, and the other had only one sentence that contained rationale information. We note that all these commits are comparatively old (2009 or before) and that during the labelling process, we noticed that the quality of the messages improved over time. Finally, we note that these commits were approved for merging though they were lacking rationale, meaning that the message combined with the code change may have had implicit rationale that was deemed sufficient.

The top region shows that many authors with few commits have high rationale density. Specifically, 14 authors had a density of 1, e.g., author L.Z. wrote two commits\textsuperscript{27,28} with all sentences containing rationale.

Result 2 – Factors impacting rationale: The quantity of rationale information reported depends neither on the commit message size nor the developers’ experience. Experienced developers have a rationale density around 60%.

4.2.3 Evolution of rationale over time. We first present the evolution of the rationale overall (RQ5) and then focus on the evolution of the rationale for the five main contributors (RQ6). We consider the authoring date of commits, not when it has been accepted.

RQ5. How does rationale evolve over time? To address this question, we visualize the evolution of the rationale density over time. First, we visualize the monthly evolution of the average rationale density in Figure 5. The figure shows a consistency around 0.6. To better contextualize this evolution, we show the yearly evolution of the average rationale density, the average commit message size and the number of commits in Figure 6. The figure shows great variations in the average commit message size and in the number of commits over the years. The figure does not suggest any relationship between these three variables. The correlation test between the average rationale density and the number of commits confirmed this ($p_{\text{value}} = 0.63 > 0.05, R = -0.09$).

Figure 7 shows the evolution of the average rationale density, the average decision density and the average supporting facts density per year. The decision density and supporting facts density are computed similarly to the rationale density. We note that the decision density is always high ($> 0.5$). However, the supporting facts density is usually low ($< 0.6$). We also note that in early and late years (2005-2010 and 2019-2022), the decision density is slightly higher than the rationale density, which is significantly higher than the supporting facts density. Between 2010 and 2019, the three densities seem to be close (around 0.55), although the decision density and the rationale density are almost always higher than the supporting facts density.

RQ6. How does rationale evolve over time for the five core contributors? The five main contributors in the OOM-Killer component are D.R. (82 commits), M.H. (48 commits), T.H. (21 commits), K.M. (20 commits), and O.N. (18 commits). Together, these contributors...

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure5}
\caption{Monthly evolution of the average rationale density}
\end{figure}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure6}
\caption{Yearly evolution of the average rationale density, the average commit message size and the number of commits}
\end{figure}
five contributors wrote 189 commits, i.e., around half of the 366
studied commits. This is consistent with past observations about
open source projects, suggesting that a relatively small number of
core developers are responsible for most contributions [21]. As
discussed in Section 4.2.2, these developers all had a consistent
average overall rationale density around 0.6. Figure 8 shows the
rationale density evolution, the average commit message size, and
the number of commits per year for these contributors. Although
rationale density was consistently around 0.6 for all the years before
2020, it dropped to around 0.4 in 2020 and 2021 and went up to 0.8
in 2022. The figure also shows that the number of commits varies
considerably each year, and that usually, the top contributors write
short commits (fewer than 8 sentences), the year 2021 being the
exception. The average commit message size (in terms of number
of sentences) also seems to be trending up slightly over time.

The culture in the Linux kernel is sufficient to maintain the level of
rationale, and should be encouraged in other software projects.

4.2.4 Structure of commit messages. Here, we discuss the average
structure of a commit message. That is, what sentence category
derevopers prefer when elaborating their commit messages.

RQ7. In what order do the categories mostly appear? We
visualize the distribution of the identified categories over the nor-
malized positions of the sentences of the commit messages in Fig-
ure 9. The figure shows that, in the first 10% of the commit message
(the summary sentence), the Decision category is present in over
350 commit messages. The Rationale also appears frequently in this
summary sentence. This could be explained by the overlap between
the categories and especially by the value judgment words (e.g., “fix”,
“simplify”, “unused”, ...). e.g., the sentence “include cleanup: Update
gfp.h and slab.h includes to prepare for breaking implicit slab.h
inclusion from percpu.h.” is labelled both Decision and Rationale.

In the first half of the commit (10% to 50%), the Supporting Facts
category is the most frequent, with the Rationale category behind
it. In the second half of the commit, the Rationale category exceeds
the Supporting Facts category. That is, the Supporting Facts are
often found in the commit beginning, and before the sentences
containing rationale. This can be explained by their reference to
the past or current state of the system. The Supporting Facts and the
Rationale categories seem to have a similar presence, especially in
the middle of the commit (30%-70%). This can be explained by
the substantial overlap between them (Figure 1). By the end of
the commit message (70%-100%), the Decision category appears again,
with the Rationale category behind it. Thus, the most common
order seems to be: Decision then Supporting Facts then Rationale
then Decision. An example of this order is presented in Table 2.

---

**Result 3 – Evolution of rationale over time:** The level
of Rationale density remains consistent (around 0.6). Decision
density is always high (> 0.5). Supporting facts density
is low (< 0.6). More experienced developers write short
commit messages (fewer than eight sentences).

Understanding the reasons behind this evolution would involve
interviewing the developers and creating a deep understanding of
the historical events of Linux Kernel development and culture as a
whole (in case there are dependencies between the modules that
impacted the OOM-Killer component). We consider performing
this more in-depth analysis as future work.

Investigating these impact factors is relevant for practitioners.
For instance, if rationale trended down with experience or project
age, a best practice might be to periodically train developers to
provide more rationale. Instead, these results could suggest that

---

https://api.github.com/repos/torvalds/linux/git/commits/5a0e3ad6af8660be21ca98a971ad0f331318e05
The identified patterns may be used in the future as guidelines, or enforced via automated tools, to ensure well-structured commit messages, or suggest improvements before merging commits.

**Result 4 – Structure of commit messages:** Developers tend to start and end their commit messages with *Decisions*, *Rationale* and *Supporting Facts* appear in the middle of the commit, with *Supporting Facts* usually preceding *Rationale* sentences.

We summarize the results of our analysis of the dataset in Table 6.

### 5 THREATS TO VALIDITY

**Construct validity.** The measures we use to identify the presence of the different categories in the commits (i.e., the densities), are quantitative measures. We do not measure this presence qualitatively. In fact, we are not aware of any qualitative measures of rationale or decisions. Furthermore, we consider the number of commits written as an indicator of author experience. This is not necessarily true. The number of commits is also computed as an overall classification for the author and not at the time of the commit. This introduces a threat as authors might have gained experience during the studied period. However, our findings mitigate this threat as we can see a clear difference between the authors with several commits and the authors with few commits (Figure 4). Thus, the data leads us to believe that the number of commits authored is a reasonable measure of experience.

**Internal validity.** It is possible that our manual labelling process could have introduced unintentional bias, as only one annotator is a native English speaker. To mitigate this, we had a total of three individuals involved in labelling the commits, developing a shared codebook to use as guide. We also had several piloting rounds and discussions throughout the labelling process. An average Fleiss kappa of 0.65 indicates a good reliability for our labelling. To mitigate cases of potential bias due to annotator background, we took an inclusive labelling approach during consolidation meetings. This approach assigned a final set of labels to a sentence based on the union of labels from all annotators.

Our decision to filter the 99 *Inapplicable* sentences might introduce bias, as the removed sentences might contain relevant information. To mitigate this threat, we only removed the sentences where two or more annotators agreed that it was *Inapplicable*.

We only studied commits that were approved by the Linux maintainers, which could introduce survivor bias. This is acceptable as we are interested in the rationale of the software as it is, not as it could have been. Comparing our findings with the characteristics of rationale in rejected commits is however an interesting research question for the future. Additionally, some metrics are calculated based on a low number of commits (e.g., some monthly and yearly averages when studying rationale evolution). This represents a threat as it may mean that the results are more metrics of a few particular commits rather than true evolutionary trends.

**External validity.** We only studied the commit history that is available on Git, from 2005 onwards. Our observations might not be applicable to contributions in the earlier years of Linux.

Additionally, there is no reason to assume that our findings can be extrapolated to a) other Linux components, or b) other OSS projects. Regarding a), we note that, to the best of our understanding, there is nothing radically different about the development of the OOM Killer module compared to the rest of the kernel. Regarding b), we note that while there are real differences between the way contributions are managed in Linux compared to other large OSS projects, there are also lessons to be learned across projects [4]. We view this paper as a step towards creating a better understanding of rationale in OSS generally, which can lead to improvements in automatic rationale extraction across projects.

### 6 RELATED WORK

*About the Linux Kernel.* Spinellis et al. studied the evolution of Unix from an architectural perspective [29]. Patel et al. studied...
the logging practices in the Linux Kernel [23]. Trinkenreich et al. [34] surveyed Linux Kernel contributors to develop a theoretical model for the sense of virtual community in open source software. However, none of this prior research focused on the rationale in the kernel commit messages.

About Commit Messages. Al Omar et al. [3] explored how developers document their activities in refactoring commits. They manually extracted the patterns used when refactoring (i.e., the keywords or phrases that frequently occur in refactoring-related commits). Other researchers tried to detect developer rationale from the refactoring commit message to recommend the refactoring that would meet the developer’s intentions [25]. Our work is different as we do not focus on one specific category of commits (i.e., refactoring commits). They also consider the rationale as the difference of the quality attribute values before and after the commit, while we study the rationale information present in free text.

Approaches have been proposed to automatically generate commit messages [32]. However, these are based on datasets that include poorly phrased commit messages. In fact, it is only recently that researchers have studied commit message quality [18, 33]. Tian et al. has tried to define what constitutes a “good” commit message [33], and has found that it should summarize what was changed, and describe why those changes are needed. They also developed a taxonomy based on recurring patterns in commit message expressions and proposed a good-message identification tool. Li et al. [18] consider link contents in addition to the commit message to train classifiers for the automatic identification of good commit messages. They also studied the commit quality evolution (i.e., whether a commit contains why and what information). Similar to our study, these works (i.e., [18, 33]) involved manually creating a commit dataset from open source projects and investigating the temporal evolution aspect. They also distinguished the evolution for the core and non-core developers. However, they only considered the evolution of the existence of what (decision) and why (rationale) information over time, while we study the evolution of their quantities. Interestingly, our results indicate stable amounts, while they found that the overall quality degrades over time. In addition, they study the correlation between defect proneness and the quality of the commit message as well as the quality of prior commits, while we focus on different correlations. They have performed their studies at the commit-level, while we label and analyze at the sentence-level. Finally, our study focuses on a component from the Linux kernel, while they report results across 32 Apache projects.

About Design Decisions and Rationale. Li et al. created a ground truth dataset from the Hibernate developer mailing list by labelling sentences as decision or non-decision [19]. Bhat et al. manually analyzed and labelled more than 1,500 issues from two large open source repositories [5]. They classified the issues into different decision categories. Unlike our work, they only considered decisions and discarded rationale expressions.

Sharma et al. studied Python email archives and produced a labelled dataset of rationale sentences behind acceptance of Python Enhancement Proposals (PEPs) [27]. They also defined 11 categories of rationale and used them to classify the rationale sentences. This work differs because a) they consider emails while we consider code commits and b) their work is specific to the PEPs, as the decisions they consider refer to the possible transitions between the PEP states (e.g., draft, accepted, refused). Kleebau et al. propose Condec tools to support documenting and managing decision knowledge for change impact analysis [17]. These tools include an automatic text classification feature for rationale. To provide ground truth for the classifier, the authors manually labelled the textual artifacts they produced when developing the Condec tools (Jira issue text, commits and code comments) to whether or not it contains rationale and they identified rationale categories (decision, issue, alternative, pro-argument, con-argument). Different from our work, they consider all textual artifacts and not only code commits and do not analyze the resulting dataset.

Vanderven et al. created a commit dataset from 710 Git projects, by asking six experts to label 100 commits if they involved a decision, rationale for a decision, and decision alternative information [35]. However, their dataset is not publicly available, and they work at the commit-level. Furthermore, their analysis only investigated the presence of rationale, decisions and alternatives in the commit messages. They neither studied the evolution of their quantities over time, nor the order in which they appeared. Alkadhi et al. investigated the presence of rationale in the Internet Relay Chat (IRC) chat messages of three OSS projects: Apache Lucene, Mozilla Thunderbird and Ubuntu [2]. They manually indicated if each message contains rationale, and identified the type of rationale elements (decision, issue, alternative, pro-argument, con-argument) included in the message. They also provided evidence of rationale existence in the chat messages, and the frequency of different rationale elements. Furthermore, they explored how developers contribute to rationale in the messages. This work differs from ours because 1) they consider chat messages instead of code commits, 2) they work at the message-level while we work at the sentence level, and 3) they did not consider the Linux kernel project.

Recently, several researchers proposed rationale representations. Hesse et al. proposed a documentation model for decision knowledge built upon the results investigating the comments to 260 issue reports from the Firefox project [14]. Soliman et al. worked on an empirically-grounded ontology for architecture knowledge from StackOverflow posts [28]. Neither one of these works considered rationale representation for code commits. Alsafwan et al. performed interviews and surveys with developers to study their perspective of rationale for code commits, and found that they decompose the rationale of code commits into 15 separate components [1].

7 CONCLUSION AND FUTURE WORK

We explored the presence of rationale in the commit message history of the Linux OMM-Killer. We created a dataset by extracting and manually labelling the commits. We analyzed it over seven research questions about the presence of rationale, its evolution, the factors impacting it and the structure of rationale information. The results are summarized in Table 6.

This lead us to insights about the nature of rationale. First, it is prevalent in the commit messages with a stable amount over the studied period. Second, there is a substantial overlap between the three defined categories, as expressions of rationale are very often interwoven with decisions and supporting facts. This may suggest the need to consider a finer-grained analysis in the future.
(e.g., clause-level analysis). Third, experienced developers are responsible for writing almost half of the commits, have a consistent rationale density, and usually write short messages. Finally, a common commit structure seems to emerge: decision then supporting facts then rationale then decision.

In the future, first we aim to increase the dataset quality and richness. We plan to classify the commits (e.g., trivial fix, refactoring, new feature) to investigate how rationale varies according to context. We are also investigating adding a large language model as another annotator. Syriani et al. show that ChatGPT can offer human-level performance for a similar labelling task, but this requires systematic prompting and performance assessment [31]. Second, we will investigate rationale management, e.g., training an automatic rationale extraction model, using our dataset as the ground truth. Along with our rationale structure insights, we could automatically detect commit messages that lack rationale or lack the correct structure when commit messages are written or requested to be merged.

ACKNOWLEDGMENTS

This work is partially funded by a Fonds de Recherche du Québec – Nature et Technologies (FRQNT) Doctoral Research Scholarship (B2X).

REFERENCES


