Demystifying Software Release Note Issues on GitHub

Jianyu Wu, Hao He, Wenxin Xiao, Kai Gao, Minghui Zhou*
School of Computer Science and School of Software & Microelectronics, Peking University, Beijing, China
Key Laboratory of High Confidence Software Technologies, Ministry of Education, Beijing, China
{wujianyu,heh,gaokai19,zhmh}@pku.edu.cn,wenxin.xiao@stu.pku.edu.cn

ABSTRACT

Release notes (RNs) summarize main changes between two consecutive software versions and serve as a central source of information when users upgrade software. While producing high quality RNs can be hard and poses a variety of challenges to developers, a comprehensive empirical understanding on these challenges is still lacking. In this paper, we bridge this knowledge gap by manually analyzing 1,731 latest GitHub issues to build a comprehensive taxonomy of RN issues with four dimensions: *Content*, *Presentation*, *Accessibility*, and *Production*. Among these issues, nearly half (48.47%) of them focus on *Production*; *Content*, *Accessibility*, and *Presentation* take 25.61%, 17.65%, and 8.27%, respectively. We find that: 1) RN producers are more likely to miss information than to include incorrect information, especially for breaking changes; 2) improper layout may bury important information and confuse users; 3) many users find RNs inaccessible due to link deterioration, lack of notification, and obfuscate RN locations; 4) automating and regulating RN production remains challenging despite the great needs of RN producers. Our taxonomy not only pictures a roadmap to improve RN production in practice but also reveals interesting future research directions for automating RN production.

CCS CONCEPTS

• **Software and its engineering → Documentation**.

KEYWORDS

release engineering, release note, empirical study, taxonomy

ACM Reference Format:
Jianyu Wu, Hao He, Wenxin Xiao, Kai Gao, Minghui Zhou. 2022. Demystifying Software Release Note Issues on GitHub. In *30th International Conference on Program Comprehension (ICPC '22), May 16–17, 2022, Virtual Event, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3524610.3527919

1 INTRODUCTION

When releasing a new software version, developers often produce a **release note** (RN) which summarizes main changes in the software since its previous release [117]. RNs serve as means of communication between the software and its users [102]. Consulting RNs is considered as an essential best practice when upgrading software [10]. Users typically use RNs to comprehend: 1) potentially beneficial changes, such as bug fixes, enhancements, new features, to help them decide whether to upgrade to the new release; 2) potentially interrupting changes, along with guidance for migration or mitigation. Besides, internal developers use RNs to formally document development progress and plans for the next release [102].[1]

For large software projects, the production of RNs is both time-consuming and error-prone. The survey by Moreno et al. [117] found that *"creating a release note by hand is a difficult and effort-prone activity that can take up to eight hours"*. The tight deadlines in agile software development may even tempt developers to reduce effort put into RNs [66]. Consequently, the produced RNs may be of low quality (bad organization, missing important changes, etc.), which brings various problems to software users. However, previous researches [97, 102, 117, 126] mainly focus on categorizing RN content and automated RN generation, while a systematic understanding of real RN issues in practice (i.e., how RNs go wrong or fail to meet users' expectations) is still lacking. Such an understanding can help formulate best practices and reveal important future research directions for automating and regulating RN production. Therefore, to bridge the knowledge gap, we ask the following research question (**RQ**): *What are the RN issues faced by developers?*

To answer this RQ, we collect 1,731 RN-related GitHub issues from GHArchive [13] and build a comprehensive taxonomy of these issues using multiple rounds of open coding. The taxonomy is further validated through semi-structured interviews. The final taxonomy consists of four main dimensions: **Content (251, 25.61%)**, **Presentation (81, 8.27%)**, **Accessibility (173, 17.65%)**, and **Production (475, 48.47%)**, that reveals the challenges of using RNs and therefore the problems of producing RNs. To the best of our knowledge, this is the *first* paper that provides such a taxonomy.

Based on our taxonomy, we derive a practitioner-oriented checklist for RN production, which involves the selection of appropriate content, organization, and writing style for RNs. We additionally provide recommendations for regulating RN production and ensuring RN completeness. Finally, we identify open research challenges, which can benefit the automation of RN production and testing of RN completeness/correctness in practice. We provide a replication package at https://doi.org/10.6084/m9.figshare.18777650.

2 BACKGROUND AND RELATED WORK

In the early years of software development, software products are often released "once and for all" with no modifications after the

*Minghui Zhou is the corresponding author.

[1]Note that the term "release note" often refers to the documentation that refine and summarize change logs. However, in practice many software projects directly use change logs as their release notes, so some results in our paper refer to both. Throughout this paper, we use the term "user" to refer to anyone reading RNs or referring RNs for their tasks. A user can be an internal developer, a downstream developer, or a software end user.

initial release. However, successful software inevitably evolves into new versions. When a new version needs to be released, documentation for explaining changes in this version, i.e., Release Note (RN), emerges as a natural requirement. Although we cannot precisely trace the history of the earliest RNs, the term "release note" has at least been used in the software industry since the 1980s [108].

From the beginning of the 21st century, the movement toward agile software development advocates "release early, release often" so that a tight feedback loop between developers and users can be created [119]. Consequently, the required effort to manage changes between consecutive software versions has significantly increased. Then, software projects begin to formulate systematic agendas for software release management, in which RNs are perhaps the most important kind of documentation [98]. Nowadays, complex software systems such as Firefox have to deal with a tremendous amount (up to thousands) of patches during each release cycle, which creates a formidable challenge in tracking changes to be included in a RN and producing the final RN. For Firefox, the Mozilla team defines a systematic process, including workflows, conventions, and automated tooling, to support the creation of RNs [12].

Meanwhile, RNs remain an understudied research topic. Early studies only use RNs as a data source for understanding other software maintenance and evolution topics [100, 116, 121, 127]. It is not until the recent decade do researchers begin to study RNs themselves with two main fronts: empirical studies for understanding RN practices and approaches for automated RN generation.

2.1 Understanding Release Note Practices

Moreno et al. [117] manually analyze 1,000 RNs from 58 industrial and open source projects. They identify 17 common change types in RNs, such as fixed bugs, new features, and new code components. Similarly, Abebe et al. [97] manually analyze 85 RNs from 15 software projects and identified six types of information: title, system overview, resource requirement, installation, addressed issues, and caveat. Bi et al. [102] study the characteristics of 32,425 RNs from 1,000 GitHub projects. They classify common RN content into eight topics including issues fixed, new features, system internal changes, etc. They find that RN content significantly differs across software in different domains, e.g., for application software and system software, new features are most frequently documented. They further uncover discrepancies between RN producers and users through interviews and surveys. However, it is still unclear *what content tends to go wrong in RNs*, which may have a different distribution.

The nature of RN is also discussed in some work related to software documentation. Aghajani et al. [98] perform a survey with 146 developers to investigate what kind of documentation types are considered as important in software development. They find that although the majority of developers consider RNs and change logs as important, their absence are also among their frequently encountered issues. Developers also suggest including documentation such as RNs as mandatory items in the release checklist.

Despite the discrepancies between RN producers and users as identified by Bi et al. [102], we still lack a comprehensive empirical understanding of real issues in RN production and usage. To the best of our knowledge, this is the *first* paper toward this direction, and our taxonomy provides a significant amount of new empirical evidence for improving RN production in practice.

2.2 Automating Release Note Production

Since producing RNs is both important and effort-prone, developers naturally begin to explore ways to automate this process. For software projects managed via a version control system (VCS), the most straightforward way of producing a RN is to aggregate all changes from the VCS (e.g., aggregating all commit messages from Git). However, such simple way of automation comes with severe drawbacks, as noted by the OpenStack documentation:

"Release notes are not meant to be a replacement for git commit messages. They should focus on the impact for the user and make that understandable, even for people who do not know the full technical context for the patch or project" [92].

To facilitate the production of high quality RNs while reducing manual effort, many open-source projects begin to adopt tools for automated RN generation, including Semantic Release [95] (~14k stars), github-changelog-generator [14] (~6k stars), Release It [91] (~4k stars), Release Drafter [90] (~2k stars), etc. All tools make the assumption that every software change should be documented using predefined templates or labels so that they can generate RNs based on predefined rules. For example, Semantic Release requires developers to write commit messages in the format specified by Angular Commit Message Conventions [4] with eight types of predefined changes. These tools are generally designed to be easily extensible and configurable to fit the needs of different projects. Even if some automation is adopted, it is still common to post edit the RNs to summarize changes, highlight, or intrigue readers, etc.

To improve the state of practice, researchers have proposed novel approaches for automated RN generation. Klepper et al. [113] propose a semi-automated RN generation tool which extracts change descriptions from issue trackers and organizes them by labels to fit the need of a specific audience. Moreno et al. [117] propose a fully automated RN generation tool, ARENA, which integrates both changes from VCS and rationales for each change from issue trackers into RNs with predefined change categories. Nath et al. [118] propose to generate RNs from commit messages and pull requests using text summarization and word embedding techniques. Jiang et al. [111] propose a language-agnostic approach to produce RNs from pull request text using deep learning.

While several automated approaches have been proposed by researchers, we are still not aware of any wide industrial adoption, indicating potential discrepancies between research and practice. Our work complements existing effort on RN automation by summarizing best automation practices and reveal future research directions for improving automated tools.

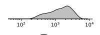
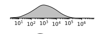
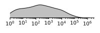
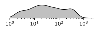
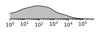
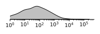
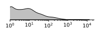
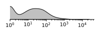
3 METHODOLOGY

3.1 Data Collection

In this study, we choose to analyze GitHub issues, which developers use to track ideas, provide feedback, report bugs, and initiate discussions [1]. We favor GitHub issues over Stack Overflow questions because GitHub issues contain more information such as reports and discussion among developers and provide concrete examples of how RNs fail, apart from developers' opinions.

3.1.1 Mining GitHub. GitHub is one of the most popular social coding platforms and provides access control and several collaboration features such as bug tracking, feature requests, task management

Table 1: Repository Statistics of the Final Issue Dataset

	Median	Mean	Std.	Distribution*
Age (in Days)	1,483.00	1.676.36	1,058.27	
# of Commits	1,053.00	7,357.25	36,020.93	
# of Stars	188.00	4,321.06	13,536.23	
# of Contributors	28.00	88.23	122.39	
# of Forks	69.50	1,024.87	3,430.94	
# of Issues	53.00	426.57	2,517.74	
# of PRs	5.00	36.22	188.31	
# of Releases	13.00	60.50	250.59	

* We increment all values by one to plot the distribution in log-scale.

for every project. It is a commonly used data source for exploring software issues in previous works [104, 110]. To this end, we use the GHArchive dataset [13] to collect all GitHub issues that: 1) have activities (at least one `IssueEvent` in GHArchive) between January 2021 and June 2021; 2) contain the keyword "release note" in their titles. We only include the latest GitHub issues (with activities in 2021) because we observe that RN practices are rapidly changing in open-source communities, and thus data timeliness is vital. For example, Bi et al. [102] report that developers do not use automated RN generation tools while the number of automated RN generation tools are gaining increasing popularity recently (Section 2.2). This initial selection results in 1,731 issues from 1,019 repositories.

3.1.2 Refining Dataset. Two authors (named as inspectors), both with over six years of software development experience, further read all the issues jointly to refine the final dataset. The inspectors browse through the GitHub issue pages of the all collected issues together as an initial familiarization of the dataset and exclude the 822 issues that are not related to certain problems in RNs (i.e., *False Positives*), including the following cases:

➢ *Release Statements (491, 28.37%)*: The issue is only an official announcement of a release or a release note.
➢ *Non-Informative (137, 7.91%)*: The issue contains too little information (e.g., only a few words in title and description) to be understood by the inspectors.
➢ *Irrelevant (121, 6.99%)*: The issue happens to have the keyword "release note" in its title but actually refers to a problem not related to RNs.
➢ *Unreachable (42, 2.43%)*: The issue is no longer available on GitHub (e.g., the repository is deleted or made private, the issue is deleted, etc.).
➢ *Non-English (17, 0.98%)*: The issue contains non-English text and is not understandable by the inspectors.
➢ *Mistake (14, 0.81%)*: The issue reporter misunderstands the RN and reports a non-existent problem.

The final dataset for our study consists of 909 issues from 722 repositories. The repository statistics are summarized in Table 3, where we can observe that most issues come from repositories with long development history, high popularity, and sufficient development activities.[2] In fact, given that only software with a sufficiently large user base may consider writing RNs or have users reporting issues for RNs, it is natural that almost all of these issues come from mature software repositories. The size of our dataset is comparable to and even larger than similar software engineering studies that conduct qualitative manual analysis on text (e.g., studies on Stack Overflow posts and patch descriptions [99, 101, 103, 124, 130]).

[2]The long tail distributions of most metrics are expected and common in mining software repository datasets [107, 131].

Table 2: Statistics for Each Round of Manual Labeling

Round	1	2	3	4	5*	Total
Analyzed	273	212	212	212	90	909
Cohen's Kappa	-	0.78	0.86	0.87	-	-
Newly Added						
- #Dimensions	3	1	0	0	0	4
- #Categories	5	2	0	0	0	7
- #Subcategories	7	4	1	0	0	12
- #Leaf Nodes	48	7	8	2	-3	62

* The fifth round samples issues from previous four rounds (Section 3.2.3, 3.2.4).

3.2 Analysis Method

For the final 909 RN-related issues, we follow an open coding procedure to inductively create the dimensions, categories, subcategories, and leaf nodes of our taxonomy in a bottom-up way [120]. Similar to previous works [103, 110], our procedure of taxonomy construction consists of four steps: pilot construction, extended construction, developer interview, and reproducibility verification. The four steps are integrated with a five-round labeling process and the statistics for each round of labeling are summarized in Table 2.

3.2.1 Pilot Construction. We randomly sample 30% (273) of the 909 issues for a pilot construction of the taxonomy in the first round with two stages. The inspectors mentioned in Section 3.1.2 independently analyze the underlying RN problems behind the sampled issues. In the first stage, the inspectors aim to be familiar with RNs' issues. They read and reread titles, descriptions, labels, and comments of each RN-related issue to understand its problems and intention. Where necessary, they additionally check relevant code changes (i.e., pull requests/commits) and release notes that reveal the final solution adopted by project developers. In the second stage, the inspectors assign short phrases as initial codes and record important information to indicate the problems and needs behind these issues. If an issue is related to multiple problems and needs, e.g., the RN misses both new features and breaking changes, it will be assigned with multiple initial codes. After the initial codes are generated, the inspectors proceed to group similar codes into categories, create a hierarchical taxonomy of RNs' issues, and assign issues to the taxonomy. We include an additional arbitrator, who has several publications in top-tier software engineering venues and more than six years of software development experience, to mediate, discuss, and resolve any disagreement during taxonomy construction. They continuously go back and forth between categories and issues to refine the taxonomy until the inspectors and the arbitrator finally approve all categories in the taxonomy.

3.2.2 Extended Construction. Based on the initial hierarchical taxonomy generated in Section 3.2.1, the inspectors and the arbitrator iteratively conduct independent labeling, conflict resolution, and taxonomy refinement in the next three rounds. In each round, two inspectors first independently label one-third of the remaining issues. When they find issues that cannot be labeled in the current taxonomy, they add them to a temporary *Pending* category. Then, the inspectors and the arbitrator organize a meeting to resolve labeling conflicts and determine whether new categories should be added for issues in the *Pending* category. After the taxonomy is refined, they update all previously labeled issues into the refined taxonomy and proceed to the next round. Saturation is reached in the third round because we add only new leaf nodes (Table 2). We finish labeling all the issues in the fourth round. In the three rounds of extended construction, we use Cohen's Kappa (κ) to measure inter-rater agreement between two inspectors. The κ values are 0.78, 0.86, and 0.87, respectively, indicating increasing and high agreement between inspectors.

3.2.3 Developer Interview. To validate our taxonomy with practitioners, we interview three industry software engineers from different large IT companies. They all have rich experience in publishing RNs with 1.5, 3, and 7 years of experience, respectively.

We opt for *semi-structured* interviews. Each of our interviews begins with the question: *what issues have you encountered around RNs in your software development process?* The purpose of this open-ended question is to see if our taxonomy covers the problems that developers usually encounter during development. They each describe three, five and two issues they encountered based on their own development experience. Then, we present our taxonomy and direct them to specific categories of issues in our taxonomy, which enables them to recall other four previous issues. All issues are covered by our taxonomy, indicating that our taxonomy has good coverage even within a different context (i.e., industry setting).

Then, we ask them to review and provide suggestions about our taxonomy. They think our taxonomy is clear and informative, though some leaf nodes can be improved. After discussion, we decide to merge seven leaf nodes into three leaf node and split one leaf nodes into two leaf nodes finally. The interview time varied between 46 minutes and 2 hours. All of interviews are conducted face to face with two authors (one is the leader and the other one asks additional questions when appropriate). The reason is that previous works [109, 110] show that participants talk much more when more than two interviewers conduct the interviews.

3.2.4 Reproducibility Verification. One problem remaining with our taxonomy is reproducibility because we intertwine taxonomy construction with independent labeling. This is hard to avoid because the taxonomy is too complex to be precisely defined in one or two rounds. Although we maintain a code book during the process, it is still unclear whether others can reproduce the taxonomy using the same code book. Therefore, we invite two interviewees and one additional Ph.D. candidate to label issues using our code book. Each of them is assigned 30 different issues and they return their results after 3 days.[3] Compared with our own results, the κ values are 0.93, 0.89, and 0.86, respectively, which also indicates a high agreement and thus good reproducibility.

Our final taxonomy includes four dimensions, seven categories, 12 subcategories, and 62 leaf nodes. The entire manual construction process takes over two months to finish.

4 RESULTS

Figure 1 illustrates the hierarchical taxonomy of RN issues. We group all these issues into four **dimensions**:

(1) **Content**: What information should RNs convey?
(2) **Presentation**: How should RNs convey information?
(3) **Accessibility**: How to make RNs easily accessible?
(4) **Production**: In what way should RNs be produced?

Each dimension is then hierarchically organized into **categories** (e.g., *Completeness*), **subcategories** (e.g., *Missing*), and **leaf nodes** (optional, e.g., *Missing Breaking Changes*). Figure 1 also shows the number of issues and percentages (within dimension) for all dimensions, categories, subcategories, and leaf nodes in the taxonomy. In the remainder of this section, we will describe our taxonomy with representative examples.

4.1 Content

In total, 251 issues discuss the **Content** of RNs, i.e., what information should RNs convey. Issues from the **Content** dimension can help better understand 1) what common mistakes developers often make when producing RNs, 2) what typical users would expect from RNs, and 3) what purposes RNs should serve as one kind of project documentation. This dimension consists of two categories: *Completeness* and *Correctness.*

4.1.1 Completeness (157, 62.55%). This category of issues concerns whether RNs contain both sufficient and necessary information required by users during software upgrades or required by internal developers for maintenance purposes. It has three subcategories: *Missing*, *Insufficient*, and *Unwanted.*

❖ *Missing (133, 52.99%)* subcategory refers to issues stating that some information perceived important by end users or internal developers is not included in the RN at all. The most frequently missed information in RNs includes:

- ➢ *Breaking Changes (31, 12.35%)*: Such issues are predominant because end users directly encounter upgrade failures if they are not notified of breaking changes from reading RNs. However, it can be difficult for RN producers to correctly locate and highlight breaking changes in RNs. For example, a developer from `mongoose` notes that *(the new version) has many errors, and fixing them is not just changing a function/field name, because function parameters/semantics have also changed* because of the *undocumented breaking changes from v6 to v7* [21].
- ➢ *Links (18, 7.17%):* In these issues, developers ask for links to external materials (e.g., related PR/issues/commits, usage guides, CVEs, etc.) to better understand information conveyed in RNs.
- ➢ *New Features (16, 6.37%):* Some implemented new features may be ignored in RNs, and (other) developers open issues in need of documenting their contributions.
- ➢ *Dependency/Environment Specification (13, 5.18%):* Undocumented dependency or environment specification may also accidentally break clients when users upgrade to new versions.
- ➢ *Migration/Usage Instruction (11, 4.38%):* Some developers ask for migration or usage instructions in RNs to help them understand the impact of breaking changes and upgrade their client code.
- ➢ *Version Information (11, 4.38%):* Some developers open issues to discuss about adding version information in RNs, e.g., release date, version number & name, checksum, and release status (draft or final) for easy reference to specific releases.
- ➢ *Attribution (7, 2.79%):* Some issues are opened by repository members to discuss about missing attribution to certain participants

[3] We do not assign more because inspecting, comprehending, and labeling issues takes significant time and energy which they lack to label more.

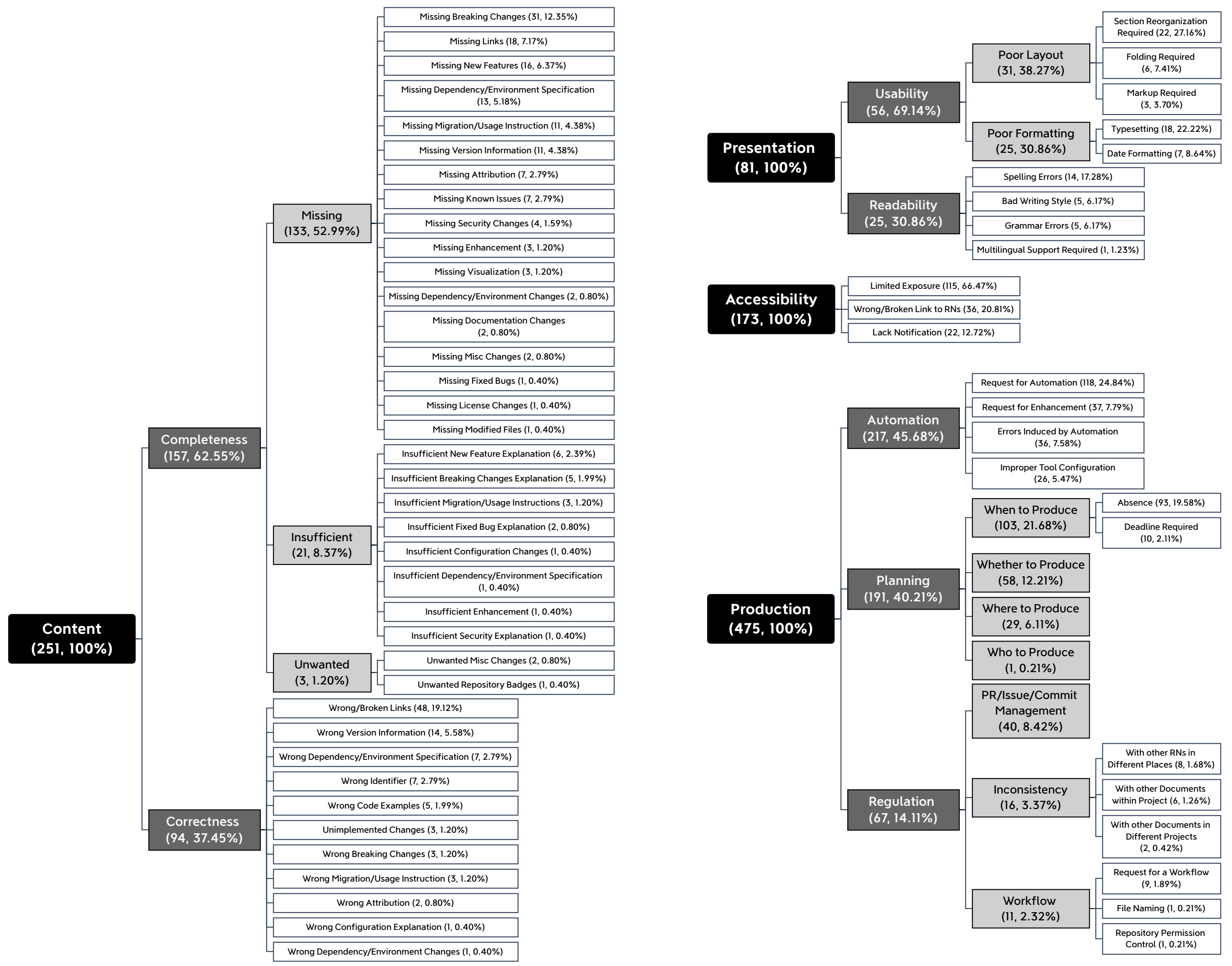

Content (251, 100%)
Completeness (157, 62.55%)
Missing (133, 52.99%)
Missing Breaking Changes (31, 12.35%)
Missing Links (18, 7.17%)
Missing New Features (16, 6.37%)
Missing Dependency/Environment Specification (13, 5.18%)
Missing Migration/Usage Instruction (11, 4.38%)
Missing Version Information (11, 4.38%)
Missing Attribution (7, 2.79%)
Missing Known Issues (7, 2.79%)
Missing Security Changes (4, 1.59%)
Missing Enhancement (3, 1.20%)
Missing Visualization (3, 1.20%)
Missing Dependency/Environment Changes (2, 0.80%)
Missing Documentation Changes (2, 0.80%)
Missing Misc Changes (2, 0.80%)
Missing Fixed Bugs (1, 0.40%)
Missing License Changes (1, 0.40%)
Missing Modified Files (1, 0.40%)
Insufficient (21, 8.37%)
Insufficient New Feature Explanation (6, 2.39%)
Insufficient Breaking Changes Explanation (5, 1.99%)
Insufficient Migration/Usage Instructions (3, 1.20%)
Insufficient Fixed Bug Explanation (2, 0.80%)
Insufficient Configuration Changes (1, 0.40%)
Insufficient Dependency/Environment Specification (1, 0.40%)
Insufficient Enhancement (1, 0.40%)
Insufficient Security Explanation (1, 0.40%)
Unwanted (3, 1.20%)
Unwanted Misc Changes (2, 0.80%)
Unwanted Repository Badges (1, 0.40%)
Correctness (94, 37.45%)
Wrong/Broken Links (48, 19.12%)
Wrong Version Information (14, 5.58%)
Wrong Dependency/Environment Specification (7, 2.79%)
Wrong Identifier (7, 2.79%)
Wrong Code Examples (5, 1.99%)
Unimplemented Changes (3, 1.20%)
Wrong Breaking Changes (3, 1.20%)
Wrong Migration/Usage Instruction (3, 1.20%)
Wrong Attribution (2, 0.80%)
Wrong Configuration Explanation (1, 0.40%)
Wrong Dependency/Environment Changes (1, 0.40%)
Presentation (81, 100%)
Usability (56, 69.14%)
Poor Layout (31, 38.27%)
Section Reorganization Required (22, 27.16%)
Folding Required (6, 7.41%)
Markup Required (3, 3.70%)
Poor Formatting (25, 30.86%)
Typesetting (18, 22.22%)
Date Formatting (7, 8.64%)
Readability (25, 30.86%)
Spelling Errors (14, 17.28%)
Bad Writing Style (5, 6.17%)
Grammar Errors (5, 6.17%)
Multilingual Support Required (1, 1.23%)
Accessibility (173, 100%)
Limited Exposure (115, 66.47%)
Wrong/Broken Link to RNs (36, 20.81%)
Lack Notification (22, 12.72%)
Production (475, 100%)
Automation (217, 45.68%)
Request for Automation (118, 24.84%)
Request for Enhancement (37, 7.79%)
Errors Induced by Automation (36, 7.58%)
Improper Tool Configuration (26, 5.47%)
Planning (191, 40.21%)
When to Produce (103, 21.68%)
Absence (93, 19.58%)
Deadline Required (10, 2.11%)
Whether to Produce (58, 12.21%)
Where to Produce (29, 6.11%)
Who to Produce (1, 0.21%)
Regulation (67, 14.11%)
PR/Issue/Commit Management (40, 8.42%)
Inconsistency (16, 3.37%)
With other RNs in Different Places (8, 1.68%)
With other Documents within Project (6, 1.26%)
With other Documents in Different Projects (2, 0.42%)
Workflow (11, 2.32%)
Request for a Workflow (9, 1.89%)
File Naming (1, 0.21%)
Repository Permission Control (1, 0.21%)

Figure 1: The Taxonomy of Release Note Issues. (■) Represents Dimensions, (▩) Represents Categories, (░) Represents Sub-Categories, and (□) Represents Leaf Nodes.

(e.g., contributors, funders, commenters, and reviewers, etc.). As stated by a maintainer of coq, *in open source software, it is very important to give credit* [67].

➢ *Known Issues (7, 2.79%):* Several issues mention that specific unsolved issues should be included in RNs to alert end users, e.g., including a `NullPointerException` crash and its workaround in the corresponding RN of `NuGet` [36].

Other kinds of information may also be reported as missing, though less frequently, including notification of security changes, enhancements, visualization (additional diagrams or plots), documentation changes, fixed bugs, license changes, modified files, etc.

❖ Issues in the *Insufficient (21, 8.37%)* subcategory arise because certain information related to important changes is not sufficiently detailed for users to understand. Two kinds of explanations are most likely to be insufficient in RNs:

➢ *New Feature Explanation (6, 2.39%):* Developers tend to ask for more information about unfamiliar new features if they intend to use them after upgrading. For example, `Keras` 2.0 renames `samples_per_epoch` to `steps_per_epoch` in `fit_generator()` but its RN fails to mention additional changes in parameter semantics, which confuses downstream developers [19].

➢ *Breaking Change Explanation (5, 1.99%)*: Developers also ask for more clarification about changes that may break downstream code. We observe a vivid example in `numpy` where a developer opens an issue to argue that *we should try to improve the release notes (and probably warnings) for the* `np.int` *and other python alias deprecations* [26].

Other insufficiently explained information include: migration/usage instructions, fixed bugs, configuration changes, dependency/environment specification, enhancements, and security.

❖ Interestingly, three issues care about *Unwanted* information in RNs, but they are likely to be only occasional. Two issues state that only critical/developer impacting changes should go in release notes instead of listing all miscellaneous changes [28], while the other issue [44] mentions that repository badges should not occur in release notes.

4.1.2 *Correctness (94, 37.45%).*

This category means that information described in RNs conveys inaccurate information.

❖ Contrary to our intuition, the majority is *Wrong/Broken Links (48, 19.12%)*, which refers to cases where links in RNs cannot be opened or direct to an incorrect page. Most links should point to other kinds of documentation, e.g., user guide, for the elaboration of changes in RNs; others are expected to point to related PR/issue/-commit, project main branch, the homepage of other projects, RNs of sibling projects, files for download, etc. These links are supposed to supplement information, but they tend to deteriorate over time, which causes poor reading experience for RN readers.

❖ Moreover, 14 issues are related to *Wrong Version Information (14, 5.58%)*, including version number/name, version date, checksum, and most of which are caused by copy-pasting from previous RNs [22, 51, 75]. Other kinds of change description that can go wrong include identifier, dependency/environment specification, code examples, breaking changes, migration/usage instructions, unimplemented changes,[4] attribution, dependency/environment changes, explanation of configuration, etc.

Summary for **Content**:

Nearly two-thirds (62.55%) of issues within this dimension concerns *Completeness*, while only about one-third (37.45%) concerns *Correctness*. Developers are most likely to 1) report wrong/broken links in RNs (19.12%), which annoyingly prevent them from accessing supplementary information, and 2) missing breaking changes (12.35%), which may mislead users and incur severe consequences after upgrading (e.g., crash).

4.2 Presentation

81 issues are related to **Presentation**, with two categories: *Readability* and *Usability*. Issues from the **Presentation** dimension help reveal how information should be organized, formatted, highlighted, visualized, and phrased in an RN, so that different RN users can make use of the RN for their purposes with maximum efficiency.

4.2.1 *Usability (56, 69.14%).*

This category refers to the degree to which users can use RNs to achieve their objectives effectively.

❖ More than half of the issues (31, 38.27%) in this category are related to *Poor Layout*, which means the changes are not clearly organized in RNs. Since different stakeholders may be interested in different kinds of information, RNs need to have a proper layout for them to quickly locate the information they want [97, 102]. On the other hand, poorly organized RNs may increase the time needed for users to grab valuable information, annoy readers [34], bury good features [80], and cause important changes to be missed by impacted users. For example, `Electron` lists two API deprecations under the "other" section in the v12.0.0 RN by mistake, which makes the deprecations easily overlooked [41]. Developers make the following suggestions in these issues for improving RN layout:

➢ Use markups (e.g., icons or emojis) to highlight breaking changes.
➢ Reorganize changes into a separate section if they are concerned by a specific audience, e.g., a separate section for database operators in the RNs of `CockroachDB` [61].
➢ Shorten RNs and fold a lengthy list of details, using detail/summary tags provided by GitHub Release Page [64], HTML, or Markdown features [34].

❖ Other issues arise from *Poor Formatting (25, 30.86%)*:

➢ *Typesetting (18, 22.22%)*: Most of these issues are caused by misusing syntax of markup languages (e.g., HTML) and usually lead to abnormal display, e.g., failing to display list due to missing HTML linebreaks [65].
➢ *Date Formatting (7, 8.64%)*: Some date format can cause ambiguities to people in different geographical regions [31].

4.2.2 *Readability (25, 30.86%).*

This category of issues concerns whether the RN is easy to read, including three subcategories: *Spelling Errors (14, 17.28%)*, *Grammar Errors (5, 6.17%)*, *Bad Writing Style (5, 6.17%)*, and *Multilingual Support Required (1, 1.23%)*. Although fixing grammar errors and spelling errors are easy, they may be hard to notice, especially for technical terms (e.g., MACs and Macs [45]). Also, certain writing style can make RNs clearer and more easily understandable, such as describing what happens after a bug is fixed instead of what used to happen [27]. One issue asks for multilingual support which helps more users understand RNs and enables product adaptation to a broader market [35].

Summary for **Presentation**:

The majority of issues (69.14%) within this dimension concerns *Usability*, especially poor layout, which may bury important information and lead to end users' misjudgement. Developers propose various solutions to alleviate this problem, including section reorganization (27.16%), folding (7.41%), and use of markups (3.70%). Other presentation issues concerns *Readability* (30.86%), such as spelling, writing style, etc.

4.3 Accessibility

173 issues are related to **Accessibility**, i.e., how to make RNs accessible for a broad audience, with three categories: *Limited Exposure*, *Wrong/Broken Link to RNs*, and *Lack Notification*. Issues in this dimension thus reveal how a software project should distribute their RNs, maintain links, and notify their users.

4.3.1 *Limited Exposure (115, 66.47%).*

Issues under this subcategory express either difficulty in finding RNs or expectation of more available ways to access RNs. The former case happens when RNs are placed in obscure locations, e.g., files with an unconventional name or in a deeply nested directory [63]. In the latter case, users suggest various locations to show RNs, e.g., *Can we get a page which explains the features and the release numbers in each of the releases of Teams Clients? If you have one, we can not find it* [18].

4.3.2 *Wrong/Broken Link to RNs (36, 20.81%).*

Missing Links and *Wrong/Broken Links* under the **Content** dimension describe cases where links in RNs are broken or wrong (see Section 4.1.1 and Section 4.1.2). In addition, many issues report that links to RNs themselves (in project website, etc.) may be broken. For example:

(1) *The section of the front page of this repo "Links to release notes" is full of dead links* [29].
(2) *Links to the Agent release notes from the APM docs left nav are returning 404s* [69].

[4] In this case, internal developers prepare a RN in advance as a task list for the next version but they fail to implement all tasks and forget to remove unfinished tasks from the final RN.

4.3.3 Lack Notification (22, 12.72%). The concerns expressed by these issues are twofold: 1) whether a certain medium should be adopted to notify users and publicize RNs, and 2) whether current ways of notification should be improved. For example, several issues mention the use of RSS feeds to notify new releases. In another case, a user complains: *Currently, the release notes for an updated version only show after the new version is installed. Basically, it is preferable to know in advance what changes are made to the app before its downloaded and installed* [17].

For improving RN accessibility, developers in our studied issues suggest the following locations to put RNs:

➢ *GitHub Release Pages*: GitHub provides a dedicated page to display the release history. Many issues show that developers often check GitHub Release Pages first when searching for RNs because they consider GitHub Release Pages as the most intuitive location for releases and RNs. As stated by a developer: *From an engineering point of view, having release notes published on GitHub is ideal, this is our source of truth* [33].

➢ *Project Websites*: Developers also expect project websites as RN management centers providing links to RNs for each release. Developers also suggest using a specific URL for the latest RN [37].

➢ *Files in Repositories*: Some developers think that a RN file in the repository (the root directory or the `doc/` folder) is more important than storing RNs on GitHub Release Page [30]. A RN file in repository makes the repository more self-contained (not depending on GitHub) and allows the usage of collaborative editing tools like Git [72]. The OpenStack community also requires that its projects must include RN files to record version changes and believe that this way can work on multiple patches simultaneously and reduce merge conflicts [93].

➢ *Apps*: Application software can provide buttons and links to access the latest RN, e.g., an 'about' button [72]. RN notifications can also be displayed when a new version is released [46].

➢ *Instant Messaging Channels*: Such channels can be used to immediately deliver new RNs to subscribed end users, e.g., Slack [50] and Telegram [56].

Summary for **Accessibility**:

Users encounter a diverse range of difficulties in accessing RNs, including *Limited Exposure* (66.47%), *Wrong/Broken Links to RNs* (20.81%), and *Lack of Notification* (12.72%).

4.4 Production

475 issues fall into the **Production** dimension, i.e., in what way RNs should be produced. Problems behind these issues can shed light on prospective automation approaches, improvement of existing tools, and design of better release processes. This dimension consists of three categories: *Automation*, *Planning*, and *Regulation*.

4.4.1 Automation (217, 45.68%). This category reflects four kinds of issues that developers frequently encounter on automated RN generation: *Request for Automation (118, 24.84%)*, *Request for Enhancement (37, 7.79%)*, *Error Induced by Automation (36, 7.58%)*, and *Improper Tool Configuration (26, 5.47%)*.

❖ *Request for Automation (118, 24.84%)*: More than half of issues in the *Automation* category are opened for discussing whether some sort of automation should be used and what specific tools to adopt for managing and generating RNs. As stated by a developer from `spid-compliant-certificates-python`: *Despite important, writing release notes is a very boring task...It would be nice to have them automatically generated every time a PR is merged* [7]. Developers propose two main types of solutions for automation: 1) writing project-specific scripts to fill predefined templates and publish releases (e.g., [8]), and 2) adopting existing automated tools such as Semantic Release [95], git-changelog-generator [14], Release It [91], and Release Drafter [90]. Although RN automation is a huge help in reducing manual toil, some developers express their concerns for full automation in their projects. They think that an automated workflow: 1) requires prefixes or labels for classifying commits or PRs, which may burden the code review and CI complexity; 2) is not suitable for important versions, e.g., stable releases, which need manual editing for better readability.

❖ *Request for Enhancement (37, 7.79%)*: These issues reveal suggested improvement of automated generation tools and scripts by users. Specifically, users mostly request for three kinds of support: 1) automated generation of RNs for different branches [70]; 2) automated retrieval of related information from multiple repositories [23]; 3) automated supplement of details, e.g., CVEs [24], attribution [73], and PR comments [40]. There are also some specific needs from various scenarios. For example, a member of `CockroachDB` proposes an extension to the RN extraction script to support the amendment of past RNs with new commits [94]. Another issue opened by a contributor of `chef/automate`, reveals the limited support of combining multiple RNs when upgrading across multiple versions and asks for further improvement [82]. Some developers suggest current tools to also add support to automate RN publishing.

❖ *Errors Induced by Automation (36, 7.58%)*: These issues report defects of automated tools and scripts, which are diverse and largely tool/project-specific. There are two types of issues: one is that these defects leads to unexpected RNs, e.g, wrong/missing content [60], repetition [88], and incorrect positions [68]; the other one is that these defects affect the generation process failure, e.g., not generating RNs that exceed certain length [6, 62]. Among these issues, most of them are caused by defects of current tools rather than project-specific scripts. Besides, these tools all have to fetch changes history from Git and several issues are caused by its complex mechanisms, such as branch control [48], rebase [49] and release tag [32], which developers need to pay more attention to in design.

❖ *Improper Tool Configuration (26, 5.47%)*: These issues usually arise from unfamiliarity with the tools, such as generating RNs without a template or by a wrong template. Most of these issues are caused by misconfigured change scopes, e.g., the expected branch, version ranges [43], certain types of changes [16], and triggered conditions [86]. Besides, parameter misconfiguration is another common cause, including the construction of paths [79], repository names [85], and environment variables [42], etc.

4.4.2 Planning (191, 40.21%). This category of issues has four subcategories: *When to Produce (103, 21.68%)*, *Whether to Produce (58, 12.21%)*, *Where to Produce (29, 6.11%)*, and *Who to Produce (1, 0.21%)*.

❖ *When to Produce (103, 21.68%)*: Developers discuss two kinds of issues in this subcategory, *Absence* and *Deadline Required*.

➢ In the former case of *Absence (93, 19.58%)*, projects do not provide RNs for all releases (i.e., some releases are missing RNs), which causes their users to open inquiry issues. For example, the absence of RN for `Recoil` confuses a user who says: *I saw that version 0.1.3 has been published on npm, but I cannot find release notes anywhere, would be good to know about potential breaking changes, deprecations and new additions* [77]. Besides, although most projects provide RNs for every version, some of them are released too late to be helpful which disappoints users [58].

➢ In the latter case of *Deadline Required (10, 2.11%)*, developers discuss how to produce RNs timely, e.g., update RNs before a new version is released [25], give a deadline for the RN [54], and announce the adoption of a formal release cycle [81].

❖ *Whether to Produce (58, 12.21%)*: This subcategory discusses the necessity of providing RNs. Some projects never provide RNs for informing changes in the new release. Consequently, in some cases, users open issues because the lack of RNs directly leads to upgrade failures and frustration [39]. They have to resort to various effort-prone methods to figure out changes from commit history, e.g., using `git diff` to show all code changes between two versions [71]. Although `git log` can list all commit messages and ease the pain of figuring out changes to some extent, as stated by a member of `Common Workflow Language`, *This requires everyone to write the best possible git commit message and have very clean git histories. While people are capable of this, it is more work for contributors* [52]. In other cases, internal developers open such issues as they notice RNs *would help developers to precisely see what notable changes have been made between each release of the project* [52]. However, not everyone agrees with providing a RN with each release, because they think the changes are only internal or too minor to be worth mentioning [59, 74]. Other project maintainers acknowledge the necessity of RNs but they lack time for them [55].

❖ Different from **Accessibility** issues, issues in the *Where to Produce (29, 6.11%)* subcategory concern where to collaboratively edit and store RN files. Although GitHub provides convenient release functionalities [3] to help developers manage RNs, it currently does not support collaborative RN editing. By contrast, many projects with a large team wish to distribute RN workload among team members so that RNs can be scalably produced. As a result, most projects opt for adding RNs as files in the git repository so that multiple developers can be involved in RN production (e.g., [30]).

❖ One special case mentions the lack of accountability in RN production and suggests someone should be responsible for it [57].

4.4.3 Regulation (67, 14.11%).

This category of issues refers to what regulations should be followed to simplify and ease the production of RNs. It covers three subcategories: *PR/Issue/Commit Management (40, 8.42%)*, *Inconsistency (16, 3.37%)*, and *Workflow (11, 2.32%)*.

❖ *PR/Issue/Commit Management (40, 8.42%)*: This subcategory refers to issues discussing how to efficiently prepare (relevant) PRs, issues, and commits for RNs. This procedure is usually time-consuming, especially for large projects. For example, a member from `pytorch/vision` complains that *I wrote the release notes last week and we spent the vast majority of the time labeling the PRs* and suggests *it'd be good to have a process that would make this faster* [53]. Some solutions emerges from the discussions in these issues. For commits, developers prefer to adopt a convention for writing *structured commit messages* (e.g., conventional commit rule [9]), so that changes (e.g, features, fixes, and breaking changes) in a commit can be documented in a machine-parsable way. For PRs, several large projects recommend to label each PR with pre-defined labels. In the case of `pytorch/vision`, developers reach a consensus on categorizing each PR with GitHub labels describing affected components and changed types (e.g., breaking changes and improvements) [53]. For issues, many developers mention the use of GitHub milestone [2] for progress tracking. Some projects create each milestone using version numbers and group issues into milestones [78], which reduces the scope of review when developers write RNs.

❖ *Inconsistency (16, 3.37%)*: This subcategory refers to issues about inconsistencies between 1) RNs published in different places, 2) RNs and other documentation within project, and 3) RNs and documentation in other projects. As revealed in the *Accessibility* dimension, RNs are usually published in different places including, GitHub Release Page, project homepage, etc. However, developers sometimes neglect to maintain their consistency. For example, a user suggests that *It'd be great to have a way to sync release notes in* docs.newrelic.com *by fetching the information from GitHub* [33]. RNs can also easily become inconsistent with other documentation within project, e.g., usage guides and READMEs. As an example of inconsistency between RNs and usage guides, a user complains that *our documentation is horribly outdated* and calls for internal developers to *go through all release notes and move all information that is not outdated and is missing from the documentation to the usage guide* [38]. Finally, RNs sometimes need to include changes or attribution information from closely related projects, which requires collaboration of developers from the related projects.

❖ *Workflow (11, 2.32%)*: This subcategory refers to issues discussing formulation of RN production workflow or improvement on existing workflow. Among the issues, nine are opened as a *Request for a Workflow*. For example, a developer from `mantid/mantidimaging` formulates a workflow as follows: *Release notes should be continuously updated during development. Almost all pull requests should have an update to the relevant file and section in* `docs/release_notes`. *If the next release name is not yet chosen, this will be* `next.rts`*, and renamed closer to release. When fixes are backported to a release branch, they can be added to the notes for that release, in an updates section* [87]. One issue discusses *File Naming* and suggests avoid confusing RN naming format [20]. Another issue discusses *Repository Permission Control* where developers in `kubernetes/test-infra` request to have the permission to collaboratively edit RNs. This project only allows very few members to have write access to RNs, causing others to *ping someone with write access or the author of the parent PR to add the release note to the PR body* [76].

Summary for Production:

Developers show a strong interest in *Automation* (45.68% of issues within this dimension), but automated tools/scripts may lack desired features, tend to induce errors, and are hard or error-prone to configure. Additionally, without proper *Planning* (40.21%), e.g., release schedules and deadlines, users may be confused about the absence of RNs. Finally, *Regulation* (14.11%) of RN production, especially conventions for pull requests (PRs), issues, and commits (8.42%), is vital for enabling efficient RN production in large software projects.

5 DISCUSSION

5.1 Implications

5.1.1 Comparison with Previous Work. Since previous works categorize RN content into different taxonomies [102, 117], it is not easy to perform detailed comparison of our results with theirs (mapping results from different work can be a possibility for future studies). We can still observe some interesting differences by summarizing most frequently occurring RN content in different taxonomies in Table 3: 1) breaking changes and links are more likely to have issues but they are not listed as a major category in previous taxonomies; 2) new features and bug fixes are not likely to have issues even if they occur most frequently in previous taxonomies; 3) some information frequently desired by users are not mentioned in previous work, such as migration/usage instructions, code examples, and dependency specifications. Our lens of observation sheds light on the most fragile parts of RNs untouched in previous taxonomies.

The taxonomy in our paper also extends the work of Bi et al. [102] with a significant amount of new empirical evidence and actionable implications. For example, they find in RQ2.2 that clear structured and the writing styles of release note documentation are vital. We go one step further and identify concrete evidence on how structure and style impact users, which we further derive into actionable advice on how to write and organize RNs.

5.1.2 A Checklist for RN Production. Based on the results summarized in Section 4, we provide a checklist as follows.

☑ *What Should be Included in RNs?* We find that issues related to RN **Content** (Section 4.1) have different distribution compared with most frequent RN content identified in previous works [97, 102, 117], which indicates that some types of information are more likely to be missed or incorrect than others. Therefore, we recommend RN producer to check whether the following eight kinds of *changes* have been described in RNs: 1) Breaking Changes, 2) New Features, 3) Enhancements, 4) Fixed Bugs, 5) Documentation Changes, 6) Dependency/environment Changes, 7) Security Changes, and 8) License Changes. We also find that additional information that benefits better understanding and tracking of these changes, e.g., links to corresponding PRs/issues/commits, is preferred by users. We therefore recommend including, where necessary, the following eight kinds of *explanatory information* in RNs: 1) Links to Change-Related PRs, Issues, and Commits, 2) Guides (e.g., upgrade, migration, or setup guides), 3) Code Examples, 4) Dependency/environment Specification, 5) Attributions (e.g., authors, reviewers, commenters, etc.), 6) Explanation for Jargon-Heavy Descriptions, 7) Versioning Information (e.g., release time, version name/number, setup package, etc), and 8) Known Issues.

☑ *How to Ensure RNs' Completeness?* Our results from Section 4.1 show that RNs are more frequently affected by *Completeness* issues than *Correctness* ones, e.g., missing breaking changes, which indicates the importance of ensuring completeness in RN production. Thus, applying completeness checks on RNs, i.e., making sure all critical changes are listed, is strongly recommended. However, our investigations reveal two main reasons for completeness issues: 1) lacking manpower or time to conduct thorough inspections; 2) difficult for a limited number of developers to understand all changes between versions. While automated tools for checking RN completeness is still lacking, we locate several practices suggested by issue participants that may make RNs more likely to be complete and reduce the pressure to review changes:

➢ For each change description in RNs, add links to the corresponding PR, issue, commit, or external resources (e.g., CVE) so that its completeness can be easily checked.
➢ Adopt a systematic and structured way to label and organize changes (i.e., PRs/commits/issues), as discussed in Section 4.4.3.
➢ Distribute workload among all contributors instead of having a central responsible person for creating RNs. For example, some projects require that each PR should contain a release notes section in the PR body that describes the affected submodule name and a list of changes for that submodule [96].

☑ *How Should RNs be Organized?* The issues related to *Presentation* indicate that layout indeed greatly influences RN reading experience, as mentioned by Bi et al. [102]. An analogy is the relationship between content and directory: if the content is misplaced or not indexed, it is easy to miss the content you are interested in [41, 80]. From these issues and their related RNs, we find that several "hierarchical structure" can be used to separate changes into categories and better organize RNs. Based on results in Section 4.2.1, we recommend two strategies to group changes: 1) by type of change (e.g., new features, fixed bugs, breaking changes); 2) by affected component (e.g., the network module). The two strategies can be combined (e.g., first by component and then by type of change). We also recommend to highlight most important changes (e.g., breaking changes, major new features) on top. After an organization is determined, we further recommend to use proper visualization and fold lengthy lists for highlighting important changes.

☑ *How to Choose Writing Style for RNs?* When investigating issues under *Bad Writing Style*, a case attracts our attention, i.e., *RN should be funny and cryptic in app stores to attract non-technical end users but concise and clear on GitHub to deliver information efficiently*. Because the requirements of users differ from these of internal developers, we recommend projects to provide different RNs in different writing styles to serve different audiences (stakeholders). For example, Apache Camel provides two types of RNs: one is more generalized and summarized [5] intended for the end users, while the other is a list of all issues that have been resolved under this update intended for someone who needs technical details [89].

☑ *How to Make RNs (More) Accessible?* This problem involves not only how users can access RN quickly (*Limited Exposure* and *Lack Notification*), but also where producers should collaboratively edit and store RNs (*Where to Produce*). It can be relieved through some more diverse ways for notification and access. As summarized in Section 4.3, we recommend developers to consider publicizing RNs in the following locations, if applicable, to make their RNs more accessible: GitHub Release Pages, Project Websites, Files in Repositories, Apps, and Instant Messaging Channels.

☑ *Link Check.* We find many issues related to links, e.g., *Missing Links* and *Wrong Links* under **Content**, and *Wrong/Broken Links to RNs* under **Accessibility**. Broken or wrong links often make users unpleasant and increase their cost of searching. A developer from `mantidproject/mantid` mentions that they need to go over release and check links work before releasing [47]. Checking invalid links regularly in RN can mitigate this problem, that can be achieved by some tools, e.g., Xenu Link Sleuth [11] and HTML Link Validator [15]. Besides, providing absolute path instead of relative

Table 3: Comparison of Most Frequent RN Content in Different Taxonomies.

Moreno et al. [117]	Bi et al. [102]	Ours (Completeness)*	Ours (Correctness)*
Fixed Bugs (90%)	Issues Fixed (79.3%)	Breaking Changes (22.93%)	Links (51.06%)
New Features (46%)	New Features (55.1%)	New Features (14.01%)	Version Information (14.89%)
New Code Components (43%)	System Internal Changes (25.1%)	Links (11.46%)	Dependency Specifications (7.45%)
Modified Features (26%)	Non-functional Requirements (10.3%)	Dependency Specifications (8.92%)	Identifiers (7.45%)
Refactoring Operations (21%)	Documentation Updates (9.5%)	Migration/Usage Instructions (8.92%)	Code Examples (5.32%)

* The percentages here are different from Figure 1 because the denominators are the total number of issues in the Completeness (157 issues) and the Correctness (94 issues) category, respectively. In the Completeness column, issues from *Missing* and *Insufficient* are merged.

path [83] can reduce potential broken risks, no matter in RNs or in other documents containing links to RNs, e.g., READMEs.

5.1.3 Automating RN Production. Apart from the tool-specific problems in Section 4.4, we further summarize the following research directions that may greatly help automated RN generation:

Automated Labeling of Software Changes: Our results in Section 4.4.1 show that many developers request for tools to automate RN production. However, to the best of our knowledge, existing tools have strong constraints on input. Some well-known tools, e.g., github-activity and Release Drafter, require a compatible PR label system. Semantic-Release requires developers to write commit messages following a specific rule, i.e., Angular Commit Message Conventions, requiring developers to specify which category a commit belongs to manually. These preconditions limit their application scope, and the whole project needs to change its production process to adapt to it [84]. Techniques for automated commit/PR classification, which we consider as a promising direction, can alleviate this problem. Existing commit classification methods (e.g., [106, 115]) mainly focus on classifying commits into three maintenance categories (i.e., corrective, adaptive, and perfective) proposed by Swanson [122], which is not suitable for RN generation. Therefore, classifying commits into categories suitable for RN generation (e.g., the eight kinds of *changes* proposed in Section 5.1.2) is needed to facilitate automated RN generation. Similar discrepancies also exist for works on PR classification [112, 128].

Automated Summarization and Language Style Unification: As reflected in Section 4.2, a fluent and unified writing style is vital to RN *Readability*. However, existing tools generate RNs by integrating existing text, e.g., PR titles and commit messages, which not only violates RNs' fundamental principle (*it should focus on the impact for the user and make that understandable* [92]), but also offloads the quality responsibility to developers writing other development text. This often leads to poor readability of the final generated RNs. With advances in natural language preprocessing (NLP) tasks like text summarization [105] and style transfer [125], it will be interesting to explore approaches that summarize existing development text and unify language style for automated RN generation.

Automated Testing of RNs: As shown in Section 4.1, *Completeness* and *Correctness* are the key to a high quality RN. Although we synthesize a checklist of practices during the process of RN production, these largely manual practices are hardly a strong guarantee for reducing the risk of incompleteness or incorrectness. To the best of our knowledge, there is still no tool designed for *testing* (i.e., inconsistency checking) of RNs. Challenges for facilitating such testing may include: 1) checking the consistency between natural language description and software changes; and 2) checking the consistency between documentation from different sources (e.g., RNs and usage guides, Section 4.4.3). Similar works for, e.g., checking code comment inconsistency [123, 129], may be a good starting port for exploring the possibility of such a tool. Furthermore, since users perceive breaking changes as important but frequently missing in RNs (Section 4.1.1), works on breaking change and update incompatibility detection [114] should also be important.

5.2 Threats to Validity

5.2.1 Internal Validity. Our taxonomy construction is based entirely on manual analysis, which may introduce subjectivity and labeling errors. To mitigate these threats, we include two inspectors and one arbitrator into the process, all with rich development experience. To ensure the quality of taxonomy, we conduct multiple iterative rounds to refine the taxonomy and incorporates feedback from real developers. We also measure inter-rater reliability to ensure that the taxonomy is precisely defined and reproducible.

5.2.2 External Validity. Our work only uses issues from GitHub projects for categorizing RN issues, which means that our results may not be generalized to another context (e.g., industry projects). Since GitHub is a huge and diverse coding platform and the projects involved in our analysis are of high quality, we believe our results reveal valuable insights and practical challenges in RN production and usage. To further confirm our belief, we invite three industry developers to validate whether our taxonomy can cover the RN issues they have encountered. However, the limited number of developers also poses a threat, which we find it hard to mitigate because it is not easy to locate industry developers experienced with RNs. Future work may be able to gain different insights through other data sources or interviews/surveys at a larger scale.

Another threat to external validity comes from using only issues with keyword "release note" in their titles. Many issues may still discuss RNs even if they do not have the keyword in their titles. The threat can be mitigated by the size of our dataset that is comparable to and even larger than existing studies [99, 101, 103, 124, 130].

6 CONCLUSION

In this paper, we have presented a taxonomy of real-world RN issues summarized from GitHub. Our taxonomy not only distills to a practitioner-oriented checklist for release note production, but also lays out an empirical foundation for several interesting research directions for release note automation. As future work, we plan to investigate such opportunities for integrating novel automation approaches with existing release note workflows.

Acknowledgments. This work is supported by the National Key R&D Program of China Grant 2018YFB1004201 and the National Natural Science Foundation of China Grant 61825201.

REFERENCES

[1] 2021. About issues - GitHub Docs. https://docs.github.com/en/issues/tracking-your-work-with-issues/about-issues.
[2] 2021. About milestones - GitHub Docs. https://docs.github.com/en/issues/using-labels-and-milestones-to-track-work/about-milestones.
[3] 2021. About releases - GitHub Docs. https://docs.github.com/en/repositories/releasing-projects-on-github/about-releases.
[4] 2021. angular.js/DEVELOPERS.md at master · angular/angular.js. https://github.com/angular/angular.js/blob/master/DEVELOPERS.md#commits.
[5] 2021. Apache Camel 3.11 What's New - Apache Camel. https://camel.apache.org/blog/2021/06/Camel311-Whatsnew/.
[6] 2021. AppCenterDistributeV3 should truncate release notes. https://github.com/microsoft/azure-pipelines-tasks/issues/11922.
[7] 2021. Auto generate release notes. https://github.com/italia/spid-compliant-certificates-python/issues/6.
[8] 2021. Automate release notes. https://github.com/eclipse/rdf4j/issues/2784.
[9] 2021. Conventional Commits. https://www.conventionalcommits.org/.
[10] 2021. Convergence Q&A: The Answer is in Black and White | Security Info Watch. https://www.securityinfowatch.com/cybersecurity/article/10840073/the-importance-of-release-notes.
[11] 2021. Find broken links on your site with Xenu's Link Sleuth (TM). https://home.snafu.de/tilman/xenulink.html.
[12] 2021. The Firefox release notes process - MozillaWiki. https://wiki.mozilla.org/Release_Management/Release_Notes.
[13] 2021. GH Archive. https://www.gharchive.org/.
[14] 2021. github-changelog-generator. https://github.com/github-changelog-generator/github-changelog-generator.
[15] 2021. HTML Link Validator - Download. https://html-link-validator.en.softonic.com/.
[16] 2021. Issue #103 of PSBicep/PSBicep. https://github.com/PSBicep/PSBicep/issues/103.
[17] 2021. Issue #1097 of sublimehq/sublime_merge. https://github.com/sublimehq/sublime_merge/issues/1097.
[18] 2021. Issue #1123 of dotnet/SqlClient. https://github.com/dotnet/SqlClient/issues/1123.
[19] 2021. Issue #11517 of keras-team/keras. https://github.com/keras-team/keras/issues/11517.
[20] 2021. Issue #1255 of newrelic/docs-website. https://github.com/newrelic/docs-website/issues/1255.
[21] 2021. Issue #1271 of cesanta/mongoose. https://github.com/cesanta/mongoose/issues/1271.
[22] 2021. Issue #12961 of babel/babel. https://github.com/babel/babel/issues/12961.
[23] 2021. Issue #131 of rfennell/ReleaseNotesAction. https://github.com/rfennell/ReleaseNotesAction/issues/131.
[24] 2021. Issue #1354 of kubernetes/release. https://github.com/kubernetes/release/issues/1354.
[25] 2021. Issue #1682 of microsoft/ApplicationInsights-Java. https://github.com/microsoft/ApplicationInsights-Java/issues/1682.
[26] 2021. Issue #17977 of numpy/numpy. https://github.com/numpy/numpy/issues/17977.
[27] 2021. Issue #18 of hazelcast/cloud-docs. https://github.com/hazelcast/cloud-docs/issues/18.
[28] 2021. Issue #1873 of Azure/azure-sdk. https://github.com/Azure/azure-sdk/issues/1873.
[29] 2021. Issue #194 of HOKGroup/HOK-Revit-Addins. https://github.com/HOKGroup/HOK-Revit-Addins/issues/194.
[30] 2021. Issue #1966 of decred/dcrwallet. https://github.com/decred/dcrwallet/issues/1966.
[31] 2021. Issue #1990 of bigcommerce/cornerstone. https://github.com/bigcommerce/cornerstone/issues/1990.
[32] 2021. Issue #2000 of DataDog/dd-trace-py. https://github.com/DataDog/dd-trace-py/issues/2000.
[33] 2021. Issue #2127 of newrelic/docs-website. https://github.com/newrelic/docs-website/issues/2127.
[34] 2021. Issue #2178 of vaadin/platform. https://github.com/vaadin/platform/issues/2178.
[35] 2021. Issue #2207 of microsoft/appcenter. https://github.com/microsoft/appcenter/issues/2207.
[36] 2021. Issue #2410 of NuGet/docs.microsoft.com-nuget. https://github.com/NuGet/docs.microsoft.com-nuget/issues/2410.
[37] 2021. Issue #2502 of 3drepo/3drepo.io. https://github.com/3drepo/3drepo.io/issues/2502.
[38] 2021. Issue #2527 of rotki/rotki. https://github.com/rotki/rotki/issues/2527.
[39] 2021. Issue #260 of getsentry/sentry-laravel. https://github.com/getsentry/sentry-laravel/issues/260.
[40] 2021. Issue #27 of stephend017/snake-charmer. https://github.com/stephend017/snake-charmer/issues/27.
[41] 2021. Issue #28375 of electron/electron. https://github.com/electron/electron/issues/28375.
[42] 2021. Issue #29 of waifu-motivator/wmp-env-action. https://github.com/waifu-motivator/wmp-env-action/issues/29.
[43] 2021. Issue #2907 of meshery/meshery. https://github.com/meshery/meshery/issues/2907.
[44] 2021. Issue #3 of cwarwicker/moodle-tool_ribbons. https://github.com/cwarwicker/moodle-tool_ribbons/issues/3.
[45] 2021. Issue #300 of MicrosoftDocs/OfficeDocs-OfficeUpdates. https://github.com/MicrosoftDocs/OfficeDocs-OfficeUpdates/issues/300.
[46] 2021. Issue #302 of chef/chef-workstation-app. https://github.com/chef/chef-workstation-app/issues/302.
[47] 2021. Issue #31371 of mantidproject/mantid. https://github.com/mantidproject/mantid/issues/31371.
[48] 2021. Issue #3145 of BlueWallet/BlueWallet. https://github.com/BlueWallet/BlueWallet/issues/3145.
[49] 2021. Issue #31816 of istio/istio. https://github.com/istio/istio/issues/31816.
[50] 2021. Issue #3193 of cdr/code-server. https://github.com/cdr/code-server/issues/3193.
[51] 2021. Issue #3238 of nushell/nushell. https://github.com/nushell/nushell/issues/3238.
[52] 2021. Issue #328 of common-workflow-language/cwlviewer. https://github.com/common-workflow-language/cwlviewer/issues/328.
[53] 2021. Issue #3351 of pytorch/vision. https://github.com/pytorch/vision/issues/3351.
[54] 2021. Issue #376 of USAJOBS-temp/openoppstasks. https://github.com/USAJOBS-temp/openoppstasks/issues/376.
[55] 2021. Issue #432 of negomi/react-burger-menu. https://github.com/negomi/react-burger-menu/issues/432.
[56] 2021. Issue #48 of wabarc/wayback. https://github.com/wabarc/wayback/issues/48.
[57] 2021. Issue #5236 of wellcomecollection/wellcomecollection.org. https://github.com/wellcomecollection/wellcomecollection.org/issues/5236.
[58] 2021. Issue #533 of hashicorp/terraform-ls. https://github.com/hashicorp/terraform-ls/issues/533.
[59] 2021. Issue #544 of dask/fastparquet. https://github.com/dask/fastparquet/issues/544.
[60] 2021. Issue #54752 of saltstack/salt. https://github.com/saltstack/salt/issues/54752.
[61] 2021. Issue #57898 of cockroachdb/cockroach. https://github.com/cockroachdb/cockroach/issues/57898.
[62] 2021. Issue #581 of digidem/mapeo-mobile. https://github.com/digidem/mapeo-mobile/issues/581.
[63] 2021. Issue #59 of hedgedoc/hedgedoc.github.io. https://github.com/hedgedoc/hedgedoc.github.io/issues/59.
[64] 2021. Issue #5913 of prisma/prisma. https://github.com/prisma/prisma/issues/5913/#issuecomment-788326709.
[65] 2021. Issue #64 of nathanwoulfe/Plumber-2. https://github.com/nathanwoulfe/Plumber-2/issues/64.
[66] 2021. Issue #663 from vue-leaflet/Vue2Leaflet. https://github.com/vue-leaflet/Vue2Leaflet/issues/663.
[67] 2021. Issue #7058 of coq/coq. https://github.com/coq/coq/issues/7058/#issuecomment-375720879.
[68] 2021. Issue #714 of gitpod-io/gitpod. https://github.com/gitpod-io/gitpod/issues/714.
[69] 2021. Issue #758 of newrelic/docs-website. https://github.com/newrelic/docs-website/issues/758.
[70] 2021. Issue #789 of opensearch-project/OpenSearch. https://github.com/opensearch-project/OpenSearch/issues/789.
[71] 2021. Issue #79 of rayokota/kcache. https://github.com/rayokota/kcache/issues/79.
[72] 2021. Issue #808 of alteryx/woodwork. https://github.com/alteryx/woodwork/issues/808.
[73] 2021. Issue #8605 of renovatebot/renovate. https://github.com/renovatebot/renovate/issues/8605.
[74] 2021. Issue #87 of dtolnay/semver. https://github.com/dtolnay/semver/issues/87.
[75] 2021. Issue #882 of MicrosoftEdge/WebView2Feedback. https://github.com/MicrosoftEdge/WebView2Feedback/issues/882.
[76] 2021. Issue #9098 of kubernetes/test-infra. https://github.com/kubernetes/test-infra/issues/9098.
[77] 2021. Issue #916 of facebookexperimental/Recoil. https://github.com/facebookexperimental/Recoil/issues/916.
[78] 2021. Issue #916 of kubernetes-sigs/multi-tenancy. https://github.com/kubernetes-sigs/multi-tenancy/issues/916.
[79] 2021. Issue #99 of zammad/zammad-helm. https://github.com/zammad/zammad-helm/issues/99.).
[80] 2021. Issue #9903 of EOSIO/eos. https://github.com/EOSIO/eos/issues/9903.

[81] 2021. Move to a formal release cycle with release notes. https://github.com/girder/cookiecutter-girder-4/issues/45.
[82] 2021. Provide release note information in a format that supports users not upgrading from version N to version N+1. https://github.com/chef/automate/issues/2141.
[83] 2021. Pull Request #1114 of hedgedoc/hedgedoc. https://github.com/hedgedoc/hedgedoc/pull/1114/files.
[84] 2021. Pull Request #1164 of opentelekomcloud/terraform-provider-opentelekomcloud. https://github.com/opentelekomcloud/terraform-provider-opentelekomcloud/pull/1164.
[85] 2021. Pull Request #279 of corgibytes/freshli-lib. https://github.com/corgibytes/freshli-lib/pull/279/files.
[86] 2021. Pull Request #301 of corgibytes/freshli-lib. https://github.com/corgibytes/freshli-lib/pull/301/files.
[87] 2021. Pull Request #798 of mantidproject/mantidimaging. https://github.com/mantidproject/mantidimaging/pull/798/files.
[88] 2021. Pull Request #83 of gohugoio/hugo. https://github.com/gohugoio/hugo/pull/83.
[89] 2021. Release 3.11.0 - Apache Camel. https://camel.apache.org/releases/release-3.11.0/.
[90] 2021. Release-Drafter. https://github.com/release-drafter/release-drafter.
[91] 2021. Release It! https://github.com/release-it/release-it.
[92] 2021. Release Management—OpenStack Project Team Guide Documentation. https://docs.openstack.org/project-team-guide/release-management.html#how-to-add-new-release-notes.
[93] 2021. reno: A New Way to Manage Release Notes — reno 3.4.1.dev1 documentation. https://docs.openstack.org/reno/latest/.
[94] 2021. scripts/release-notes: make it possible to edit a release note in a different commit. https://github.com/cockroachdb/cockroach/issues/42163.
[95] 2021. Semantic Release. https://github.com/semantic-release/semantic-release.
[96] 2021. Writing Release Notes ·sympy/sympy Wiki. https://github.com/sympy/sympy/wiki/Writing-Release-Notes.
[97] Surafel Lemma Abebe, Nasir Ali, and Ahmed E Hassan. 2016. An empirical study of software release notes. *Empirical Software Engineering* 21, 3 (2016), 1107–1142. https://doi.org/10.1007/s10664-015-9377-5
[98] Emad Aghajani, Csaba Nagy, Mario Linares-Vásquez, Laura Moreno, Gabriele Bavota, Michele Lanza, and David C Shepherd. 2020. Software documentation: the practitioners' perspective. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*. IEEE, 590–601. https://doi.org/10.1145/3377811.3380405
[99] Emad Aghajani, Csaba Nagy, Olga Lucero Vega-Márquez, Mario Linares-Vásquez, Laura Moreno, Gabriele Bavota, and Michele Lanza. 2019. Software documentation issues unveiled. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 1199–1210. https://doi.org/10.1109/ICSE.2019.00122
[100] Abdulkareem Alali, Huzefa Kagdi, and Jonathan I Maletic. 2008. What's a typical commit? A characterization of open source software repositories. In *2008 16th IEEE International Conference on Program Comprehension*. IEEE, 182–191. https://doi.org/10.1109/ICPC.2008.24
[101] Stefanie Beyer, Christian Macho, Massimiliano Di Penta, and Martin Pinzger. 2018. Automatically classifying posts into question categories on Stack Overflow. In *2018 IEEE/ACM 26th International Conference on Program Comprehension (ICPC)*. IEEE, 211–21110. https://doi.org/10.1145/3196321.3196333
[102] Tingting Bi, Xin Xia, David Lo, John Grundy, and Thomas Zimmermann. 2020. An empirical study of release note production and usage in practice. *IEEE Transactions on Software Engineering* (2020). https://doi.org/10.1109/TSE.2020.3038881
[103] Zhenpeng Chen, Yanbin Cao, Yuanqiang Liu, Haoyu Wang, Tao Xie, and Xuanzhe Liu. 2020. A comprehensive study on challenges in deploying deep learning based software. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 750–762. https://doi.org/10.1145/3368089.3409759
[104] Roberta Coelho, Lucas Almeida, Georgios Gousios, and Arie Van Deursen. 2015. Unveiling exception handling bug hazards in android based on GitHub and Google code issues. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. IEEE, 134–145. https://doi.org/10.1109/MSR.2015.20
[105] Wafaa S El-Kassas, Cherif R Salama, Ahmed A Rafea, and Hoda K Mohamed. 2021. Automatic text summarization: A comprehensive survey. *Expert Systems with Applications* 165 (2021), 113679. https://doi.org/10.1016/j.eswa.2020.113679
[106] Lobna Ghadhab, Ilyes Jenhani, Mohamed Wiem Mkaouer, and Montassar Ben Messaoud. 2021. Augmenting commit classification by using fine-grained source code changes and a pre-trained deep neural language model. *Information and Software Technology* 135 (2021), 106566. https://doi.org/10.1016/j.infsof.2021.106566
[107] Hao He, Runzhi He, Haiqiao Gu, and Minghui Zhou. 2021. A large-scale empirical study on Java library migrations: prevalence, trends, and rationales. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 478–490.
[108] F W Holloway. 1985. Praxis release notes, Versions 7. 4 and 7. 5. (9 1985). https://www.osti.gov/biblio/5141606
[109] Siw Elisabeth Hove and Bente Anda. 2005. Experiences from conducting semi-structured interviews in empirical software engineering research. In *11th IEEE International Software Metrics Symposium (METRICS'05)*. IEEE, 10–pp.
[110] Nargiz Humbatova, Gunel Jahangirova, Gabriele Bavota, Vincenzo Riccio, Andrea Stocco, and Paolo Tonella. 2020. Taxonomy of real faults in deep learning systems. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. 1110–1121. https://doi.org/10.1145/3377811.3380395
[111] Huaxi Jiang, Jie Zhu, Li Yang, Geng Liang, and Chun Zuo. 2021. DeepRelease: Language-agnostic Release Notes Generation from Pull Requests of Open-source Software. In *2021 28th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 101–110. https://doi.org/10.1109/APSEC53868.2021.00018
[112] Jing Jiang, Qiudi Wu, Jin Cao, Xin Xia, and Li Zhang. 2021. Recommending tags for pull requests in GitHub. *Information and Software Technology* 129 (2021), 106394. https://doi.org/10.1016/j.infsof.2020.106394
[113] Sebastian Klepper, Stephan Krusche, and Bernd Bruegge. 2016. Semi-automatic generation of audience-specific release notes. In *2016 IEEE/ACM International Workshop on Continuous Software Evolution and Delivery (CSED)*. IEEE, 19–22.
[114] Patrick Lam, Jens Dietrich, and David J Pearce. 2020. Putting the semantics into semantic versioning. In *Proceedings of the 2020 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*. 157–179. https://doi.org/10.1145/3426428.3426922
[115] Stanislav Levin and Amiram Yehudai. 2017. Boosting Automatic Commit Classification Into Maintenance Activities By Utilizing Source Code Changes. Association for Computing Machinery, New York, NY, USA.
[116] Walid Maalej and Hans-Jörg Happel. 2010. Can development work describe itself?. In *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*. IEEE, 191–200. https://doi.org/10.1109/MSR.2010.5463344
[117] Laura Moreno, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, Andrian Marcus, and Gerardo Canfora. 2017. ARENA: An Approach for the Automated Generation of Release Notes. *IEEE Transactions on Software Engineering* 43, 2 (2017), 106–127. https://doi.org/10.1109/TSE.2016.2591536
[118] Sristy Sumana Nath and Banani Roy. 2021. Towards automatically generating release notes using extractive summarization technique. In *International Conference on Software Engineering & Knowledge Engineering, SEKE*. 241–248.
[119] Helena Holmström Olsson and Jan Bosch. 2014. From opinions to data-driven software R&D: A multi-case study on how to close the 'open loop' problem. In *2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications*. IEEE, 9–16. https://doi.org/10.1109/SEAA.2014.75
[120] Carolyn B. Seaman. 1999. Qualitative methods in empirical studies of software engineering. *IEEE Transactions on software engineering* 25, 4 (1999), 557–572.
[121] Emad Shihab, Akinori Ihara, Yasutaka Kamei, Walid M Ibrahim, Masao Ohira, Bram Adams, Ahmed E Hassan, and Ken-ichi Matsumoto. 2013. Studying re-opened bugs in open source software. *Empirical Software Engineering* 18, 5 (2013), 1005–1042. https://doi.org/10.1007/s10664-012-9228-6
[122] E. Burton Swanson. 1976. The Dimensions of Maintenance. In *Proceedings of the 2nd International Conference on Software Engineering* (San Francisco, California, USA) *(ICSE '76)*. IEEE Computer Society Press, Washington, DC, USA, 492–497.
[123] Lin Tan, Ding Yuan, Gopal Krishna, and Yuanyuan Zhou. 2007. /* iComment: Bugs or bad comments? */. In *Proceedings of 21st ACM SIGOPS Symposium on Operating Systems Principles*. 145–158. https://doi.org/10.1145/1294261.1294276
[124] Xin Tan and Minghui Zhou. 2019. How to communicate when submitting patches: An empirical study of the Linux kernel. *Proceedings of the ACM on Human-Computer Interaction* 3, CSCW (2019), 1–26.
[125] Martina Toshevska and Sonja Gievska. 2021. A Review of Text Style Transfer using Deep Learning. *IEEE Transactions on Artificial Intelligence* (2021), 1–1.
[126] Aidan ZH Yang, Safwat Hassan, Ying Zou, and Ahmed E Hassan. 2021. An Empirical Study on Release Notes Patterns of Popular Apps in the Google Play Store. *Empirical Software Engineering* (2021), 1–41.
[127] Liguo Yu. 2009. Mining change logs and release notes to understand software maintenance and evolution. *CLEI Electron Journal* 12, 2 (2009), 1–10.
[128] Song Yu, Li Xu, Yan Zhang, Jinsong Wu, Zhifang Liao, and Yanbing Li. 2018. NBSL: A Supervised Classification Model of Pull Request in Github. In *2018 IEEE International Conference on Communications (ICC)*. 1–6.
[129] Juan Zhai, Yu Shi, Minxue Pan, Guian Zhou, Yongxiang Liu, Chunrong Fang, Shiqing Ma, Lin Tan, and Xiangyu Zhang. 2020. C2S: translating natural language comments to formal program specifications. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 25–37.
[130] Tianyi Zhang, Cuiyun Gao, Lei Ma, Michael Lyu, and Miryung Kim. 2019. An empirical study of common challenges in developing deep learning applications. In *2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 104–115. https://doi.org/10.1109/ISSRE.2019.00020
[131] Yuxia Zhang, Minghui Zhou, Audris Mockus, and Zhi Jin. 2019. Companies' Participation in OSS development–An empirical study of OpenStack. *IEEE Transactions on Software Engineering* 47, 10 (2019), 2242–2259.