Noname manuscript No.
(will be inserted by the editor)

How to Get It Right? A Comprehensive Analysis of Challenges and Strategies for Software Release Notes on GitHub

Jianyu Wu · Hao He · Kai Gao · Wenxin Xiao · Jingyue Li · Minghui Zhou

Received: date / Accepted: date

Abstract Release notes (RNs) refer to the technical documentation that offers users, developers, and other stakeholders comprehensive information about the changes and updates of a new software version. Producing high-quality RNs can be challenging, and it remains unknown what issues developers commonly encounter and what effective strategies can be adopted to mitigate them. To bridge this knowledge gap, we conduct a manual analysis of 1,529 latest RN-related issues in the GitHub issue tracker by using multiple rounds of open coding and construct 1) a comprehensive taxonomy of RN-related issues with four dimensions validated through three semi-structured interviews; 2) an effective framework with eight categories of strategies to overcome these challenges. The four dimensions of RN-related issues revealed by the taxonomy and the corresponding strategies from the framework include: 1) *Content* (419, 25.47%): RN producers tend to overlook information rather than include inaccurate details, especially for breaking changes. To address this, effective completeness validations are recommended, such as managing Pull Requests, issues, and commits related to RNs; 2) *Presentation* (150, 9.12%): inadequate layout may bury important information and lead to end users' confusion, which can be mitigated by employing a hierarchical structure, standardized format, rendering RNs, and folding techniques; 3) *Accessibility* (303, 18.42%): many users find RNs inaccessible due to link deterioration, insufficient notification, and obfuscated RN locations. This can be alleviated by adopting appropriate locations and channels (such as project websites) and standardizing link management.; 4) *Production* (773, 46.99%): despite the high demand from RN producers, automating and standardizing the RN production process remains

Jianyu Wu · Hao He · Kai Gao · Wenxin Xiao · Minghui Zhou
Key Laboratory of High Confidence Software Technologies, Ministry of Education
Peking University, Beijing, China
Jingyue Li
Norwegian University of Science and Technology, Trondheim, Norway
E-mail: {jy.wu, heh, gaokai19, zhmh}@pku.edu.cn, wenxin.xiao@stu.pku.edu.cn, jingyue.li@ntnu.no

challenging. Developers resolve this problem by using some mature tools on GitHub (like Release Drafter). Additionally, offering guidance, clarifying responsibilities, and distributing workloads are effective in improving collaboration within the team. Mechanisms for distributing and verifying RNs are also selected to enhance synchronization management. Our taxonomy provides a comprehensive blueprint to improve RN production in practice and also reveals interesting future research directions.

Keywords Release engineering · Release note · Empirical study · Taxonomy

1 Introduction

When a new software version needs to be released, documentation for introducing the important changes, such as new features and bugs fixed, between two successive software releases (Moreno et al., 2017), i.e., Release Notes (RNs), emerges as a natural requirement. RNs offer a central source of information to 1) facilitate the communication between the software and its users[1] (Bi et al., 2020); 2) track the evolution of software and provide insights into the rationales behind specific changes (Maalej and Happel, 2010; Shihab et al., 2013; Wu et al., 2023). When upgrading software, consulting RNs is widely recognized as a best practice for users. For users, RNs are typically used to inform them of: 1) potentially beneficial changes, such as bug fixes, enhancements, and 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 utilize RNs to formally document development progress and plans for the upcoming release (Bi et al., 2020).

Providing an informative RN for users is a challenging task. Firstly, the tight deadlines in agile software development may even tempt developers to reduce the effort put into RNs.[2] Additionally, the creation of RNs can be a laborious and error-prone process. For example, the RN producers of Firefox have to traverse thousands of changes between two releases within a week and carefully select the significant ones to be included in the final RNs.[3] The survey by (Moreno et al., 2017) also reveals that *"creating a release note by hand is a difficult and effort-prone activity that can take up to eight hours"*. Despite substantial effort invested in producing RNs, the final RNs may be of low quality (bad organization, missing important changes, etc.) and these quality problems often trouble software users. Currently, many researchers focus on 1) summarizing information categories found in RNs (Abebe et al., 2016; Bi et al., 2020; Moreno et al., 2017); 2) identifying RN-related artifacts (Nath and Roy, 2021); and 3) exploring automated technologies to facilitate RN generation (Yang et al., 2021). However, there is still a lack of a systematic

[1]Throughout this paper, we use the term "user" to refer to anyone reading or referring to RNs for their tasks, including internal developers, downstream developers, and software end users.

[2]`https://github.com/vue-leaflet/Vue2Leaflet/issues/663`

[3]`https://wiki.mozilla.org/Release_Management/Release_Notes`

understanding of real RN-related issues[4] and the corresponding strategies, that is, how RNs go wrong or fail to meet users' expectations and how developers resolve the corresponding issues. Such an understanding can help formulate best practices and reveal important future research directions for automating and regulating RN production.

To bridge the knowledge gap, we pose the following research questions:

– *RQ1: What are the RN-related issues proposed on GitHub?*
– *RQ2: What strategies do developers adopt to resolve the RN-related issues?*

To answer RQ1, we collect 1,529 RN-related issues in the GitHub issue tracker from GHArchive[5] and build a comprehensive taxonomy of these issues using multiple rounds of open coding. The taxonomy is further validated through three semi-structured interviews. The results of RQ1 reveal four dimensions of RN-related issues:

– **Content (419, 25.47%)**: Issues concerning *Completeness* (274, 16.66%) are more prominent than *Correctness* (145, 8.81%), especially for breaking changes and new features. RN producers should ensure that these two types of changes are not omitted and are sufficiently described. Missing, wrong, and broken links are also particularly salient, which annoyingly prevent users from accessing supplementary information. Besides, some issues are less frequent but also critical and meaningful, such as missing security changes, license changes, dependency/environment specifications, etc.
– **Presentation (150, 9.12%)**: A considerable proportion of issues concern *Usability* (94, 5.71%). The issues concern *Language* (56, 3.40%), most frequently caused by spelling, writing style, grammar, etc. The poor layout of RNs increases users' difficulty in locating relevant information and may even cause users to miss their desired information.
– **Accessibility (303, 18.42%)**: Many users complain that RNs are difficult to access due to link deterioration (77, 4.68%), lack of notification (33, 2.01%), and covert RN locations (188, 11.43%). Users may also become irritated when they receive excessive notifications (5, 0.30%).
– **Production (773, 46.99%)**: Developers demonstrate a strong demand for *Automation* (314, 19.09%), but automated tools/scripts may lack desired features, tend to induce errors, and are hard/error-prone to configure. Additionally, without proper *Planning* (328, 19.94%), e.g., release schedules and deadlines, users may be confused about the absence of RNs. *Standardization* (94, 5.71%) of RN production, especially conventions for issues, pull requests (PRs), and commits, is vital to efficient RN production in complex and large software projects. Finally, attention should be given to ensure the *Consistency* (37, 2.25%) of RNs across various locations.

To answer RQ2, we perform multiple rounds of open coding on the above issues to construct a comprehensive framework of strategies employed by devel-

[4]In this paper, the term "RN-related issues" refers to the issues proposed by developers in the GitHub issue tracker system for software release notes.

[5]`https://www.gharchive.org`

opers. The framework consists of 483 solutions, organized into eight categories of strategies:

1. **Location & Channel options (149)**: Developers select diverse locations and channels to tackle the challenges of RN notification, access, and management, such as project websites (48), files in the repositories (38), GitHub Release Page (28), within apps (27), and instant Message Channels (8).
2. **Automation Recommendation (128)**: Developers adopt a total of 26 different automation tools to streamline the generation of RNs and enhance the efficiency of the release management workflow. The three most frequently chosen ones are `Cake.recipe` (19), `Release Drafter` (15), and GitHub's release features (12).
3. **Presentation Improvement (76)**: Developers improve RN's readability and organization by applying a hierarchical structure (26), rendering RNs (24), standardizing the RN format (17), and folding mechanisms (8).
4. **Completeness Validation (54)**: Developers recommend effective management for PRs, commits, and GitHub issues to validate the completeness of RN. This includes categorizing PRs based on affected components and change types (30), utilizing commit conventions such as Conventional Commits[6] (14), and assigning issues to milestones (6).
5. **Collaboration Management (22)**: Offering guidance to produce RNs (15), clarifying responsibilities (4), and distributing workloads (3) are three effective strategies to enhance the collaboration among the developers during RN production.
6. **Link Standardization (19)**: Formatting links (12), periodic verification (4), and centralized management (3) are commonly adopted measures to prevent users from accessing the intended website.
7. **Synchronization Management (18)**: Developers adopt the mechanisms to distribute (16) and verify (2) the RNs to ensure the consistency between RNs displayed across various places and with other relevant documentation.
8. **Timely Delivery (17)**: Implementing standardized publishing practices, such as establishing deadlines (10) or defining a sequential process (7), can greatly facilitate the timely production of RNs.

Based on our results, we further discuss the factors that drive developers to select these strategies and explore effective implementations in various scenarios. We additionally identify open research challenges of automated RN generation and testing of RN completeness/correctness in practice.

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 initial release. However,

[6] `https://www.conventionalcommits.org/`

successful software inevitably evolves into new versions. With the release of a new version, the need for documentation, i.e., Release Note (RN), to explain the changes in this version arises naturally. 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 (Holloway, 1985).

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 (Olsson and Bosch, 2014). 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 (Aghajani et al., 2020). 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.[7]

However, RNs remain an understudied research topic. Early studies only use RNs as a data source for understanding other software maintenance and evolution topics (Alali et al., 2008; Maalej and Happel, 2010; Shihab et al., 2013; Yu, 2009). 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. (2017) manually analyzes 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. (2016) manually analyzes 85 RNs from 15 software projects and identifies six types of information: title, system overview, resource requirement, installation, addressed issues, and caveat. Bi et al. (2020) 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. Nath and Roy (2022) explore and analyze relevant artifacts of 3,347 RNs on GitHub and conclude that issues, PRs, commits, and CVEs are four kinds of key artifacts in RNs. Moreover, Yang et al. (2022) collect 69,851 RNs of popular apps on the Google Play Store and identify six patterns on the app stores regarding the length and

[7] `https://wiki.mozilla.org/Release_Management/Release_Notes`

content updatability of RNs. Wu et al. (2023) analyze 612 RNs from top popular GitHub projects to characterize the structure, writing style, and content of these RNs. 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. (2020) perform a survey with 150 developers to investigate what kind of documentation types are considered important in software development. They find that although most developers consider RNs and change logs to be necessary, their absence is 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. (2020), we still lack a comprehensive empirical understanding of real issues in RN production and usage, as well as effective strategies to address them. Our taxonomy and framework provide 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 a 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".[8]

To facilitate the production of high quality RNs while reducing manual effort, many open-source projects have begun to adopt tools for automated RN generation, including `Semantic Release`[9] (~14k stars), `Release It`[10] (~4k stars), `Release Drafter`[11] (~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[12] 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

[8] `https://docs.openstack.org/project-team-guide/release-management.html#how-to-add-new-release-notes`

[9] `https://github.com/semantic-release/semantic-release`

[10] `https://github.com/release-it/release-it`

[11] `https://github.com/release-drafter/release-drafter`

[12] `https://github.com/angular/angular.js/blob/master/DEVELOPERS.md#commits`

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. (2016) propose a semi-automated RN generation tool which extracts change descriptions from issue trackers and organizes them by labels to meet the need of a specific audience. Moreno et al. (2017) 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 and Roy (2021) propose to generate RNs from commit messages and pull requests using text summarization and word embedding techniques. Jiang et al. (2021a) propose a language-agnostic approach to produce RNs from pull request text. Recently, Kamezawa et al. (2022) propose two deep learning-based approaches for generating RNs with unlabeled commits based on a dataset of approximately 82,000 English RNs and associated commit messages.

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 efforts on RN automation by summarizing best automation practices and reveals future research directions for improving automated tools. Except for RN automation, our framework of strategies can also be applied to address other RN-related issues, thereby filling a significant gap in the existing research.

3 RQ1: What are the RN-related issues proposed on GitHub?

3.1 Methodology

3.1.1 Data Collection

In this study, we choose to analyze GitHub issues, which developers use to track ideas, provide feedback, report bugs and initiate discussions.[13]We favor GitHub issues over Stack Overflow questions because GitHub issues contain more information, such as reports and discussions among developers, and provide concrete examples about how RNs fail, apart from developers' opinions.

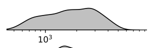
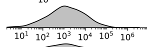
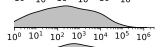
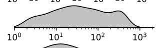
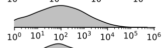
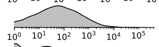
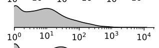
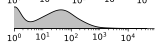
➤ **Mining GitHub**: GitHub is one of the most popular social coding platforms and provides access control and several collaboration features such as bug tracking, feature requests, and task management for every project. It is a commonly used data source for exploring software issues in previous works (Coelho et al., 2015; Humbatova et al., 2020). To this end, we use the GHArchive dataset to collect all GitHub issues that:

– have activities (at least one `IssueEvent`[14] in GHArchive) in 2021;

[13]`https://docs.github.com/en/issues/tracking-your-work-with-issues/about-issues`

[14]An IssueEvent can refer to any activity associated with issues, e.g., creating an issue and commenting on an issue. `https://docs.github.com/en/rest/issues/events`

Table 1: Repository Statistics of the Final RN-related Issue Dataset

	Mean	Median	Std.	Distribution*
Age (in Days)	1,989.97	1,790.00	1,055.23	
# of Commits	7,504.85	1,216.00	32,079.19	
# of Stars	4,399.52	232.00	14,124.59	
# of Contributors	92.49	33.00	123.59	
# of Forks	1,066.93	86.00	4,365.92	
# of Issues	382.21	52.00	2,207.51	
# of PRs	32.76	6.00	138.37	
# of Releases	86.20	17.00	423.64	

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

– 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. (2020) report that developers do not use automated RN generation tools while the number of automated RN generation tools is gaining increasing popularity recently (Section 2.2). This initial selection results in 3,297 issues from 1,139 repositories.

➤ **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 all the collected issues together as an initial familiarization of the dataset and exclude 1,768 issues that are not related to certain problems in RNs (i.e., *False Positives*), including the following cases:

– *Release Statements (1,162, 35.24%)*: The issue is only an official announcement of a release or a release note.
– *Non-Informative (228, 6.92%)*: The issue contains too little information (e.g., only a few words in title and description) to be understood by the inspectors.
– *Irrelevant (221, 6.70%)*: The issue happens to have the keyword "release note" in its title but actually refers to a problem not related to RNs.
– *Unreachable (98, 2.97%)*: The issue is no longer available on GitHub (e.g., the repository is deleted or made private, the issue is deleted, etc.).
– *Non-English (32, 0.97%)*: The issue contains non-English text and is not understandable by the inspectors.
– *Mistake (27, 0.82%)*: The issue reporter misunderstands the RN and reports a non-existent problem.

The final dataset for our study consists of 1,529 issues from 1,139 repositories. The repository statistics are summarized in Table 1 where we can observe that most issues come from repositories with long development history, high

Table 2: Statistics for Each Round of Manual Labeling

Round	1	2	3	4	5*	Total
Analyzed	459	356	356	356	90	1,529
Cohen's Kappa	-	0.80	0.87	0.88	-	-
Newly Added						
- #Dimensions	3	1	0	0	0	4
- #Categories	5	3	0	0	0	8
- #Subcategories	2	2	1	0	0	5
- #Leaf Nodes	48	14	10	4	-3	73

* The fifth round samples issues from the previous four rounds.

popularity, and sufficient development activities.[15] The size of our dataset is larger than similar software engineering studies that conduct qualitative manual analysis on text (e.g., studies on Stack Overflow posts and patch descriptions (Aghajani et al., 2019; Beyer et al., 2018; Chen et al., 2020; Tan and Zhou, 2019; Zhang et al., 2019a)).

We also gather information about the reporter and the duration for each RN-related issue in our final dataset to understand the characteristics of these issues. Specifically, GitHub defines five roles in the organization/project,[16] i.e., Owner, Member, Collaborator, Contributor, and None. We consider the first three roles with write permissions as the project's "Core" developers. The duration is calculated as the number of hours between the time the issue is reported and the time it is closed.[17]

3.1.2 Data Analysis

For the final 1,529 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 (Seaman, 1999). Similar to previous works (Chen et al., 2020; Humbatova et al., 2020), 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[18] and the statistics for each round of labeling are summarized in Table 2.

➤ **Pilot Construction.** We randomly sample 30% (459) of the 1,529 issues for a pilot construction of the taxonomy in the first round with two stages.

[15] The long tail distributions of most metrics are expected and common in mining software repository datasets (He et al., 2021; Zhang et al., 2019b).

[16] `https://docs.github.com/en/account-and-profile/setting-up-and-managing-your-account-on-github/managing-personal-account-settings/permission-levels-for-a-personal-account-repository`

[17] We exclude issues whose status is open and whose duration time is over one year in order to avoid bias introduced by forgetting to close and inactivity.

[18] Compared with the ICPC paper (Wu et al., 2022), we replicated the entire experiment on the extended dataset in this paper.

The inspectors mentioned in Section 3.1 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 intentions. Where necessary, they additionally check relevant code changes (i.e., pull requests/commits) and RNs that reveal the final solutions 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 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.

➤ **Extended Construction**: Based on the initial hierarchical taxonomy generated in Section 3.1.2, 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 0.80, 0.87, and 0.88, respectively, indicating increasing and high agreement between inspectors.

➤ **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 3, 4, and 8 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 the 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 nodes and split one leaf node into two leaf nodes finally. The interview time varies between 32 minutes and 2 hours. All 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 (Hove and Anda, 2005; Humbatova et al., 2020) show that participants talk much more when more than two interviewers conduct the interviews.

➤ **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.[19] Compared with our own results, the κ values are 0.89, 0.87, and 0.85, respectively, which also indicates a high agreement and thus good reproducibility. Our final taxonomy includes four dimensions, seven categories, four subcategories, and 70 leaf nodes. The entire manual construction process takes over three months to finish.

3.2 Results

We group all these issues into four **dimensions**, as depicted in Figure 3 - Figure 6. These figures present the hierarchical taxonomy of RN-related issues within each dimension:

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** (optional, e.g., *Missing*), and **leaf nodes** (e.g., *Missing Breaking Changes*). In each figure, we also can see the number of RN-related issues and percentages for all dimensions, categories, subcategories, and leaf nodes in the entire taxonomy.

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

Figure 1 and Figure 2 respectively illustrate the distribution of the percentage of reporters who are the core developers for the project and the duration time for commonly encountered leaf nodes of issues.[20]

In Figure 1, we can see that the top eight leaf nodes of issues are all under *Production*, especially including *Workflow Management* and *PR/Issue/Commit Management*. The issues within the *Production* dimension demand the involvement of more knowledgeable and experienced developers for effective identification and resolution. Consequently, the core developers should distribute additional attention to these issues. Conversely, certain issues within the *Content* dimension, such as *Missing Breaking Changes* and *Missing Links*, are less frequently reported by core developers compared to other issues.

Figure 2 further reveals distinct characteristics among the leaf nodes under the *Production* dimension, such as *Workflow Management* and *Request for Automation*. These issues exhibit a longer upper whisker and a shorter lower whisker, indicating a wider variation among higher values and a more concentrated distribution among lower values. We find that the issues reported by a significant proportion of core developers suggest that resolving these problems may require a longer time and much effort, with the exception of *When to Produce*. For example, addressing issues related to *Request for Automation* involves initial discussions on suitable scripts and tools. In the process of development or deployment, seeking comments and suggestions from other developers also results in a substantial time investment (Ahire, 2021). Furthermore, issues within the leaf nodes of *Limited Exposure*, *Lack Notification*, and *Poor Layout* also tend to have longer resolution processes. This is because they require extensive deliberation on the best locations to publicize RNs and how to effectively organize RNs. On the other hand, certain issues like *Spelling Errors* can be resolved relatively quickly with limited participation.

Summary: RN-related issues focus on four dimensions: *Content*, *Presentation*, *Accessibility*, and *Production*. Among these issues, nearly half (773, 46.99%) of them focus on *Production*; *Content*, *Accessibility*, and *Presentation* take 25.47%, 18.42%, and 9.12%. It is worth noting that a significant percentage of core developers report most issues within the *Production* dimension. These issues also tend to have a long duration time, indicating that resolving them may indeed require additional effort.

In the remainder of this section, we will describe our taxonomy with representative examples.

3.2.1 Content

In total, 419 issues discuss the **Content** of RNs, i.e., what information RNs should convey. Issues from the **Content** dimension can help better understand

[20]Considering a large number of leaf nodes, we only plot left nodes whose issue number is greater than 30, which cover all four dimensions and account for 77.87% of issues in our dataset.

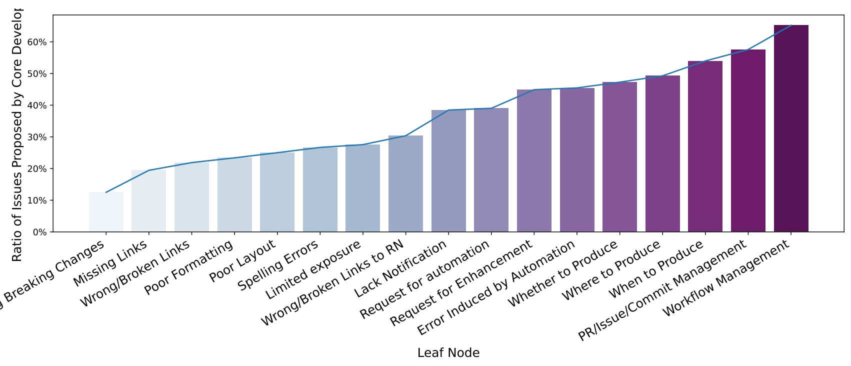


Fig. 1: For all issues in each leaf node (with ≥30 issues), the percentage of issue reporters who are core developers in the organizations/projects.

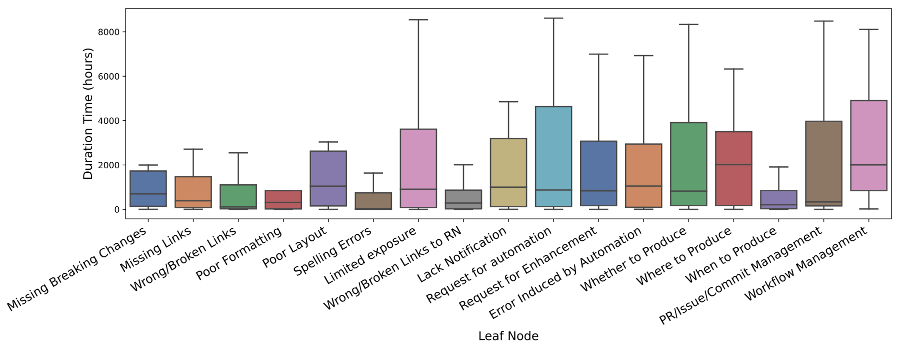


Fig. 2: For all issues in each leaf node (with ≥30 issues), the distribution of issue duration time (in hours).

1) what typical users would expect from RNs, and 2) what purposes RNs should serve as one kind of project documentation. This dimension consists of two categories: *Completeness* and *Correctness*.

Within this dimension, the majority of issues, specifically 16.66%, belong to the category of *Completeness*, while approximately 8.81% fall under the category of *Correctness*. Firstly, they frequently encounter wrong or broken links in RNs, accounting for 5.71% of the reported problems. These links hinder developers from accessing supplementary information, causing frustration. Secondly, missing breaking changes constitute 3.40% of the issues. This particular problem can mislead users and lead to significant consequences following an upgrade, such as software crashes.

➤ **Completeness (274, 16.66%)**: 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*.

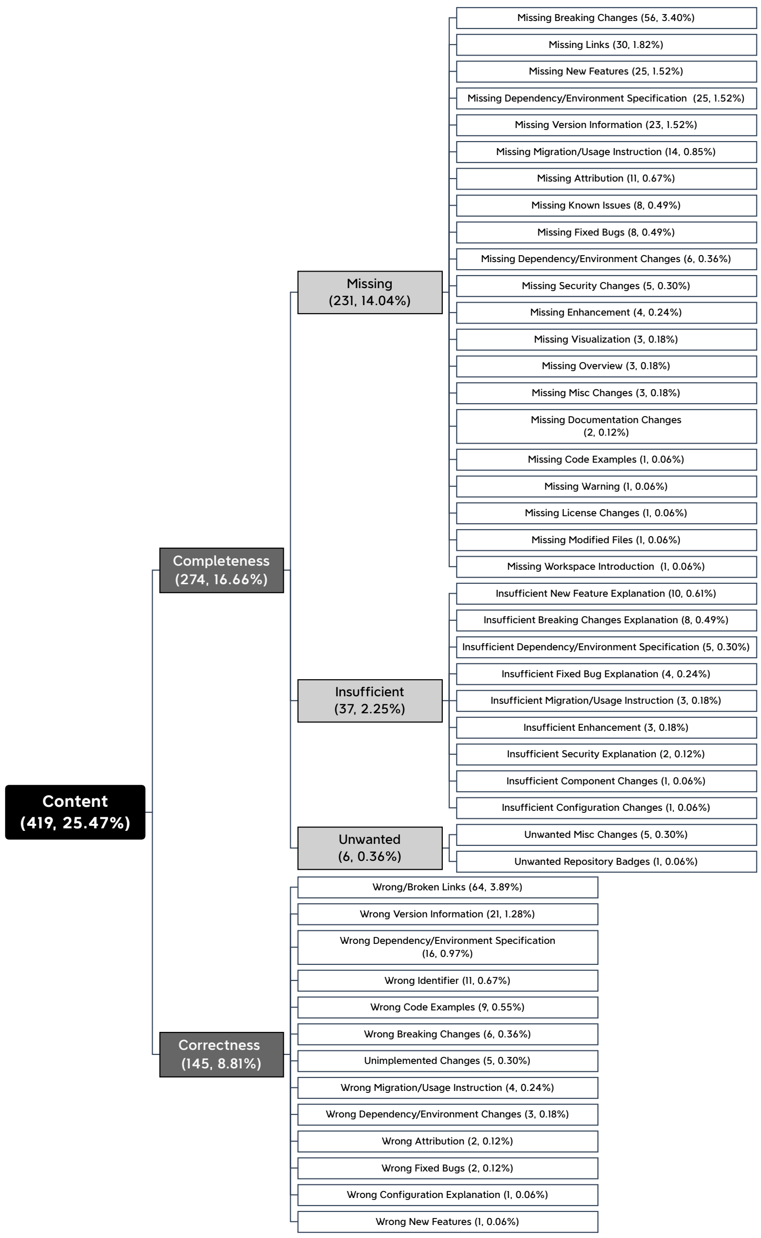

Content (419, 25.47%)
Completeness (274, 16.66%)
Missing (231, 14.04%)
Missing Breaking Changes (56, 3.40%)
Missing Links (30, 1.82%)
Missing New Features (25, 1.52%)
Missing Dependency/Environment Specification (25, 1.52%)
Missing Version Information (23, 1.52%)
Missing Migration/Usage Instruction (14, 0.85%)
Missing Attribution (11, 0.67%)
Missing Known Issues (8, 0.49%)
Missing Fixed Bugs (8, 0.49%)
Missing Dependency/Environment Changes (6, 0.36%)
Missing Security Changes (5, 0.30%)
Missing Enhancement (4, 0.24%)
Missing Visualization (3, 0.18%)
Missing Overview (3, 0.18%)
Missing Misc Changes (3, 0.18%)
Missing Documentation Changes (2, 0.12%)
Missing Code Examples (1, 0.06%)
Missing Warning (1, 0.06%)
Missing License Changes (1, 0.06%)
Missing Modified Files (1, 0.06%)
Missing Workspace Introduction (1, 0.06%)
Insufficient (37, 2.25%)
Insufficient New Feature Explanation (10, 0.61%)
Insufficient Breaking Changes Explanation (8, 0.49%)
Insufficient Dependency/Environment Specification (5, 0.30%)
Insufficient Fixed Bug Explanation (4, 0.24%)
Insufficient Migration/Usage Instruction (3, 0.18%)
Insufficient Enhancement (3, 0.18%)
Insufficient Security Explanation (2, 0.12%)
Insufficient Component Changes (1, 0.06%)
Insufficient Configuration Changes (1, 0.06%)
Unwanted (6, 0.36%)
Unwanted Misc Changes (5, 0.30%)
Unwanted Repository Badges (1, 0.06%)
Correctness (145, 8.81%)
Wrong/Broken Links (64, 3.89%)
Wrong Version Information (21, 1.28%)
Wrong Dependency/Environment Specification (16, 0.97%)
Wrong Identifier (11, 0.67%)
Wrong Code Examples (9, 0.55%)
Wrong Breaking Changes (6, 0.36%)
Unimplemented Changes (5, 0.30%)
Wrong Migration/Usage Instruction (4, 0.24%)
Wrong Dependency/Environment Changes (3, 0.18%)
Wrong Attribution (2, 0.12%)
Wrong Fixed Bugs (2, 0.12%)
Wrong Configuration Explanation (1, 0.06%)
Wrong New Features (1, 0.06%)

Fig. 3: RN-Related Issues under the Content Dimension. (■) Represents Dimensions, (▩) Represents Categories, (▒) Represents Subcategories, and (□) Represents Leaf Nodes.

Missing (231, 14.04%) subcategory refers to issues stating that some information perceived important by end users or developers is not included in the RN at all. The most frequently missed information in RNs includes:

- *Breaking Changes (56, 3.40%)*: 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 (30, 1.82%):* 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 (25, 1.52%):* Some implemented new features may be ignored in RNs, and (other) developers open issues in need of documenting their contributions.
- *Dependency/Environment Specification (25, 1.52%):* Undocumented dependency or environment specification may also accidentally break clients when users upgrade to new versions.
- *Version Information (23, 1.40%):* Some developers open issues to discuss 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.
- *Migration/Usage Instruction (14, 0.85%):* Some developers ask for migration or usage instructions in RNs to help them understand the impact of breaking changes and upgrade their client code.
- *Attribution (11, 0.67%):* Some issues are opened by repository members to discuss missing attribution to certain participants (e.g., contributors, funders, commenters, reviewers, etc.). As stated by a maintainer of `coq`, *in open source software, it is very important to give credit.*[22]
- *Known Issues (8, 0.49%):* Several issues mention that specific unsolved issues should be included in RNs to alert end users, e.g., including a crash about `NullPointerException` and its workaround in the corresponding RN of `NuGet`.[23]

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

Issues in the *Insufficient (37, 2.25%)* subcategory arise because certain information related to essential changes is not sufficiently detailed for users

[21] `https://github.com/cesanta/mongoose/issues/1271`

[22] `https://github.com/coq/coq/issues/7058/#issuecomment-375720879`

[23] `https://github.com/NuGet/docs.microsoft.com-nuget/issues/2410`

to understand. Two kinds of explanations are most likely to be insufficient in RNs:

- *New Feature Explanation (10, 0.61%):* 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.[24]
- *Breaking Change Explanation (8, 0.49%)*: 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 RNs (and probably warnings) for the `np.int` and other python alias deprecations.*[25]

Other insufficiently explained information includes dependency/environment specifications, fixed bugs, migration/usage instructions, enhancements explanation, security changes, configuration changes, and component changes.

Interestingly, nine issues care about *Unwanted* information in RNs, but they are likely to be only occasional. Five issues state that only critical/developer-impacting changes should go in RNs instead of listing all miscellaneous changes[26], while one issue[27] mentions that repository badges should not occur in RNs.

➤ **Correctness (145, 8.81%)**: This category means that information described in RNs conveys inaccurate information.

Contrary to our intuition, the majority is *Wrong/Broken Links (64, 3.89%)*, which refers to cases where links in RNs cannot be opened or directed 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 a poor reading experience for RN readers.

Moreover, 21 issues are related to *Wrong Version Information (21, 1.28%)*, including version number/name, version date, checksum, and most of which are caused by copy-pasting from previous RNs.[28,29,30] Other kinds of change descriptions that can go wrong include dependency/environment specification, identifier, code examples, breaking changes, migration/usage instructions, unimplemented changes, attribution, explanation of configuration, dependency/environment changes, etc.

[24] `https://github.com/keras-team/keras/issues/11517`
[25] `https://github.com/numpy/numpy/issues/17977`
[26] `https://github.com/Azure/azure-sdk/issues/1873`
[27] `https://github.com/cwarwicker/moodle-tool_ribbons/issues/3`
[28] `https://github.com/nushell/nushell/issues/3238`
[29] `https://github.com/babel/babel/issues/12961`
[30] `https://github.com/MicrosoftEdge/WebView2Feedback/issues/882`

Summary: Nearly two-thirds (65.39%) of issues within this dimension concerns *Completeness*, while only about one-third (34.61%) concerns *Correctness*. Developers are most likely to 1) report wrong/broken links in RNs (5.71%), which annoyingly prevent them from accessing supplementary information, and 2) missing breaking changes (3.40%), which may mislead users and incur severe consequences after upgrading (e.g., crash).

3.2.2 Presentation

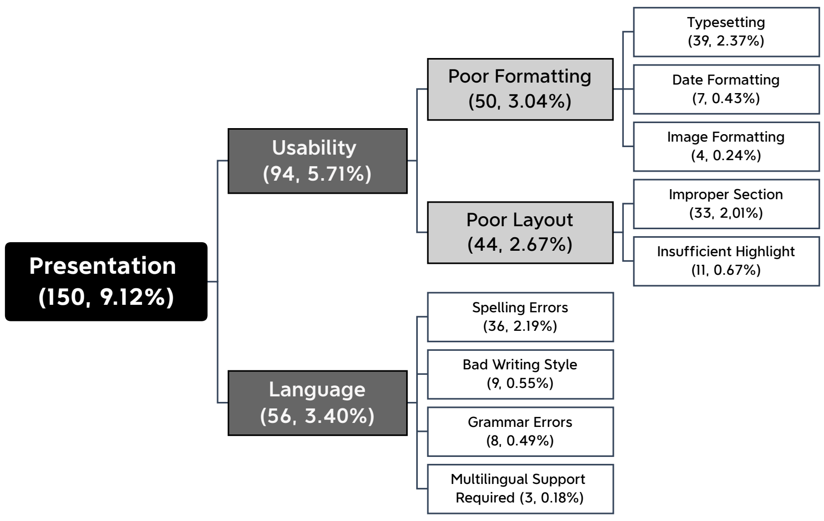

Presentation (150, 9.12%)
Usability (94, 5.71%)
Poor Formatting (50, 3.04%)
Typesetting (39, 2.37%)
Date Formatting (7, 0.43%)
Image Formatting (4, 0.24%)
Poor Layout (44, 2.67%)
Improper Section (33, 2,01%)
Insufficient Highlight (11, 0.67%)
Language (56, 3.40%)
Spelling Errors (36, 2.19%)
Bad Writing Style (9, 0.55%)
Grammar Errors (8, 0.49%)
Multilingual Support Required (3, 0.18%)

Fig. 4: RN-Related Issues under the Presentation Dimension.

150 issues are related to **Presentation** with two categories: *Language (56, 3.40%)* and *Usability (94, 5.71%)*. Issues from the **Presentation** dimension help reveal how information should be organized, formatted, highlighted, visualized, and phrased in a RN, so that different RN users can make use of the RN for their purposes with maximum efficiency.

The primary concern within this dimension is *Usability*. One of the key aspects affecting *Usability* is the *Poor Layout* of the RNs, which has the potential to bury important information and mislead end users. Additionally, the *Language* errors of the RNs are caused by various factors, including spelling and grammar errors, a subpar writing style, and the need for multilingual support.

➢ **Usability (94, 5.71%)**: This category refers to the degree to which users can use RNs to achieve their objectives effectively.

❖ More than half of the issues arise from *Poor Formatting (50, 3.04%)*, resulting in the impression that the software development process is unprofessional or unorganized, thereby reducing user engagement.[31] There are three leaf nodes under this dimension: *Typesetting (39, 2.37%)*, *Data Formatting (7, 0.43%)* and *Image Formatting (4, 0.24%)*. Most of these issues are caused by *Typesetting*, that is, the misusing syntax of markup languages (e.g., HTML) and usually lead to abnormal display, such as failing to display list due to missing HTML linebreaks.[32] Furthermore, it is worth noting that inconsistencies in date formats can lead to confusion among individuals residing in different geographical regions.[33] Additionally, the content of the RN is sometimes distracted by improper image format, such as GIF animations[34], and excessive resource loading, such as high-resolution images.[35]

❖ 49 issues in this category are related to *Poor Layout (44, 2.67%)*, which means the changes are not clearly organized in RNs. Two key challenges for improving RN readability are *Improper Section (33, 2.01%)* and *Insufficient Highlight (11, 0.67%)*. Since different stakeholders may be interested in different kinds of information, RNs need to have a proper section structure for them to quickly locate the information they want (Abebe et al., 2016; Bi et al., 2020). *Improper Section* and *Insufficient Highlight* can increase the time needed for users to grab valuable information, annoy readers,[36]bury good features,[37] 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.[38]

➤ **Languages (56, 3.40%)**: This category of issues concerns whether the language of RN is fluent and easy to read, including four leaf nodes: *Spelling Errors (36, 2.19%)*, *Bad Writing Style (9, 0.55%)*, *Grammar Errors (8, 0.49%)*, and *Multilingual Support Required (3, 0.18%)*. Although *Spelling Errors* are often reported by non-core developers and are quick to fix, they may be hard to notice, especially for technical terms (e.g., MACs and Macs).[39]Thus, extra manpower or time to conduct thorough inspections is essential. Also, certain writing styles can make RNs clearer and more easily understandable, such as describing what happens after a bug is fixed instead of what used to happen.[40] One issue asks for multilingual support, which helps more users understand RNs and enables product adaptation to a broader market.[41]

[31] `https://github.com/bitwarden/web/issues/766`
[32] `https://github.com/nathanwoulfe/Plumber-2/issues/64`
[33] `https://github.com/bigcommerce/cornerstone/issues/1990`
[34] `https://github.com/microsoft/vscode/issues/138720`
[35] `https://github.com/microsoft/vscode-docs/issues/5000`
[36] `https://github.com/vaadin/platform/issues/2178`
[37] `https://github.com/EOSIO/eos/issues/9903`
[38] `https://github.com/electron/electron/issues/28375`
[39] `https://github.com/MicrosoftDocs/OfficeDocs-OfficeUpdates/issues/300`
[40] `https://github.com/hazelcast/cloud-docs/issues/18`
[41] `https://github.com/microsoft/appcenter/issues/2207`

Summary: The majority of issues (94, 5.71%) within this dimension concerns *Usability*, especially the poor layout, which may bury important information and lead to end users' misjudgement. The *Language* (56, 3.40%) errors of RNs are caused by several factors, including spelling and grammar errors, poor writing style, and the requirement for multilingual support.

3.2.3 Accessibility

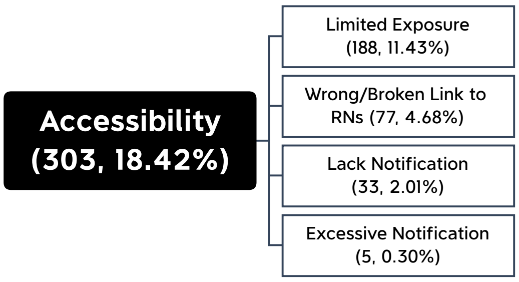

Accessibility (303, 18.42%)
Limited Exposure (188, 11.43%)
Wrong/Broken Link to RNs (77, 4.68%)
Lack Notification (33, 2.01%)
Excessive Notification (5, 0.30%)

Fig. 5: RN-Related Issues under the Accessibility Dimension.

Users face various challenges when trying to access RNs. 303 issues are related to **Accessibility**, i.e., how to make RNs accessible to a broad audience, with four leaf nodes: *Limited Exposure (188, 11.43%)*, *Wrong/Broken Link to RNs (77, 4.68%)*, *Lacking Notification (33, 2.01%)*, and *Excessive Notification (5, 0.30%)*. Issues in this dimension thus reveal how a software project should distribute its RNs, maintain links, and notify its users properly.

➤ **Limited Exposure (188, 11.43%)**: 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.[42] 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.*[43]

➤ **Wrong/Broken Link to RNs (77, 4.68%)**: *Wrong/Broken Links* and *Missing Links* under the **Content** dimension describe cases where links in RNs are broken or wrong (see Section 3.2.1). The issues here are related to external links supposed to point to RNs themselves (in the project website, etc.) that may be broken. For example:

[42] `https://github.com/hedgedoc/hedgedoc.github.io/issues/59`
[43] `https://github.com/dotnet/SqlClient/issues/1123`

1. *The section of the front page of this repo "Links to release notes" is full of dead links.*[44]
2. *Links to the Agent release notes from the APM docs left nav are returning 404s.*[45]

➤ **Lack (33, 2.01%) & Excessive(5, 0.30%) Notification**: 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.*[46] However, providing notifications is not always a silver bullet and users may also become irritated when they receive excessive notifications.[47]

Summary: Users encounter a diverse range of difficulties in accessing RNs, including *Limited Exposure* (188, 11.43%), *Wrong/Broken Links to RNs* (77, 4.68%), *Lack of Notification* (33, 2.01%) and *Excessive Notification (5, 0.30%).*

3.2.4 Production

773 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 four categories: *Planning (328, 19.94%)*, *Automation (314, 19.09%)*, and *Standardization (94, 5.71%)* and *Consistency (37, 2.25%)*.

Developers demonstrate a significant interest in automation, which accounts for 19.09% of the issues. However, automated tools/scripts often lack desired features, tend to introduce errors, and can be challenging or error-prone to configure. Inadequate planning, such as the absence of release schedules and deadlines, contributes to 19.94% of the taxonomy. This lack of planning can confuse users regarding the availability of RNs. Furthermore, proper standardization of RN production, particularly in terms of conventions for PRs, issues, and commits, is crucial for facilitating efficient RN production in large software projects. This aspect of standardization accounts for 5.71% of the difficulties, with PRs, issues, and commits making up 3.83% faced by developers. Besides, it is important to pay attention to the inconsistency between RNs' different locations and related documentation.

[44] https://github.com/HOKGroup/HOK-Revit-Addins/issues/194
[45] https://github.com/newrelic/docs-website/issues/758
[46] https://github.com/sublimehq/sublimes_merge/issues/1097
[47] https://github.com/cpriest/SnapLinksPlus/issues/287

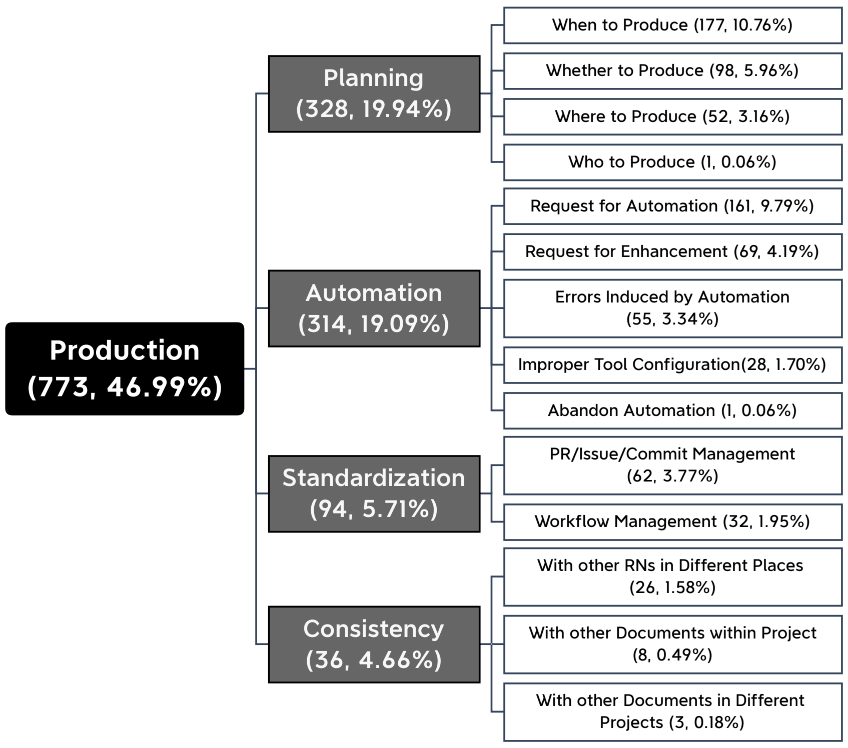

Production (773, 46.99%)
Planning (328, 19.94%)
When to Produce (177, 10.76%)
Whether to Produce (98, 5.96%)
Where to Produce (52, 3.16%)
Who to Produce (1, 0.06%)
Automation (314, 19.09%)
Request for Automation (161, 9.79%)
Request for Enhancement (69, 4.19%)
Errors Induced by Automation (55, 3.34%)
Improper Tool Configuration(28, 1.70%)
Abandon Automation (1, 0.06%)
Standardization (94, 5.71%)
PR/Issue/Commit Management (62, 3.77%)
Workflow Management (32, 1.95%)
Consistency (36, 4.66%)
With other RNs in Different Places (26, 1.58%)
With other Documents within Project (8, 0.49%)
With other Documents in Different Projects (3, 0.18%)

Fig. 6: RN-Related Issues under the Production Dimension.

➢ **Planning (328, 19.94%)** This category of issues are with four subcategories: *When to Produce (176, 10.76%)*, *Whether to Produce (98, 5.96%)*, *Where to Produce (52, 3.16%)*, and *Who to Produce (1, 0.06%)*.

❖ *When to Produce (176, 10.76%)*: In these cases, 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.*[48] Besides, although most projects provide RNs for every version, some of them are released too late to be helpful which disappoints RN users.[49]

❖ *Whether to Produce (98, 5.96%)*: 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.[50]

[48]`https://github.com/facebookexperimental/Recoil/issues/916`
[49]`https://github.com/hashicorp/terraform-ls/issues/533`
[50]`https://github.com/getsentry/sentry-laravel/issues/260`

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.[51] 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.*[53] 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.[54,55] Other project maintainers acknowledge the necessity of RNs, but they lack time for them.[56]

❖ *Where to Produce (52, 3.16%)* subcategory concern where to collaboratively edit and store RN files. Although GitHub provides convenient release functionalities[57] 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.[58] One special case mentions the lack of accountability in RN production and suggests someone should be responsible for it.[59]

➢ **Automation (314, 19.09%)**: This category reflects four kinds of issues that developers frequently encounter on automated RN generation: *Request for Automation (168, 20.83%)*, *Request for Enhancement (69, 8.93%)*, *Error Induced by Automation (55, 7.06%)*, and *Improper Tool Configuration (28, 3.59%)* and *Abandon Automation (1, 0.13%).*

❖ *Request for Automation(161, 9.79%) & Abandon Automation(1, 0.06%)*: More than half of the 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. These issues are mainly raised by core developers and with an extremely long duration. 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*

[51] https://github.com/rayokota/kcache/issues/79

[52] https://github.com/common-workflow-language/cwlviewer/issues/328#issuecomment-823410828

[53] https://github.com/common-workflow-language/cwlviewer/issues/328

[54] https://github.com/dask/fastparquet/issues/544

[55] https://github.com/dtolnay/semver/issues/87

[56] https://github.com/negomi/react-burger-menu/issues/432

[57] https://docs.github.com/en/repositories/releasing-projects-on-github/about-releases

[58] https://github.com/decred/dcrwallet/issues/1966

[59] https://github.com/wellcomecollection/wellcomecollection.org/issues/5236

generated every time a PR is merged.[60] Automation can be accomplished in two main ways according to developers: 1) writing project-specific scripts, e.g., filling predefined templates and publishing releases,[61] and 2) adopting existing automated tools such as `Semantic Release`, `github-changelog-generator`, and `Release Drafter`. Although RN automation is a huge help in reducing manual toil, some developers express their concerns about 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. One issue mentions they give up automation to generate RNs because they think *no need to make a release for every new push to main.*[62]

❖ *Request for Enhancement (69, 4.19%)*: These issues reveal suggested improvements in automated generation tools and scripts by users. Specifically, users mostly request three kinds of support: 1) automated generation of RNs for different branches;[63] 2) automated retrieval of related information from multiple repositories;[64]3) automated supplement of details, e.g., CVEs[65], attribution[66], and PR comments[67]. 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.[68] 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.[69] Some developers suggest current tools to add support to automate RN publishing.

❖ *Errors Induced by Automation (55, 3.34%)*: 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 lead to unexpected RNs, e.g., wrong/missing content[70], repetition[71], and incorrect positions[72]; the other one is that these defects affect the generation process, e.g., not generating RNs that exceed a certain length.[73,74] 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 sev-

[60]`https://github.com/italia/spid-compliant-certificates-python/issues/6`
[61]`https://github.com/eclipse/rdf4j/issues/2784`
[62]`https://github.com/cpriest/SnapLinksPlus/issues/287`
[63]`https://github.com/opensearch-project/OpenSearch/issues/789`
[64]`https://github.com/rfennell/ReleaseNotesAction/issues/131`
[65]`https://github.com/kubernetes/release/issues/1354`
[66]`https://github.com/renovatebot/renovate/issues/8605`
[67]`https://github.com/stephend017/snake-charmer/issues/27`
[68]`https://github.com/cockroachdb/cockroach/issues/42163`
[69]`https://github.com/chef/automate/issues/2141`
[70]`https://github.com/saltstack/salt/issues/54752`
[71]`https://github.com/gohugoio/hugo/pull/83`
[72]`https://github.com/gitpod-io/gitpod/issues/714`
[73]`https://github.com/microsoft/azure-pipelines-tasks/issues/11922`
[74]`https://github.com/digidem/mapeo-mobile/issues/581`

eral issues are caused by its complex mechanisms, such as branch control[75], rebase[76], and release tag[77], which developers need to pay more attention to in design.

❖ *Improper Tool Configuration (28, 1.70%)*: 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[78], certain types of changes[79], and triggered conditions[80]. Besides, parameter misconfiguration is another common cause, including the construction of paths[81], repository names[82], environment variables[83], etc.

➤ **Standardization (94, 5.71%)**: This category of issues refers to what standardizations should be followed to simplify and ease the production of RNs. It covers two leaf nodes of issues: *PR/Issue/Commit Management (63, 3.83%)* and *Workflow Management (32, 1.95%).*

❖ *PR/Issue/Commit Management (63, 3.83%)*: 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.*[84]

❖ *Workflow Management(32, 1.95%)*: This subcategory refers to issues discussing formulation of RN production workflow or improvement on existing workflow. Among the issues, most issues discuss requesting for a workflow to standardize the process of producing RNs, which helps ensure that the notes are accurate and effective. For example, a developer who comes 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.*[85] These issues are also reported by the core developers, like *Request for Automation* and tend to take a considerable amount of time to resolve.

[75]https://github.com/BlueWallet/BlueWallet/issues/3145
[76]https://github.com/istio/istio/issues/31816
[77]https://github.com/DataDog/dd-trace-py/issues/2000
[78]https://github.com/layer5io/meshery/issues/2907
[79]https://github.com/PSBicep/PSBicep/issues/103
[80]https://github.com/corgibytes/freshli-lib/pull/301/files
[81]https://github.com/zammad/zammad-helm/issues/99
[82]https://github.com/corgibytes/freshli-lib/pull/279/files
[83]https://github.com/waifu-motivator/wmp-env-action/issues/29
[84]https://github.com/pytorch/vision/issues/3351
[85]https://github.com/mantidproject/mantidimaging/pull/798

➢ **Consistency (37, 2.25%)**: This category refers to issues about inconsistencies between 1) RNs published in different places, 2) RNs and other documentation within the project, and 3) RNs and documentation in other projects. 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.*[86] 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.*[87] Finally, RNs sometimes need to include changes or attribution information from closely related projects, which requires the collaboration of developers from the related projects.

Summary: Developers show a strong interest in *Automation* (314, 19.09%), but automated tools/scripts may lack desired features, tend to induce errors, and are hard or error-prone to configure. Additionally, without proper *Planning* (328, 19.94%), e.g. release schedules and deadlines, users may be confused about the absence of RNs. *Standardization* (94, 5.71%) of RN production, especially conventions for pull requests (PRs), issues, and commits (63, 3.83%), is vital for enabling efficient RN production in large software projects. The *Consistency* (37, 2.25%) checks between the RNs under different locations are also worth noting.

4 RQ2: What strategies do developers adopt to resolve the RN-related issues

As revealed in Section 3.2, developers encounter a broad spectrum of challenges in producing and using RNs, and it becomes crucial to formulate strategies to mitigate these challenges. In this section, we aim to gain a comprehensive understanding of the strategies employed by developers in addressing these RN-related issues.

4.1 Methodology

To identify the final selection of strategies adopted by developers, we meticulously re-examine all 1,529 RN-related issues identified in RQ1 for the second

[86] `https://github.com/newrelic/docs-website/issues/2127`
[87] `https://github.com/rotki/rotki/issues/2527`

time. Considering that an issue may or may not contain the resolved solutions, the same inspectors in RQ1 first collectively go through all issues and identify issues with such strategies. In this way, we identify 435 issues containing solutions adopted by developers. Then, we conduct a similar open coding procedure as in Section 3.1.2 on all 1,529 issues to categorize the strategies.

More specifically, the inspectors first randomly sample 30% (130) of the 435 issues for a pilot study to derive best practices. In the pilot study, the inspectors independently review the status, labels, and comments related to each RN-related issue, as well as analyze the corresponding code changes. If an issue is related to multiple solutions, it will be assigned multiple initial codes. Then they generate initial codes to describe the solution and merge similar codes into categories. Subsequently, the inspectors and an arbitrator hold a meeting to mediate, discuss, and resolve any disagreements in the labeling results until a final agreement is reached on the code book. Following this, the inspectors independently utilize the code book to label the remaining 70% of the issues, and any disagreements are resolved through discussions. The inter-rater agreement (Cohen's κ) between the two inspectors is 0.85, indicating a high agreement.

4.2 Results

We summarize the strategies into eight categories, encompassing a total of 483 solutions from 435 RN-related issues, to effectively resolve the respective challenges. In this section, the number in the bracket of the section title indicates the frequency of the solutions. Figure 7 illustrates the mapping between these strategies and their targeted RN-related issues.

4.2.1 Location & Channel Options (149)

Strategically determining the appropriate locations for publicizing RNs and selecting effective channels to notify users can address several important challenges, such as *Limited Exposure*, *Lack Notification*. It also mitigates the challenges (*Where to Produce*) of the positions for collaborative editing among RN producers.

Among the 149 solutions, developers have adopted various locations and channels to place RNs. These include:

- *Project Websites (48)*: Developers have utilized project websites to provide accessible and centralized access to RNs for each software version. They also suggest using a specific URL for the latest RN.[88]
- *Files in Repositories (38)*: Some developers think that providing a RN file in the repository (e.g., the root directory or the `doc` folder) is more important than storing RNs on GitHub Release Page.[89] A RN file in the

[88]`https://github.com/3drepo/3drepo.io/issues/2502`
[89]`https://github.com/decred/dcrwallet/issues/1966`

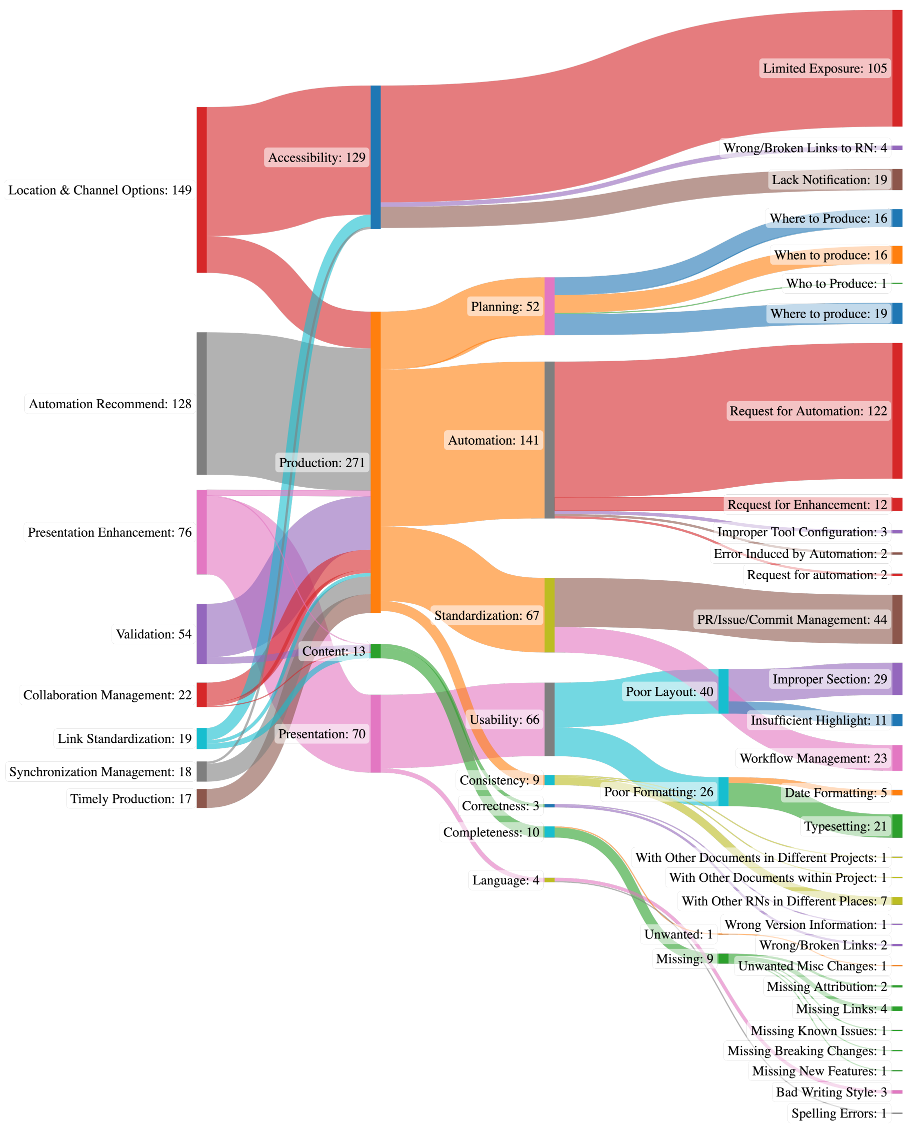


Fig. 7: The Sankey Diagram Illustrates the Mapping Between Each Category of Strategies and the Corresponding RN-related Issues.

repository makes the repository more self-contained (not depending on GitHub) and allows the usage of collaborative editing tools like Git.[90] The OpenStack community also requires that its projects must include RN files

[90] `https://github.com/decred/dcrwallet/issues/1966`

to record version changes and believes that this way can work on multiple patches simultaneously and reduce merge conflicts.[91]
- *GitHub Release Page (28)*: GitHub provides a dedicated page to display the release history. Many issues show that developers often check GitHub Release Page 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.*[92]
- *Within Apps (27)*: Some developers have integrated RNs directly within their app itself. Application software can provide buttons and links to access the latest RN, e.g. an "about" button.[93] RN notifications can also be displayed when a new version is released.[94]
- *Instant Message Channels (8)*: Some developers have chosen to disseminate RNs through instant message channels. Such channels can be used to immediately deliver new RNs to subscribed end-users and facilitate discussions and feedback related to the release, e.g. Slack[95], Discord[96] and Telegram[97].

4.2.2 Automation Recommendation (128)

To effectively address the RN-related issues identified under leaf nodes such as *Request for Automation*, *Request for Enhancement*, *Improper Tool Configuration*, and *Errors induced by Automation*, 128 solutions adopt an appropriate automated method for generating RNs. By leveraging proper automation, developers can streamline the RN generation process and ensure that all relevant information is included accurately. This not only enhances efficiency and effectiveness but also greatly reduces the risk of omission errors in the final RNs.

Considerable efforts have been dedicated to automating the production of RNs on GitHub through the utilization of custom scripts as well as a diverse range of specialized tools.[98] We identify 26 types of automation tools for generating RNs among these solutions. Figure 8 illustrates the distribution of these automated generation tools and custom scripts employed by developers. The number following the comma indicates the frequency of issues that have adopted each respective tool or script.

[91] `https://docs.openstack.org/reno/latest/`

[92] `https://github.com/newrelic/docs-website/issues/2127`

[93] `https://github.com/alteryx/woodwork/issues/808`

[94] `https://github.com/chef/chef-workstation-app/issues/302`

[95] `https://github.com/cdr/code-server/issues/3193`

[96] `https://github.com/Python-Practice-Discord/template_python_discord_bot/issues/1`

[97] `https://github.com/wabarc/wayback/issues/48`

[98] Most automation tools primarily focus on automating RN generation, while there are also several tools recommended for assisting in RN management. In the following strategies, we will introduce other tools according to their goals.

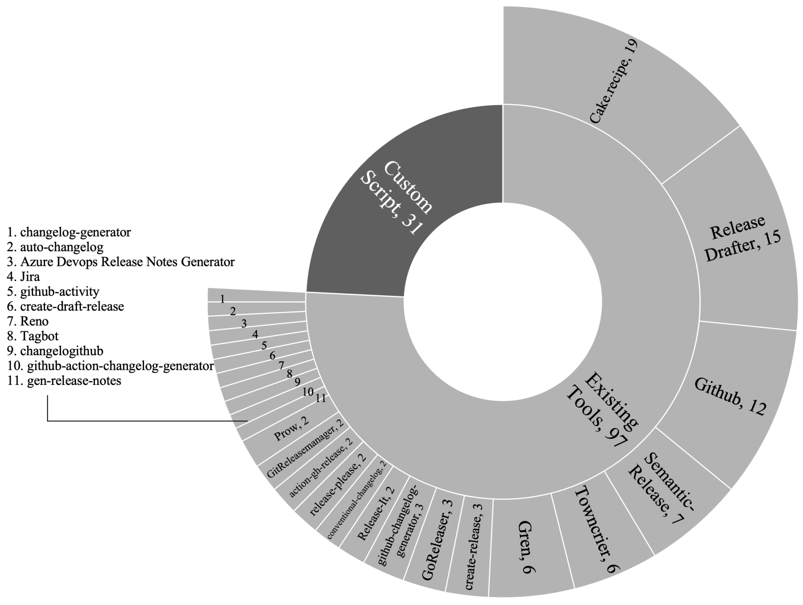


Fig. 8: The Distribution of Automated Generation Tools and Scripts for RNs in our Dataset

Among these tools, we find the most recommended tools on GitHub for generating RN are `Cake.recipe`[99] (19), `Release Drafter` (15), the release features provided by GitHub (12), `Semantic Release` (7), `Gren`[100] (6), and `Towncrier`[101] (6). Figure 9 provides the basic information of the six tools. We find that `Release Drafter`, `Gren`, and the official release features provided by GitHub are specifically designed for generating RNs on GitHub, while `Semantic Release` and `Cake.recipe` also offer extra functions, such as releasing management and publishing packages. The majority of these tools have been in development for more than five years and have gained significant popularity, attracting numerous projects to deploy them.[102] Especially, GitHub introduced a new feature that enables the automatic generation of RNs in October 2021 based on PRs between two releases.[103] These automation tools utilize various information as input sources, such as PRs (like the `Release Draft`), issues (like `Gren` and `Cake.recipt`) and commits (like `Gren` and `Semantic Release`). Then, they categorize the input sources based on tagged labels and extract their descriptions to generate RNs.

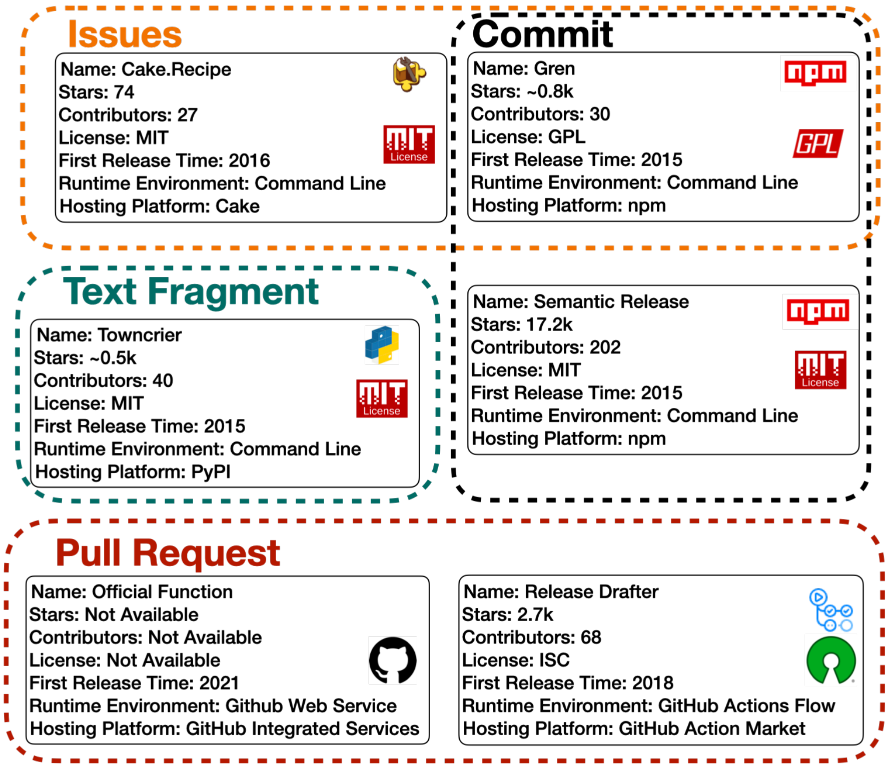
[99]`https://github.com/cake-contrib/Cake.Recipe`

[100]`https://github.com/github-tools/github-release-notes`

[101]`https://github.com/twisted/towncrier`

[102]`Cake.recipe` is exclusively deployed across multiple projects but only within their organization and with 74 stars.

[103]`https://github.blog/2021-10-04-beta-github-releases-improving-release-experience/`



Name: Cake.Recipe
Stars: 74
Contributors: 27
License: MIT
First Release Time: 2016
Runtime Environment: Command Line
Hosting Platform: Cake

Name: Gren
Stars: ~0.8k
Contributors: 30
License: GPL
First Release Time: 2015
Runtime Environment: Command Line
Hosting Platform: npm

Name: Towncrier
Stars: ~0.5k
Contributors: 40
License: MIT
First Release Time: 2015
Runtime Environment: Command Line
Hosting Platform: PyPI

Name: Semantic Release
Stars: 17.2k
Contributors: 202
License: MIT
First Release Time: 2015
Runtime Environment: Command Line
Hosting Platform: npm

Name: Official Function
Stars: Not Available
Contributors: Not Available
License: Not Available
First Release Time: 2021
Runtime Environment: Github Web Service
Hosting Platform: GitHub Integrated Services

Name: Release Drafter
Stars: 2.7k
Contributors: 68
License: ISC
First Release Time: 2018
Runtime Environment: GitHub Actions Flow
Hosting Platform: GitHub Action Market

Fig. 9: Basic Information of Popular Tools to Generate RNs in our Dataset

4.2.3 Presentation Enhancement (76)

In Section 3.2.2, we introduce the negative impact of the presentation of RNs, such as increasing the time required for users to access valuable information, annoying readers, and causing import changes to be overlooked by the impacted users. Based on the issues under the *Language* and *Usability* categories, as well as the issues related to *Request for Enhancement*, *Workflow Management*, *Request for Automation* and *Missing Known Issues*, we identify 76 solutions to improve the presentation of RNs:

– *Hierarchical Structure (26)*: Utilizing a hierarchical structure and categorizing the changes into sections or headings based on their significance or functionality can fulfill the specific needs of the audience. The recommendations include 1) organizing changes into a separate section,[104] rather than a plain list; 2) highlighting important or breaking changes;[105] 3) restruc-

[104]`https://github.com/cockroachdb/cockroach/issues/57898`
[105]`https://github.com/EOSIO/eos/issues/9903`

turing current sections by incorporating new ones such as "Enhancements" or "Known Issues".[106]
– *Rendering (24)*: Rendering RNs by using a lightweight markup language, such as Markdown, instead of plain text, offers multiple benefits. It can improve the consistency of RNs across various platforms and devices, such as web browsers and mobile devices. Additionally, it enhances the readability and visual appeal of the RNs, making them more engaging and user-friendly. For example, a developer claims *"when I go to packages I already use to see what's changed between releases, the Release Notes for that package show unformatted markdown, which is difficult to read compared to the rendered form"*.[107] To ensure that the importance of breaking changes is readily apparent, using markups (e.g., icons or emojis) to highlight breaking changes is strongly recommended.[108]
– *Standardized Format (17)*: RN formats can be standardized by implementing various standardization, such as date format,[109] file names, and version schemas,[110] phrasal verbs for sections, sentence case for captions, and diagram labels, etc.[111]
– *Folding Mechanism (8)*: Incorporating a folding mechanism in RNs can be a valuable addition. They can shorten RNs and fold a lengthy list of details to focus on the specific changes they are interested in, reducing information overload and providing a more streamlined reading experience for users by using detail/summary tags provided by GitHub Release Page,[112] HTML or Markdown features.[113]

A case proposes that by implementing reverse chronological lists of new features, breaking changes, and bug fixes, the reading experience for non-incremental release notes will be greatly improved.[114]

4.2.4 Completeness Validation (54)

Due to limited manpower or time available for conducting thorough inspections and the challenge of comprehending all changes between versions, RNs are more frequently affected by completeness problems, such as *PR/Issue/Commit Management*, *Missing New Features*, *Missing Links*, *Wrong/Broken Links*, *Missing Attribution* and *Unwanted Misc Changes*. Additionally, difficulties arise in preparing relevant PRs, issues, and commits for RNs, as highlighted in Section 3.2.4.

[106] `https://www.github.com/elastic/security-docs/issues/564`
[107] `https://github.com/NuGet/NuGetGallery/issues/8889`
[108] `https://github.com/charleskorn/kaml/issues/138`
[109] `https://github.com/bigcommerce/cornerstone/issues/1990`
[110] `https://github.com/mr-ice/maptool-macros/issues/112`
[111] `https://github.com/StyleGuides/WritingStyleGuide/issues/268`
[112] `https://github.com/prisma/prisma/issues/5913#issuecomment-788326709`
[113] `https://github.com/vaadin/platform/issues/2178`
[114] `https://github.com/chef/automate/issues/2141`

Through 54 selected solutions, we find that systematic and structured approaches to labelling and organizing PR/commits/issues can mitigate the challenges of completeness validation:

– Pull Requests (30): several large projects adopt labeling each PR with predefined labels to apply the sanity check[115]. In the case of `pytorch/vision`, developers reach a consensus on categorizing each PR with labels describing affected components and changed types (e.g. breaking changes and improvements).[116]
– Commits (14): developers prefer to adopt a convention for writing structured commit messages (e.g., Conventional Commits) so that changes (e.g., features, fixes, and breaking changes) in a commit can be documented in a machine-parsable way. Some mature tools have been developed to assist in detecting compliance with conventions, such as `commitlink`.[117]
– Issues (6): many developers adopt the milestone mechanism on GitHub for progress tracking.[118] Some projects create each milestone using version numbers and group relevant issues into milestones,[119] which reduces the scope of review when developers write RNs.

Besides, the other four solutions propose that for each change description in RNs, RN producers can add links to the corresponding PR, issue, and commit, so that its completeness can be easily validated.[120]

4.2.5 Collaboration Management (22)

Collaboration is crucial in producing RNs because the process often requires the involvement of multiple developers. Within the leaf nodes of *Workflow Management*, *Who to Produce*, *Workflow Management*, and *PR/Issue/Commit Management*, 22 solutions aim to enhance communication and transparency among internal developers, ensuring the accuracy and efficiency of the RN production process.

– *Offer Guidances (15)*: Projects can provide guidance and templates to streamline the production of RNs, reducing the time and effort required to contribute. This includes instructions on creating and organizing RNs, as well as specifying the necessary information to be included.[121]
– *Clarify Responsibilities (4)*: Projects can clarify RN producers' responsibilities, such as editing, reviewing, and publishing RNs.[122,123] Some develop-

[115]`https://github.com/shipwright-io/build/pull/771`
[116]`https://github.com/pytorch/vision/issues/3351`
[117]`https://github.com/athensresearch/athens/issues/642`
[118]`https://docs.github.com/en/issues/using-labels-and-milestones-to-track-work/about-milestones`
[119]`https://github.com/kubernetes-sigs/multi-tenancy/issues/916`
[120]`https://github.com/elastic/observability-docs/issues/703`
[121]`https://github.com/govuk-react/govuk-react/issues/507`
[122]`https://github.com/kubernetes/test-infra/issues/9098`
[123]`https://github.com/kubernetes/release/issues/1889`

ers adopt pre-publishing the drafts of RNs to facilitate collaboration with other teams, such as translation efforts.[124]

- *Distribute Workloads (3)*: Collaboratively producing RNs among all contributors instead of relying on a centrally responsible developer for producing RNs.[125] For example, some projects require that each PR description include the RNs entry in its description, including the affected submodule name and a list of changes for that submodule.[126]

4.2.6 Link Standardization (19)

RN-related issues under *Wrong/Broken Links*, *Wrong/Broken Links to RNs*, and *Missing Links*, highlight the problem of incorrect or broken links that hinder users from accessing the intended websites. These issues can have a detrimental effect on the user experience and increase search costs, although they can be promptly resolved. To establish systematic standardization for link checking and management, developers have implemented three strategies to mitigate this problem:

- *Formatting Link (12)*: Implementing a consistent and fixed link format for RNs is effective in maintaining the validity and predictability of the links. For example, a URL ending with "/latest" always redirects to the most recent RN.[127] Developers also suggest providing an absolute path instead of a relative path[128] can reduce potential broken risks, e.g., in READMEs.
- *Periodical Verification (4)*: Checking invalid links regularly in a RN can mitigate this problem, e.g., a developer from `mantidproject/mantid` mentions that they need to go over the release and check links work before releasing.[129] Some developers also suggest a link checker should be introduced into the project to mitigate the problem.[130]
- *Centralized Management (3)*: Another strategy adopted is to maintain a centralized webpage, similar to `arxiv.org`, that contains the links or content for each RN.[131] This allows users to quickly browse through the RNs and access the desired information.

4.2.7 Synchronization Management (18)

The RN producers bear an additional burden in maintaining the consistency of RNs across multiple locations (such as GitHub Releases, project websites, etc.) and RNs with other documentation.[132] 18 solutions adopt automating

[124] `https://github.com/KurtBestor/Hitomi-Downloader/issues/3561`
[125] `https://github.com/edmcouncil/fibo/issues/964`
[126] `https://github.com/sympy/sympy/wiki/Writing-Release-Notes`
[127] `https://github.com/3drepo/3drepo.io/issues/2502`
[128] `https://github.com/semanticarts/gist/issues/422`
[129] `https://github.com/mantidproject/mantid/issues/31371`
[130] `https://github.com/opensearch-project/opensearch-build/issues/698`
[131] `https://github.com/cockroachdb/docs/issues/10963`
[132] `https://github.com/newrelic/docs-website/issues/2127`

this process to improve RN synchronization management by the following mechanisms:

– *Synchronous Distribution (16)*: Upload/push a final RN to various locations automatically, such as Telegram[133], and web pages[134]. Two off-the-shelf tools have been identified by developers to accomplish the task, i.e., `telegram-action`[135] and `Upload files to a Github release`[136].
– *Synchronous Verification (2)*: Enforce an synchronous check to ensure RNs' existence. The developers recommend monitoring the directory for the presence of RNs and providing timely alerts.[137]

4.2.8 Timely Delivery (17)

Our results in Section 3.2.4 indicate that developers frequently encounter difficulties in promptly creating and releasing RNs. They must strike a balance between their development tasks and the responsibility of documenting and communicating the changes. As a result, RNs may not be regularly updated, leading to confusion and frustration among users. 17 solutions propose two strategies that aim to improve the timeliness of RN production and prevent user disappointment:

– *Deadline Required (10)*: It motivates RN producers to prioritize the timely delivery of RNs by assigning a specific deadline[138] or announcing the adoption of a formal release cycle[139] to the users;
– *Sequence Required (7)*: It involves making it a mandatory step for developers to complete the RNs before the release of a new version[140] or simultaneously with the building of the release, which prioritizes the timely delivery and update of RNs alongside their development tasks.[141]

Summary: We identify eight categories of strategies to resolve the corresponding issues, including the identification of appropriate locations and channels for RNs (149), automation recommendations (128), enhancing the presentation of RNs (76), validating RN completeness through PR/issue/commit management (54), checking and managing links (19), improvement in collaboration management (22) and synchronization management (18) and establishing conventions for timely RN delivery (17).

[133]`https://github.com/wabarc/wayback/issues/48`
[134]`https://github.com/newrelic/node-newrelic/issues/590`
[135]`https://github.com/appleboy/telegram-action`
[136]`https://github.com/marketplace/actions/upload-files-to-a-github-release`
[137]`https://github.com/kubernetes/test-infra/issues/18309`
[138]`https://github.com/USAJOBS-temp/openoppstasks/issues/376`
[139]`https://github.com/girder/cookiecutter-girder-4/issues/45`
[140]`https://github.com/microsoft/ApplicationInsights-Java/issues/1682`
[141]`https://github.com/Dynatrace/dynatrace-configuration-as-code/issues/5`

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

Moreno et al. (2017)	Bi et al. (2020)
Fixed Bugs (90%)	Issues Fixed (79.3%)
New Features (46%)	New Features (55.1%)
New Code Components (43%)	System Internal Changes (25.1%)
Modified Features (26%)	Non-functional Requirements (10.3%)
Refactoring Operations (21%)	Documentation Updates (9.5%)
Ours (Completeness)*	**Ours (Correctness)***
Breaking Changes (23.36%)	Links (44.14%)
New Features (12.77%)	Version Information (14.48%)
Links (10.95%)	Dependency Specifications (11.03%)
Dependency Specifications (10.95%)	Identifiers (7.59%)
Migration/Usage Instructions (6.20%)	Code Examples (6.21%)

* The percentages here are different from the taxonomy because the denominators are the total number of issues in the Completeness (274 issues) and the Correctness (145 issues) category, respectively. In the Completeness column, issues from *Missing* and *Insufficient* are merged.

5 Discussion

5.1 Comparison with the Related Work

Since previous works categorize RN content into different taxonomies (Bi et al., 2020; Moreno et al., 2017), it is not easy to perform a 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 from the most frequently occurring RN content in different taxonomies (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 is 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. (2020) with a significant amount of new empirical evidence and actionable implications. For example, they find in their RQ2.2 that clear structure and the writing styles of RN 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.

Compared with our previous conference work (Wu et al., 2022), this paper expands the dataset of RN-related issues from the period between January 2021 and June 2021 to encompass the entire year 2021. The extended dataset includes an additional 620 RN-related issues, which proves valuable for comprehending the taxonomy and the associated strategies. Regarding the taxonomy, these extra issues contribute to a more comprehensive and in-depth examination in the following ways: 1) enrich the details of the leaf nodes within the taxonomy, such as introducing the *Image Formatting* leaf node under the *Poor Formatting* subcategory and the *Excessive Notification* leaf node within the *Accessibility* dimension; 2) confirm the saturation reached in the previous paper (only adding new leaf nodes). In the work of Wu et al. (2022), saturation is achieved in the third round with approximately 697 leaf nodes. In this paper, we randomize our data, repeat the labeling process, and reach saturation in the third round with a total of 1,171 leaf nodes. Thus, our experience offers valuable guidance for determining the number of labels among these range required for similar labeling tasks (Aghajani et al., 2019; Beyer et al., 2018; Chen et al., 2020; Tan and Zhou, 2019; Zhang et al., 2019a). In addition, this paper further explores the strategies to resolve these RN-related issues. Taking into account that not each issue provides a definitive solution (only one-third of the issues explicitly include the strategies in our dataset), the expanded dataset from the entire year of 2021 allows us to develop a comprehensive understanding of how these strategies contribute to issue resolution, with a particular focus on valuable strategies, such as the *Periodically Verification* strategy under Section 4.2.6.

5.2 Implications

5.2.1 Strategies Preference

Based on the results of RQ1 and RQ2, we will discuss what makes developers prefer some strategies over others in this section. Furthermore, we will also discuss that some mappings are not observed in Figure 7 and propose potential strategies to mitigate the associated challenges.

? *How to select suitable tools to automate RN generation?* A total of 26 tools (mentioned in Section 4.2.2 are being finally utilized by developers. In Figure 9, we have presented the fundamental details of the most frequent tools adopted by developers. To gain insights into the factors influencing their choices of RN generation methods, we have thoroughly examined the titles, descriptions, labels, and comments of each RN-related issue within the *Automation* category. Based on our analysis, we have identified the following factors that should be considered when selecting appropriate automated tools or scripts:

– *Feature*: While popular tools are able to automate the categorization and grouping of input sources,[142] developers may have specific requirements that these tools can not fulfill, such as supporting generating RNs for mul-

[142]`https://github.com/executablebooks/github-activity/issues/58`

tiple branches,[143] multiple projects,[144] and add supplement details for the changes.[145]

- *Cost*: Developers weigh the decision to adopt such tools by evaluating the additional costs introduced for all stakeholders involved in the software development process. They evaluate the costs of adopting such tools, including the need to learn new skills/technology and comply with certain practices.[146]
- *Integration*: Some developers prefer tools that can be integrated seamlessly with the existing workflow and development environment, rather than those that require extra steps or a completely new process. For example, one developer supports the adoption of `Semantic Release`, because *it works with react/next, and I guess everything related to js/ts.*[147]
- *Input*: The granularity and scope of input sources are important factors for developers to consider when choosing a tool, as they must determine which input sources are appropriate for their projects.[148]
- *Customization*: Custom scripts offer full control over the RN generation process and can be tailored to specific needs and requirements of the project, while existing tools are designed universally, limiting some customization features. For example, the developer who is not satisfied with popular generation solutions may choose to write a custom script for their project to avoid branch conflict and circumvent the need to label information.[149]
- *Configuration Friendliness*: Developers prioritize the ease of use and intuitiveness of RN generation tools.[150] Some tools, such as `Semantic Release`, provide an interactive plugin `semantic-release-cli`[151] that simplifies the configuration process.
- *Maintenance*: Community support for RN generation tools is an important factor in the tool selection for developers. For example, one developer intends to use the release features provided by GitHub for RN generation in order to reduce the maintenance burden rather than `github-activity`.[152]

Which are the most suitable locations to place RNs and manage them synchronously? Developers usually adopt the five following locations & channels for publicizing RNs: GitHub Release Pages, Project Websites, Files in Repositories, Apps, and Instant Messaging Channels. It is advisable to carefully consider the trade-offs of each location/channel for placing RNs:

[143]https://github.com/release-drafter/release-drafter/issues/844
[144]https://github.com/rfennell/ReleaseNotesAction/issues/131
[145]https://github.com/kubernetes/release/issues/1354
[146]https://github.com/dzcode-io/dzcode.io/issues/389
[147]https://github.com/dzcode-io/dzcode.io/issues/389
[148]https://github.com/cookpad/terraform-aws-eks/issues/183
[149]https://github.com/lightningnetwork/lnd/issues/6091
[150]https://github.com/jazzband/django-redis/issues/535
[151]https://github.com/semantic-release/cli
[152]https://github.com/executablebooks/github-activity/issues/58

1) In terms of facilitating collaboration and management, GitHub Release Pages and files in Repositories are convenient options. GitHub Release Pages provide a dedicated location for internal developers to edit and manage RNs on the website. Additionally, using RN files in repositories makes the repository more self-contained, allowing for collaborative editing with tools like Git. The OpenStack community, for example, requires the inclusion of RN files in projects to record version changes, as it enables working on multiple patches simultaneously and reduces merge conflicts; 2) Regarding accessibility, Project Websites, Apps, and Instant Messaging Channels are better choices. The official project website serves as a centre for showcasing and communicating RNs, which is easily discoverable through search engines. Embedding RNs within the application allows users to easily access and view the latest updates and changes while using the application, enhancing visibility and convenience. Sending RNs through instant messaging channels enables real-time communication of important updates and changes to developers and users, ensuring timely dissemination of information; 3) In terms of maintenance costs, Instant Messaging Channels, GitHub Release Pages, and Files in Repositories are better choices compared to the other options. Embedding RNs within the application and maintaining a project website may require additional development, integration and maintenance efforts to ensure the proper display and delivery of RNs.

In Section 4.2.7, we find that developers adopt the mechanisms of synchronous distribution and verification to maintain the consistency of RNs in different locations and auto-check the RN's existence. A better approach might be to involve combining the location strategies and synchronization management mechanisms to produce high-quality RNs.

❓ *How to improve the RN presentation?* An analogy to the RN presentation issues 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.[153,154]. Two main challenges, i.e., *Usability* and *Language*, in Section 3.2.2 provide the evidence that layout indeed greatly influences RN reading experience, as also mentioned by (Bi et al., 2020). Among the four strategies to relieve the RN presentation challenge, *Hierarchical Structure* provides a clear organization that allows users to quickly browse and selectively read specific sections based on their interests and needs. However, it requires careful design and maintenance of the structure to ensure that each section is clear and properly reflects the importance of the updates, imposing extra labour on RN producers. From these issues and their related RNs, we find that several hierarchical structures can be used to separate changes into categories and better organize RNs. Based on results in Section 3.2.2, 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

[153]`https://github.com/electron/electron/issues/28375`
[154]`https://github.com/EOSIO/eos/issues/9903`

change) (Wu et al., 2023). We also recommend putting the most important changes (e.g., breaking changes, major new features) on top. It is worth noting that our strategies match with the Information Architecture models (Li et al., 2017) that focus on effectively structuring, organizing, and labeling the content. *Standardized Format* improves consistency and comprehensibility of the information, while it also requires developers to adhere to and apply standardized formatting. *Rendering* and *Folding Mechanism* are great options to emphasise key changes and attract users' attention by using different formatting, e.g., icons.[155] Thus, after the structure is determined, it is recommended to employ a standardized format combined with proper visualization and folding lengthy lists to highlight important changes.

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 those of internal developers, we also 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 intended for the end users,[156] while the other is a list of all issues that have been resolved under this update intended for someone who needs technical details.[157]

? *Which strategy is more effective to ensure RNs' completeness?* The RN-related issues under *Completeness* category highlight the importance of ensuring comprehensive and reliable information in RNs. Our findings in Section 5.2.2 highlight two primary reasons for completeness problems: 1) insufficient resources or time to conduct thorough inspections; 2) difficulty for a limited number of developers to comprehend all changes between versions. Figure 9 demonstrates that tools adopt diverse management strategies for PRs, issues, and commits to ensuring completeness in RN production. PRs often provide a higher-level description of changes in a user-friendly manner, while commits are typically presented in a code-oriented fashion, making it challenging for non-technical users to understand and interpret each commit's meaning and impact. For projects that lack a strict collaboration and review process, it is better to combine these two types of information to ensure completeness. Issues contain detailed descriptions, reproduction steps, and solutions related to specific problems, making them suitable for feature-driven projects (Michlmayr, 2007). However, due to the abundance of information, only a portion of it is relevant to the RNs, which requires additional time to organize. If the project is configured with Jira, it is recommended to utilize Issues for generating RNs.[158]

[155] https://github.com/charleskorn/kaml/issues/138
[156] https://camel.apache.org/blog/2021/06/Camel311-Whatsnew/
[157] https://camel.apache.org/releases/release-3.11.0/
[158] https://support.atlassian.com/jira-cloud-administration/docs/create-release-notes/

?⃝ *Which strategy is appropriate for developers to produce RN timely?* in Section 4.2.8, we observe that developers typically employ two strategies to ensure the timely production of RNs: 1) establishing the sequence conventions, like mandate completion before the release; 2) setting clear deadlines or enforcing strict release cycles for RN production. The former fosters a sense of responsibility and ensures that the necessary preparation work for RNs is completed prior to the release. However, there is a risk that the release itself may be overlooked or delayed, despite having accompanying RNs; The latter involves time requirements. It provides a clear time limit and motivates developers to finish their work before the designated deadline, which is particularly suitable for projects that require scheduled releases, such as LibreOffice[159]. These two strategies are well-suited respectively for feature-driven and time-driven projects (Michlmayr, 2007). It is important to note that these two approaches can not coexist harmoniously (Rossi et al., 2009), that is, to release on time with the desired level of quality, a project may need to sacrifice certain features. Conversely, if a project aims to deliver a set of high-quality features, it may have to accept a delay.

?⃝ *How to effectively collaborate to produce and manage RNs?* We have identified three strategies to enhance collaboration, communication, and transparency among developers: offering guidance, clarifying responsibilities, and distributing workloads. The strategy of offering guidance provides a clear framework on how to effectively collaborate in order to produce RNs. By clearly defining each team developer's responsibilities and permissions, developers can gain a clear understanding of their roles within the collaborative process, promoting accountability and reducing confusion. Properly distributing workloads can reduce the pressure on RN producers before releasing new releases to the daily code reviews. In practice, a more effective approach may need to combine these strategies and adjust them based on the specific needs and circumstances of the project.

?⃝ *How to ensure that the link is valid?* Broken or incorrect links often lead to a negative user experience and increase the time and effort required for searching. It is effective to maintain a central website, adopt a fixed link format, and regularly check the validity of the links to mitigate this problem. However, they are not always silver bullets. Maintaining a dedicated website with RNs also introduces the challenge of link maintenance, which can temporarily disrupt access to RNs. Additionally, if the fixed link format needs to be updated in the future, it is necessary to modify existing links within all other documentation and websites.[160] Regularly checking for invalid links requires continuous effort and resources. Depending on the frequency of releases and the number of links involved, manually verifying all the links can become time-consuming. An effective solution is to combine these approaches into a link standardization process with guidelines for link creation, updates, and removal. This can be further facilitated by utilizing existing tools such

[159]`https://endoflife.date/libreoffice`

[160]`https://github.com/hedgedoc/hedgedoc/pull/1114/files`

as Xenu Link Sleuth[161], HTML Link Validator[162] and webmaster tools[163], to streamline the effort.

? *Why do we have missing strategies for some RN-related issues?* We find that there are some missing maps of relevant strategies and RN-related issues, which can be attributed to two factors: 1) certain RN-related issues have a relatively low number, like *Missing License Changes*. Consequently, there is a scarcity of specific strategies available for addressing these issues; 2) many of these issues can be easily resolved by the developers, such as by adding more comprehensive information to clarify the changes (like *Insufficient*) and fixing the bugs (like Language and Correctness) without mentioning specific strategies to alleviate or prevent such problems.

From Figure 3 - Figure 7, our framework of strategies effectively covers most issues within our taxonomy of RN-related issues, with the exception of many issues under the *Content* dimension. In Section 3.2.1, we have proposed strategies for completeness validation by efficiently managing PRs, issues, and commits to mitigate the problem of *Completeness*. However, it remains unclear what specific information should be included in RNs. We also find that other studies (Abebe et al., 2016; Bi et al., 2020; Moreno et al., 2017) show different distributions compared with the most frequent RN content identified in our results (Table 3), which indicates that some types of information are more likely to be missed or incorrect than others. Therefore, we recommend RN producers 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.

5.2.2 Automating RN Production

Apart from the tool-specific problems in Section 3.2.4, we further summarize the following research directions that may greatly help automate RN generation and resolve RN-related issues:

Automated Labeling of Software Changes. Our results in Section 3.2.4 show that many developers request tools to automate RN production. However, to the best of our knowledge, existing tools have strong constraints on

[161] `https://home.snafu.de/tilman/xenulink.html.`

[162] `https://html-link-validator.en.softonic.com/`

[163] `https://www.bigcommerce.com/ecommerce-answers/what-google-webmaster-tools/`

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 (e.g., Angular Commit Message Conventions). This enforces 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.[164] Techniques for automated commit or PR classification, which we consider as a promising direction, can alleviate this problem. Existing commit classification methods (e.g., Ghadhab et al. (2021); Levin and Yehudai (2017)) mainly focus on classifying commits into three maintenance categories (i.e., corrective, adaptive, and perfective) proposed by Swanson (1976), which is not suitable for RN generation. Therefore, classifying commits into categories suitable for RN generation is needed to facilitate automated RN generation. Similar discrepancies also exist for works on PR classification (Jiang et al., 2021b; Yu et al., 2018).

Automated Summarization and Language Style Unification. As reflected in Section 3.2.2, a fluent and unified writing style is vital to RN *Language*. 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*),[165] but also offloads the quality responsibility to developers writing other development texts, like PR titles. This often leads to poor readability of the final generated RNs. With advances in natural language preprocessing (NLP) tasks like text summarization (El-Kassas et al., 2021) and style transfer (Toshevska and Gievska, 2021), 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 3.2.1, *Completeness* and *Correctness* are the key to a high quality RN. The 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' content. 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, in Section 3.2.4). Similar works for, e.g., checking code comment inconsistency (Tan et al., 2007; Zhai et al., 2020), 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 3.2.1), it is vital to prioritize the detection of breaking changes and update incompatibilities (Lam et al., 2020).

[164] `https://github.com/opentelekomcloud/terraform-provider-opentelekomcloud/pull/1164`

[165] `https://docs.openstack.org/project-team-guide/release-management.html#how-to-add-new-release-notes`

5.3 Threats to Validity

5.3.1 Internal Validity

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

5.3.2 External Validity

Our work only uses data from GitHub projects for categorizing RN-related 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-related issues they have encountered. To mitigate the risk of generalizability, we investigate RN-related features provided by other mainstream code hosting platforms, such as GitLab[166] and Bitbucket[167], to evaluate the applicability of the taxonomy and strategies proposed in this paper. Due to the platform-specific features, the frequency of issues and strategies may vary on different platforms. For example, official RN generation functions and special tools that rely on GitHub Action features, such as Release Drafter, can not be used directly on GitLab and be recommended by developers on GitLab; While Bitbucket provides auto-generated features for RNs, it primarily relies on Jira for generating RNs based on issues, rather than using PRs on GitHub. These platform-specific features primarily affect issues and strategies related to *Production* dimension, such as *Request for Automation* and *PR/Issue/Commit Management.* However, their impact on issues and strategies related to *Content* and *Presentation* dimensions is limited. Our findings still provide valuable insights for ensuring the production of high-quality RNs and guiding the functional design of other platforms. Moreover, the limited number of developers also poses a threat, which we find 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 on a larger scale.

Moreover, the scarcity of RN-related issues poses a potential challenge to this paper. After applying filtering criteria, on average, there may be slightly more than one issue per year per project. However, several factors can help

[166] `https://docs.gitlab.com/ee/user/project/releases/`

[167] `https://support.atlassian.com/jira-cloud-administration/docs/create-release-notes/`

comprehend this threat: 1) variations in RN practices: not all software repositories have the convention of writing RNs. Table 1 indicates that a significant number of issues come from repositories with long development history, high popularity, and active development activities; 2) specific audience of RNs: RNs primarily target users who rely on the project. These users consult RNs to understand incremental changes and assess the potential impact on their software. However, developers may not read or comprehend all the changes, making it inherently challenging to identify issues. Furthermore, not each problematic change will affect developers or be discovered by them; 3) frequency of software releases: the number of software releases per year tends to be relatively low, and the release cycles are often lengthy. For example, in our dataset, `PyTorch/PyTorch` only released seven versions in 2021;[168] We also collected issues related to similar artifacts in software development (Aghajani et al., 2019), such as "migration guides", "API references", and "user guides", by using the method described in Section 3.1.1. The respective numbers of issues for these artifacts were 635, 747, and 535 in 2021, indicating a similar scarcity in relation to RN-related issues. While acknowledging the relatively low frequency of RN-related issues, it is crucial to recognize their significance. Even a small number of issues can have a substantial impact if they affect critical features or introduce vulnerabilities, such as *Missing Breaking Changes* within the Content dimension. The rarity of these occurrences does not diminish the importance of RN-related issues; instead, it highlights the necessity for diligent attention and proactive measures to ensure the quality and accuracy of RNs, which is also further supported by our findings in Section 3.2.

Another threat to external validity comes from using only issues with the 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 which is comparable to and even larger than existing studies (Aghajani et al., 2019; Beyer et al., 2018; Chen et al., 2020; Tan and Zhou, 2019; Zhang et al., 2019a).

6 Conclusion

In this paper, we manually analyze 1,529 latest RN-related issues on GitHub and construct a taxonomy of real-world RN-related issues with four dimensions and a framework with eight topics of strategies for resolving these challenges. Our results demonstrate the gap for RN production and provide a research roadmap for further improvement that we expect the community may benefit from. In future work, we plan to investigate such opportunities for integrating novel automation approaches with existing RN workflows.

[168] `https://github.com/pytorch/pytorch/releases`

7 Data Availability

A replication package includes all supplementary materials, which can be found at `https://figshare.com/s/b86957f784a8c872c042`. This package includes:

– An Excel spreadsheet contains the complete issues and all of our analysis about the RN-related issues;
– The scripts we used to collect, analyze, and visualize the data;
– A markdown contains the practitioner-oriented checklist for producing a high-quality RN.

Acknowledgements This work is supported by the National Natural Science Foundation of China Grant 61825201 and 62142201.

References

Abebe SL, Ali N, Hassan AE (2016) An empirical study of software release notes. Empirical Software Engineering 21(3):1107–1142, DOI 10.1007/s10664-015-9377-5

Aghajani E, Nagy C, Vega-Márquez OL, Linares-Vásquez M, Moreno L, Bavota G, Lanza M (2019) Software documentation issues unveiled. In: 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), IEEE, pp 1199–1210, DOI 10.1109/ICSE.2019.00122

Aghajani E, Nagy C, Linares-Vásquez M, Moreno L, Bavota G, Lanza M, Shepherd DC (2020) Software documentation: the practitioners' perspective. In: 2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE), IEEE, pp 590–601, DOI 10.1145/3377811.3380405

Ahire JB (2021) Issue #156 of hypertrace/hypertrace. `https://github.com/hypertrace/hypertrace/issues/156`

Alali A, Kagdi H, Maletic JI (2008) What's a typical commit? a characterization of open source software repositories. In: 2008 16th IEEE International Conference on Program Comprehension, IEEE, pp 182–191, DOI 10.1109/ICPC.2008.24

Beyer S, Macho C, Di Penta M, Pinzger M (2018) Automatically classifying posts into question categories on stack overflow. In: 2018 IEEE/ACM 26th International Conference on Program Comprehension (ICPC), IEEE, pp 211–21110, DOI 10.1145/3196321.3196333

Bi T, Xia X, Lo D, Grundy J, Zimmermann T (2020) An empirical study of release note production and usage in practice. IEEE Transactions on Software Engineering DOI 10.1109/TSE.2020.3038881

Chen Z, Cao Y, Liu Y, Wang H, Xie T, Liu X (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, pp 750–762, DOI 10.1145/3368089.3409759

Coelho R, Almeida L, Gousios G, Van Deursen A (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, pp 134–145, DOI 10.1109/MSR.2015.20

El-Kassas WS, Salama CR, Rafea AA, Mohamed HK (2021) Automatic text summarization: A comprehensive survey. Expert Systems with Applications 165:113679, DOI 10.1016/j.eswa.2020.113679

Ghadhab L, Jenhani I, Mkaouer MW, Ben Messaoud M (2021) Augmenting commit classification by using fine-grained source code changes and a pre-trained deep neural language model. Information and Software Technology 135:106566, DOI https://doi.org/10.1016/j.infsof.2021.106566, URL `https://www.sciencedirect.com/science/article/pii/S0950584921000495`

He H, He R, Gu H, Zhou M (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, pp 478–490

Holloway FW (1985) Praxis release notes, versions 7. 4 and 7. 5 URL `https://www.osti.gov/biblio/5141606`

Hove SE, Anda B (2005) Experiences from conducting semi-structured interviews in empirical software engineering research. In: 11th IEEE International Software Metrics Symposium (METRICS'05), IEEE, pp 10–pp

Humbatova N, Jahangirova G, Bavota G, Riccio V, Stocco A, Tonella P (2020) Taxonomy of real faults in deep learning systems. In: Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering, pp 1110–1121, DOI 10.1145/3377811.3380395

Jiang H, Zhu J, Yang L, Liang G, Zuo C (2021a) Deeprelease: Language-agnostic release notes generation from pull requests of open-source software. In: 2021 28th Asia-Pacific Software Engineering Conference (APSEC), IEEE, pp 101–110, DOI 10.1109/APSEC53868.2021.00018

Jiang J, Wu Q, Cao J, Xia X, Zhang L (2021b) Recommending tags for pull requests in github. Information and Software Technology 129:106394, DOI https://doi.org/10.1016/j.infsof.2020.106394

Kamezawa H, Nishida N, Shimizu N, Miyazaki T, Nakayama H (2022) Rnsum: A large-scale dataset for automatic release note generation via commit logs summarization. In: Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp 8718–8735

Klepper S, Krusche S, Bruegge B (2016) Semi-automatic generation of audience-specific release notes. In: 2016 IEEE/ACM International Workshop on Continuous Software Evolution and Delivery (CSED), IEEE, pp 19–22

Lam P, Dietrich J, Pearce DJ (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, pp 157–179, DOI 10.1145/3426428.3426922

Levin S, Yehudai A (2017) Boosting automatic commit classification into maintenance activities by utilizing source code changes. Association for

Computing Machinery, New York, NY, USA, PROMISE, p 97–106, DOI 10.1145/3127005.3127016

Li M, Gao R, Hu X, Chen Y (2017) Comparing infovis designs with different information architecture for communicating complex information. Communication Design Quarterly Review 5(1):43–56

Maalej W, Happel HJ (2010) Can development work describe itself? In: 2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010), IEEE, pp 191–200, DOI 10.1109/MSR.2010.5463344

Michlmayr M (2007) Quality improvement in volunteer free and open source software projects: exploring the impact of release management. PhD thesis, University of Cambridge

Moreno L, Bavota G, Penta MD, Oliveto R, Marcus A, Canfora G (2017) Arena: An approach for the automated generation of release notes. IEEE Transactions on Software Engineering 43(2):106–127, DOI 10.1109/TSE.2016.2591536

Nath SS, Roy B (2021) Towards automatically generating release notes using extractive summarization technique. In: International Conference on Software Engineering & Knowledge Engineering, SEKE, pp 241–248

Nath SS, Roy B (2022) Exploring relevant artifacts of release notes: The practitioners' perspective. In: IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2022, Honolulu, HI, USA, March 15-18, 2022, IEEE, pp 1270–1277, DOI 10.1109/SANER53432.2022.00152, URL `https://doi.org/10.1109/SANER53432.2022.00152`

Olsson HH, Bosch J (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, pp 9–16, DOI 10.1109/SEAA.2014.75

Rossi B, Russo B, Succi G (2009) Analysis of open source software development iterations by means of burst detection techniques. In: Boldyreff C, Crowston K, Lundell B, Wasserman AI (eds) Open Source Ecosystems: Diverse Communities Interacting, 5th IFIP WG 2.13 International Conference on Open Source Systems, OSS 2009, Skövde, Sweden, June 3-6, 2009. Proceedings, Springer, IFIP Advances in Information and Communication Technology, vol 299, pp 83–93, DOI 10.1007/978-3-642-02032-2_9, URL `https://doi.org/10.1007/978-3-642-02032-2_9`

Seaman CB (1999) Qualitative methods in empirical studies of software engineering. IEEE Transactions on software engineering 25(4):557–572

Shihab E, Ihara A, Kamei Y, Ibrahim WM, Ohira M, Adams B, Hassan AE, Matsumoto Ki (2013) Studying re-opened bugs in open source software. Empirical Software Engineering 18(5):1005–1042, DOI 10.1007/s10664-012-9228-6

Swanson EB (1976) The dimensions of maintenance. In: Proceedings of the 2nd International Conference on Software Engineering, IEEE Computer Society Press, Washington, DC, USA, ICSE '76, p 492–497

Tan L, Yuan D, Krishna G, Zhou Y (2007) /* icomment: Bugs or bad comments? */. In: Proceedings of 21st ACM SIGOPS Symposium on Operating

Systems Principles, pp 145–158, DOI 10.1145/1294261.1294276

Tan X, Zhou M (2019) How to communicate when submitting patches: An empirical study of the linux kernel. Proceedings of the ACM on Human-Computer Interaction 3(CSCW):1–26

Toshevska M, Gievska S (2021) A review of text style transfer using deep learning. IEEE Transactions on Artificial Intelligence pp 1–1, DOI 10.1109/TAI.2021.3115992

Wu J, He H, Xiao W, Gao K, Zhou M (2022) Demystifying software release note issues on github. In: 2022 IEEE/ACM 30th International Conference on Program Comprehension (ICPC), pp 602–613, DOI 10.1145/3524610.3527919

Wu J, Xu W, Gao K, Li J, Zhou M (2023) Characterize software release notes of github projects: Structure, writing style, and content. In: Zhang T, Xia X, Novielli N (eds) IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2023, Taipa, Macao, March 21-24, 2023, IEEE, pp 473–484, DOI 10.1109/SANER56733.2023.00051, URL https://doi.org/10.1109/SANER56733.2023.00051

Yang AZ, Hassan S, Zou Y, Hassan AE (2021) An empirical study on release notes patterns of popular apps in the google play store. Empirical Software Engineering pp 1–41

Yang AZH, Hassan S, Zou Y, Hassan AE (2022) An empirical study on release notes patterns of popular apps in the google play store. Empir Softw Eng 27(2):55, DOI 10.1007/s10664-021-10086-2, URL https://doi.org/10.1007/s10664-021-10086-2

Yu L (2009) Mining change logs and release notes to understand software maintenance and evolution. CLEI Electron Journal 12(2):1–10

Yu S, Xu L, Zhang Y, Wu J, Liao Z, Li Y (2018) Nbsl: A supervised classification model of pull request in github. In: 2018 IEEE International Conference on Communications (ICC), pp 1–6

Zhai J, Shi Y, Pan M, Zhou G, Liu Y, Fang C, Ma S, Tan L, Zhang X (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, pp 25–37

Zhang T, Gao C, Ma L, Lyu M, Kim M (2019a) An empirical study of common challenges in developing deep learning applications. In: 2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE), IEEE, pp 104–115, DOI 10.1109/ISSRE.2019.00020

Zhang Y, Zhou M, Mockus A, Jin Z (2019b) Companies' participation in oss development–an empirical study of openstack. IEEE Transactions on Software Engineering 47(10):2242–2259