Characterize Software Release Notes of GitHub Projects: Structure, Writing Style, and Content

Jianyu Wu*, Weiwei Xu*, Kai Gao*, Jingyue Li†, Minghui Zhou*‡
*School of Computer Science and School of Software & Microelectronics, Peking University, Beijing, China
*Key Laboratory of High Confidence Software Technologies, Ministry of Education, Beijing, China
†Norwegian University of Science and Technology, Trondheim, Norway
Email: {wujianyu, gaokai19, zhmh}@pku.edu.cn, xuww@stu.pku.edu.cn, jingyue.li@ntnu.no

***Abstract*—Release notes (RNs) summarize important changes between two successive software releases to facilitate software upgrades and serve as means of communication between software and its users. However, existing research has shown that many users cannot extract the information they want from RNs effectively and efficiently due to poor structure and insufficient content. Many efforts have been devoted to categorizing documented information in RNs, however, how exactly RNs are organized, in what way RNs are written, and what is written in RNs with respect to project domains and release types remain under investigation. To bridge this knowledge gap, we manually analyzed 612 RNs from 233 top popular GitHub projects to characterize their Structure, Writing Style, and Content. We find 64.54% of RNs organize changes into hierarchical structures following three strategies, i.e., by Change Type, Affected Module, or Change Priority. And 11.60% of RNs adopt multiple strategies to present the changes. Among the three strategies to organize changes, by Change Type is mostly adopted. RNs of major releases and System Software are more likely to organize changes by Affected Module. We also find three types of Writing Styles: Expository, Descriptive, and Persuasive with increasing explanation information and manual effort, taking 30.07%, 34.80%, and 35.13% of RNs, respectively. 83.10% of RNs in System Software projects adopt the Descriptive and Persuasive writing style. For Content, we find System Software and Libraries & Frameworks projects are more likely to record Breaking Changes, while Software Tools projects emphasize Enhancements. Besides, Fixed Bugs and Security Changes are less common in RNs of major releases. Our findings not only serve practitioners with a roadmap for customizing high-quality RNs but also shed light on future research on automating RN.**

***Index Terms*—release note, release engineering, characteristic, empirical study**

I. Introduction

When releasing a new version of software, release notes (RNs) often serve as one of the most essential artifacts that summarize the main changes between two consecutive releases [1], such as what bugs are fixed and what new features are introduced. In this sense, RNs serve as a communication channel between RN producers and various stakeholders, such as end-users and other internal software developers, to deliver key information for software upgrades. In practice, stakeholders usually refer to RNs to understand what has been done in the new release. For example, end-users read RNs for new features that may improve their user experience. Downstream developers also refer to RNs to comprehend potential beneficial and interrupting changes in the new release. Internal developers sometimes treat RNs as a checklist to review completed and pending tasks, e.g., known issues [2].[1]

With the wide adoption of "release early, release often" software development philosophy [3], the pace of producing RNs becomes faster but the process becomes more challenging [4]. Research has shown that producing RNs is a complicated, strenuous, and time-consuming process [1], [4]. For example, the operating system openEuler has to deal with about 15,000 commits per month [5]. Surveys conducted by Moneno et al. [1] and Bi et al. [4] both reveal that producing RNs usually takes more than four hours. Moreover, despite great efforts devoted to producing RNs, the RNs may be of poor quality, e.g., missing important changes or being poorly organized [2]. Some developers warn that each additional sentence loses 10% of the readers, and it is difficult to maintain a balance between providing sufficient information and not becoming too lengthy [6]. Unfortunately, there is no uniform or widely recognized standard and guidelines to produce RNs [1].

To tackle these challenges, some efforts [1], [4], [7] have been devoted to understanding what information RNs contain in general. They summarize several types of information, such as issues fixed and system internal changes. A recent work conducted by Wu et al. [2] investigates the issues related to RNs on GitHub. They find that (1) RNs suffer from poor structure, resulting in extra costs to find information in need, such as burying important new features and breaking changes; (2) RNs also have issues with writing style, reducing clarity and understandability. However, the best practices of organizing RNs (i.e. structure) and style of writing RNs (i.e. writing style) remain uninvestigated. Moreover, due to the different nature of projects in different domains and different kinds of releases (i.e., major, minor, and patch), how the structure, writing style, and content vary with project domains and release types is worthy of study. Achieving such an understanding would help to produce RNs meeting developers' needs in a more targeted way and further shed light on prospective improvements in customizing RN production automatically.

Therefore, we set out to characterize the state of the practice

‡ Corresponding Author

[1]In this paper, we use the term "user" to refer to anyone who reads RNs or uses RNs.

for Structure, Writing Style, and Content of different RNs across software domains and release types. To that end, we collect 612 RNs from the latest major, minor, and patch releases of 233 top popular projects from GitHub. Our analysis consists of two steps: (1) we use open coding on the 612 RNs to derive the structure and writing style of RN. For the content, we follow the category provided by the latest work based on GitHub projects [2] and check the existence of each category in these RNs; (2) we compare the distribution of structure, writing style, and content for each project domain and release type to understand how different domains of projects and different types of releases produce RNs. The key findings are:

- **Structure**: we find that 64.54% of RNs organize changes into hierarchy following three strategies: by *Change Type*, *Affected Module*, or *Change Priority*. 11.60% of RNs adopt multiple strategies to organize the changes. Projects in all domains mostly organize changes by *Change Type*. RNs of major releases and system software are more likely to organize changes by *Affected Module*.
- **Writing Style**: we find three types of Writing Styles: *Expository*, *Descriptive*, and *Persuasive* with increasing manual involvement, taking 30.07%, 34.80%, and 35.13% of RNs respectively. 83.10% of RNs of System Software projects adopt *Descriptive* and *Persuasive* style.
- **Content**: we find that RNs of System Software and Libraries & Frameworks projects record *Breaking Changes* more frequently, while Software Tools projects emphasize on *Enhancements*. RNs of major releases contain fewer *Fixed Bugs* and Security than RNs of minor releases.

Based on the results, we provide a roadmap to customize RNs of appropriate content, structure, and writing style across the project domain and release type and discuss how far we are to generate well-compiled RNs automatically. We provide a replication package at doi.org/10.6084/m9.figshare.21383280

II. Background and Related Work

There are many important artifacts in software development [8], among which, release notes (RNs) are used to summarize the major changes in the software since its previous release. README [9] and user guide [10] are also two types of important documentation available to users. In practice, users are advised to check the README first to determine whether the software meets their needs and to obtain basic information about it. If so, they then refer to the user guide for instructions on how to use the software. When a new version of software is released, users consult RNs to understand the incremental information and determine whether the new changes will have an impact on their software [7].

Since switching the release pattern from "once and for all" in the 20th century to "release early, release often" in the 21st century, many software companies have attempted to shorten their release cycles and speed up the delivery of their latest innovative products to users [11]. This results in a tight feedback loop between developers and users, posing new challenges for the production of high-quality RNs. For example, Firefox, a famous open source browser with worldwide users [12], often contains thousands of patches within one release cycle, leading to the challenge in producing RNs [13] Many researchers begin to study RNs from two major aspects: empirical studies to understand RN practices and approaches to automate the generation of RNs.

A. Empirical Studies of Release Note practices

Existing literature about software RN practices primarily focuses on two aspects: (1) summarizing information categories in RNs and related artifacts about RNs; (2) demystifying challenges in the production and usage of RNs.

For the first aspect, Moreno et al. [1] manually examine 990 release notes from 55 open source projects to analyze and categorize their content into 17 types, such as fixed bugs, new features, and modified code components. Abebe et al. [7] manually analyze 85 release notes across 15 different software systems and identify six different types of information, such as caveats and problems. They also examine the scope of issues listed in RNs, i.e., all issues or selected issues are included in RNs, and discover that the majority of RNs list only a limited number of issues. Bi et al. [4] study the content of 32,425 RNs from 1,000 GitHub projects and categorize common RN content into eight topics, including bug fixes, internal system changes, etc. They find that RN content varies across domains of software, for example, for application software and system software, new features are the most frequently documented. Nath et al. [14] analyze relevant artifacts of 3,347 RNs and find that key artifacts include issues, PRs, commits, and CVEs. Besides, Yang [15] collect 69,851 RNs of popular apps on the Google Play Store. Combined with surveys and user reviews, they reveal six patterns of RNs, for example, Short updating steady and Short non-updating steady. For the second aspect, Aghajani et al. [8] investigate the importance of different kinds of documentation by survey. They find that RN absences are common, and suggest that RN producers should include RNs on the release checklist as a mandatory item. Wu et al.[2] further explore how RNs go wrong or fail to meet users' expectations. They manually analyze 1,731 GitHub issues to build a comprehensive taxonomy of RN issues with four dimensions: Content, Presentation, Accessibility, and Production.

However, we still lack a comprehensive empirical understanding of how exactly software RNs are organized, in what way RNs are written, and what is written in RNs across different project domains and release types, which is critical to further customize RNs.

B. Release Note Automation Tools

To reduce the manual effort involved in the production of RNs, prior research has explored automated methods for generating RNs. Klepper et al. [16] propose a semi-automated RN generation tool based on information gathered from both build server and issue tracker which can tailor RNs to specific audiences' needs. Moreno et al.[1] provide ARENA that integrates both change information from version control systems and rationale information from issue trackers into RNs with predefined categories. Ali et al. [17] extract code changes

by git diff between two versions, generate natural language summaries for changes, link related issues to changes, and output a RN document by Doc Generator. Nath et al. [18] propose a method which generates RNs from commit messages and PRs by integrating the GloVe word embedding technique with TextRank. Jiang et al. [19] propose an approach called DeepRelease by formulating change entry generation as text summarization tasks and change categorization (e.g., new features and bug fixes) as multi-class classification tasks. Recently, Kamezawa et al. [20] construct a dataset called RNSum which contains approximately 82,000 English release notes and associated commit messages. They also propose two deep learning-based approaches to generate RNs with unlabeled commits.

Furthermore, many automated RN generation tools have been developed to facilitate the generation of RNs since the year 2014, such as Semantic Release, [21], github-changelog-generator [22], Release It [23] and Release Drafter [24]. However, all tools require that each change should be documented using predefined templates or labels in order to generate RNs based on predefined criteria, which is a laborious task for contributors and maintainers. RN producers may still post-edit the generated RNs to improve their readability, for example by summarizing the changes.

However, most of these approaches ignore the differences between different project domains and release types, resulting in a lack of customization of RN production. Our work aims to understand the characteristics of the structure, writing style, and content of RNs with respect to project domains and release types, which help produce RNs that are better tailored to developers' needs as well as shed light on future improvements in automating the production of customized RNs.

III. Methodology

A. Data Collection

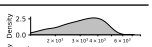
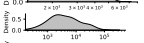
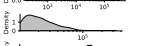
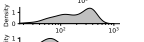
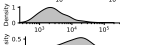
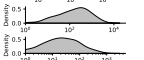
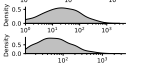
1) Project Selection: To formulate good practices regarding Structure, Writing Style, and Content of RNs across different domains and release types, we carefully select projects with the following criteria via GitHub API:

- The project should be created before July 1st, 2019 (three years ago) and have at least one commit in 2022 to ensure the activeness;
- The project should have more than ten releases to filter out those projects without RNs and inexperienced ones;
- The project should have more than 10k stars to ensure high popularity and representativeness.

The purpose of collecting data from GitHub is twofold: (1) GitHub hosts the largest collection of open source software in the world with over 83 million developers and 200 million repositories [25]. (2) GitHub provides developers with "Release Pages" to conveniently exhibit RNs to users.

We collect a total of 955 projects fulfilling the above criteria. To make manual effort affordable, we further randomly select 274 projects from these projects based on a 95% confidence level and a 5% confidence interval, followed by previous work [26], [27], [28], [29]. Then, we browse through their GitHub repository pages as an initial familiarization and manually exclude 28 projects that are non-software projects, e.g., OpenAPI-Specification [30], and have released multiple versions but no release notes. Finally, we obtain 246 projects.

TABLE I
Project Statistics of the Dataset for RNs

	Mean	Median	Std.	Distribution
Age (in Days)	3,078.37	3,060.00	963.90	
# of Commits	8,044.40	3,365.0	11990.78	
# of Stars	25,357.79	19,488.00	21,007.73	
# of Contributors	222.85	211.0	134.63	
# of Forks	4,609.06	2,344.00	7,828.34	
# of Issues	403.97	189.00	576.35	
# of PRs	53.68	21.00	91.60	
# of Releases	98.50	58.00	125.75	

2) Release Note Selection: There are different types of releases, e.g., major and minor releases [31]. To inform users of release types, many versioning schemes are proposed, among which, Semantic Versioning (also known as SemVer) [32] is increasingly adopted by software projects [33]. So, in this paper, we rely on SemVer to identify release types. SemVer specifies the version number denoted as $x.y.z$, where $x.0.0$ as a Major release, $x.y.0$ ($y \neq 0$) as a Minor release, and $x.y.z$ ($z \neq 0$) as a Patch release. We filter out the projects and their RNs that do not follow the SemVer versioning schema. There are, however, some releases that follow the format, but they may not adhere to the rule of denoting different types of releases. For example, the version 0.4.9 of ArchiveBox is actually a major release as stated in the RN. To mitigate this problem, we go through the RN content of each release to check whether there is any explicitly declared release type information. If not, we label its type based on SemVer.

Afterwards, we collect the most recent RNs of each project's major, minor, and patch releases because we consider the most recent RN can reflect the current state of the software RN management practices. To ensure the quality of RN, the first author browses through these 246 GitHub projects' release pages and excludes 87 RNs written in non-English languages, of unidentifiable release type, or containing little information about the changes. Besides, some projects have not yet released their major versions, e.g., vivus.js [34] and only produced RNs for specific versions, e.g., sentry [35]. The final dataset in our study consists of 612 RNs from 233 projects, including 162 major releases, 227 minor releases, and 223 patch releases. These projects cover over 10 types of programming languages and the repository statistics are summarized in Table I. We can observe a long-tail distribution in the metrics, which is expected and common in many mining software repository researches [36], [2], [37].

Since RNs summarize changes between each collected release with their previous releases, we also clone their repository and use PyDriller [38] to get the number of commits to gain a deeper understanding of release characteristics.

B. Data Analysis

To achieve our research goal, we classify the obtained projects into different domains, categorize the structure, writing style, and content of their RNs, and make a comparison of the RNs across software domains and release types.

1) Domain Classification: Researchers have found that the documented information in RN varies across domains [4], which motivates us to further understand the variation of RN characteristics across project domains. Borges et al. [39] collected 2,500 projects on GitHub and manually classified these projects into six domains: (1) Application Software, (2) System Software, (3) Web Libraries & Frameworks, (4) Non-web Libraries & Frameworks, (5) Software Tools, (6) Document. However, they did not provide a detailed process of the classification or coding guide. Therefore, we first sample 30 projects from each domain (except the Document domain) provided by the previous work [39] to be familiar with their classification convention. Two authors, both with over seven years of software development experience, individually label these projects into the other five domains by reading their READMEs, repository labels and the first ten pages returned by Googling their names. They then compare their labeling results with theirs [39]. They find that the differences mainly concentrate on the projects of "Web Libraries & Frameworks", and "Non-web Libraries & Frameworks" since there is no clear definition of what is "web". As a result, they merge the two domains into "Libraries & Frameworks", and the final four software domains are:

- System Software: software that offers basic services and infrastructure to other software, e.g., operating systems, servers, and databases.
- Libraries & Frameworks: software that provides a collection of reusable functionalities to facilitate software development in specific domains such as Web and machine learning.
- Software Tools: software that facilitates developers with universal software development tasks, like IDEs and compilers.
- Application Software: software that offers end-users with functionality, such as browsers and text editors.

Then the two authors individually classify the 233 projects into four domains. We use Cohen's Kappa (k) to measure the inter-rater agreement between two authors and the k value is 0.83, indicating a high agreement. For the disagreed results, the authors discuss thoroughly until reaching an agreement. The number of projects under Libraries & Frameworks, Application Software, Software Tools, and System Software is 91, 69, 45, and 28, respectively.

2) Categorization of Structure, Writing Style, and Content: We next describe how we identify the categories of structure, writing style, and content of RNs.

Structure aims to investigate how RNs are organized, and writing style focuses on in what way producers use available information to describe changes in RNs. To the best of our knowledge, there is no prior work systematically investigating the two characteristics. While for content, existing work has proposed several content categories. In this study, we choose to follow the category proposed by Wu et al. [2] since their category is based on the practical issues proposed by producers and users on GitHub. Their category consists of eight types of change content: (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.

Therefore, we choose to conduct qualitative manual labeling which consists of two parts:

- A pilot study to derive the types of structure and writing style and familiarize the eight change content types proposed by Wu et al. [2].
- An extended study to identify and classify the structure, writing style, and content of the remaining RNs based on the results of a pilot study.

Pilot Study. We randomly sample 183 (30%) from the 612 RNs for a pilot study. The first two authors, named inspectors, participate in the pilot study. They first read the RNs to get familiarized with them. Then they independently develop labels to describe the structure and writing style of RNs and get familiarized with the eight content types proposed by Wu et al. [2]. Specifically, for Structure, since RNs on GitHub pages are written in Markdown, they independently go through each level of headings in a top-down manner to understand what strategies producers adopt to organize content. If the RN has no heading level, that is, just stacks them up, the structure of the RN is labeled as *Plain List*. For Writing Style, they compare each change with the content of related commit/PR/issue like Abebe et al. [7] to investigate how producers use available information to describe changes in RNs. For Content, they first read the code book and the issues related to RN content provided by Wu et al. [2] respectively. Then for each change in these RNs, they review the titles, descriptions, labels, changed codes, and comments of each related commit/PR/issue to understand what types of category this change refers to. The above process is iterative, and the third author is included as an arbitrator to (1) discuss with the two inspectors about their label for the structure and writing style until an agreement is reached to produce the final codes; (2) mediate, discuss, and resolve any disagreement regarding the labeling results for the content.

Extended Study. Based on the initial categories in Section III-B2, the inspectors iteratively conduct independent labeling for remaining RNs. We follow the work [2] to set another three rounds (143 RNs for each round) to analyze the results. If a RN's writing style or structure cannot be classified into an existing category, this code will be added into a temporary *Pending* category. Each round is followed by a meeting among the inspectors and arbitrator to (1) resolve labeling conflicts and determine the final label for the structure, writing style, and content; (2) justify whether new categories should be added for codes in the *Pending* category. By the end of each round, we also use Cohen's Kappa (k) to measure inter-rater agreement between two inspectors and

the values for Content range from 0.76 to 0.92[2], for Structure from 0.87 to 0.92, and for Writing Style from 0.74 to 0.82, indicating increasing and high agreement. Note that (1) a RN may adopt multiple strategies to organize its content and will be assigned multiple strategies; (2) If a change refers to multiple types of content, e.g., both *Fixed Bugs* and *Breaking Changes*, it will be assigned multiple labels.

C. Developer Interview

To validate our categories for the structure and writing style, we conduct semi-structured interviews with two industry software engineers (named *interviewee A* and *interviewee B* respectively) from famous IT companies. The two developers both have rich experience in publishing and using RNs, especially that *interviewee A* is the core developer responsible for producing RNs within the team. In order to facilitate a better interaction, both interviews are conducted face-to-face by two authors (one is the leader and the other asks additional questions as needed) [40], [41] and take 45 minutes and 1 hour and 21 minutes respectively. Following the process [41], two interviews begin with the question *"What structure did you use to organize the changes and in what way did you describe the changes in your software development process?"*. Then, we present our categories of the structure strategy and writing style and ask another question *"What are your thoughts on the following structure strategies and writing styles? Have you seen them or considered using them?"* These open-ended questions are intended to assess the coverage and representativeness of our categories. First, *Interviewee A* replies that they are imitating how they organize and describe the changes in their well-known similar projects. *Interviewee B* shows us the RNs of their project and explains that as their project is in the initial stage, they primarily focus on development and only use the *Plain list* to announce significant changes. In summary, they consider that our categories are clear and informative and cover the structure and writing styles of RNs they have encountered.

We finally derive four strategies of structure and three types of writing styles. It takes more than six weeks to complete the manual labeling process.

IV. Results

A. Structure

In this section, we illustrate how different domains of projects and different types of releases structure their RNs. Following the open coding process described in Section III-A2, we categorize four mainstream strategies of RN structures, i.e., *Plain List*, or a hierarchical list by *Change Type*, by *Affected Module*, and by *Change Priority*, whose corresponding descriptions are shown in Table II. Furthermore, the table shows the percentage of RNs adopted by major, minor, and patch releases for each strategy.

As we can see in Table II, 35.46% of the studied RNs are structured as *Plain List*, while the remainder is structured as hierarchical lists. Among the three strategies for hierarchical lists, by *Change Type* is the most frequently adopted, accounting for more than half of all studied RNs. It is possibly because organizing changes by their content is a common practice supported by many automated tools, e.g., Release Drafter [24], and specifications, e.g., the well-known Angular Conventional Commits [42]. Contrarily, only 15.69% and 6.70% of all the studied RNs are organized by *Affected Module* and *Change Priority*, respectively. Interestingly, we also observe 71 (11.60%) RNs that adopt multiple strategies simultaneously. Among the 71 RNs adopting multiple strategies which we refer as *Hybrid*, 51 RNs organize changes by both *Change Type* and *Affected Module*, 18 RNs by *Change Type* and *Change Priority* strategies, and only two RNs by *Affected Module* and *Change Priority* strategies. When changes are organized into a multiple-level list, the outer and the middle level are usually organized by different strategies, and the inner level specifies concrete changes. For example, a native GraphQL database with graph backend, adopts *Change Type* strategy and *Affected Module* strategy in the outer two levels, respectively, to structure changes in RN for the minor version 21.12.0 [43].

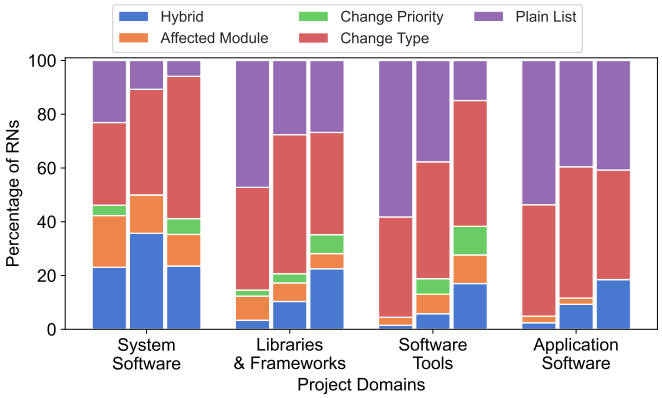


Fig. 1. Distribution of Structure under four domains for release types. Bars from left to right represent Patch, Minor, and Major releases respectively. Hybrid represents multiple strategies.

Different strategies are preferred for the types of releases. Specifically, for patch releases, nearly half (48.88%) of RNs adopt plain list strategy to organize their changes, while RNs of minor and major releases adopt more hierarchical strategies, especially the *Change Priority* strategy increasing from 2.24% in patch releases to 13.58% in major releases.

Figure 1 shows the distribution of strategies adopted by different types of releases of each project domain. RNs of **System Software** projects rarely adopt the *Plain List* structure to organize changes. Compared with the other three domains, System Software projects prefer to organize their changes by *Affected Module* and *Hybrid* strategies. The reason might be that System Software projects usually involve complex development processes, with hundreds or even thousands of commits per release. Therefore, hierarchical lists are more clear than *Plain List* in presenting information.

In the case of **Application Software** projects, RN producers prefer to present the changes as *Plain List* or a hierarchical

[2]We respectively measure the Cohen's Kappa values for the seven categories of RN content expect for the License Changes (as there are only two cases).

TABLE II
STRATEGIES TO ORGANIZE THE CHANGES IN RELEASE NOTES.

Strategy	Description	Major.	Minor.	Patch.	Total.
Plain List	All changes are presented as a plain list.	23.46%	30.84%	48.88%	35.46%
Change Type[a]	Changes are organized based on their types of content, e.g., New Features, Fixed Bugs, and Breaking Changes.	62.96%	58.59%	42.15%	53.76%
Affected Module[b]	Changes are grouped based on the modules they affect.	20.37%	16.30%	11.66%	15.69%
Change Priority[c]	Changes are ranked based on their importance perceived by RN producers.	13.58%	6.16%	2.24%	6.70%

* Note that a RN may adopt multiple strategies, so the sum of each column is greater than 100%.
[a] Example Reference: https://github.com/socketio/socket.io/releases/tag/4.0.0
[b] Example Reference: https://github.com/dbeaver/dbeaver/releases/tag/22.0.0
[c] Example Reference: https://github.com/debug-js/debug/releases/tag/4.0.0

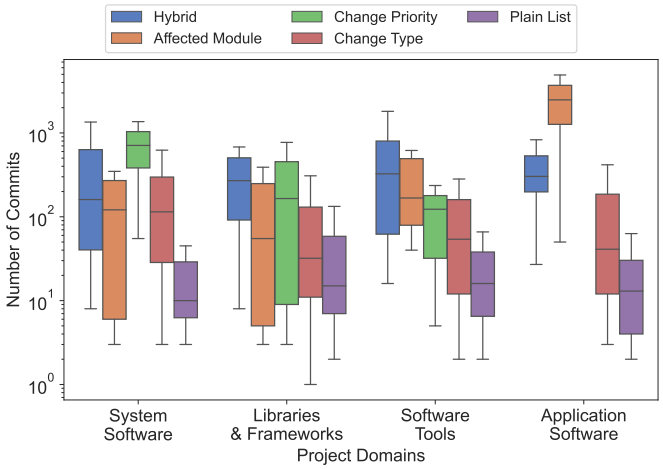


Fig. 2. Distribution of the number of commits (log) between releases under four domains for five Structure strategies.

list based on change types. A possible explanation is that Application Software projects provide specific functionality for users, and such structures are easy for users to comprehend. For example, Aerial, a famous mac screensaver with more than 20k stars, only lists the information about new features to users without providing any technical details. RNs of **Libraries & Frameworks** and **Software Tools** projects follow a similar distribution over the five strategies in their patch, minor, and major releases. Particularly, they prefer to present changes as *Plain List* in patch releases and as hierarchical lists in minor and major releases. For example, Slim, a PHP framework that helps developers quickly write simple web applications and APIs, highlights the most valuable changes in the "Major Changes" section, while other changes are listed in the "Changelog" section in the RN of version 4.0.0 [44].

Figure 2 presents the distribution of the number of commits between two successive releases over these strategies for each project domain. Obviously, *Plain List* strategy is popular for releases containing fewer commits. Perhaps not surprisingly, fewer commits suggest less information, and *Plain List* is sufficient and efficient to present changes clearly. With more commits between successive releases, RN producers prefer to use the hierarchical structure to present the changes. Despite the fact that hierarchical structures require more effort, they make it easier for users to locate information quickly, especially when there are numerous changes. We can also observe different shift patterns of strategy preference for different domains of projects with the increase in the number of commits between two successive releases. For **Software Tools** projects, as the number of commits between releases increases, the preferred structure strategy gradually shifts from *Plain List*, to *Change Type*, to *Affected Module*, and to *Hybrid*.

Summary for Structure:

We find about 1/3 of RNs are organized as plain lists and 2/3 RNs are organized as hierarchical lists. We identify three strategies that developers adopt to develop hierarchies including by *Change Type*, *Affected Module*, and *Change Priority*. Changes are primarily organized by *Change Type* across all domains. RNs of major releases and RNs of System Software projects are more likely to organize changes by *Affected Module*. 11.60% of RNs adopt multiple strategies to present the changes.

B. Writing Style

In this section, we report what styles producers employ to describe changes in RNs. We derive three types of writing styles for RNs based on classical writing theory [45], as shown in Table III, i.e., *Expository*, *Descriptive*[3], and *Persuasive* with increasing manual involvement.

Table III also provides the frequency of each writing style for major, minor, and patch releases. 41.70% of RNs for patch releases are written in the *Expository* style, i.e., RN producers do not include any additional information in RNs other than the content of the change-related commits/PRs/issues. 59.26% of RNs for major releases contain additional information to assist developers in understanding the changes.

[3]We refer to the web writing style convention [46] and also combine the "Descriptive" and "Narrative" writing styles into "Descriptive" in RNs.

TABLE III
CATEGORIES OF WRITING STYLES IN RELEASE NOTES.

Level	Description	Major.	Minor.	Patch.	Total.
Expository[a]	RN producers directly list the content (usually title) of change-related commits/PRs/issues	14.20%	29.96%	41.70%	30.07%
Descriptive[b]	RN producers re-phrase the content of change-related commits/PRs/issues to increase the readability. Sometimes RN producers summarize the content of similar commits/PRs/issues.	26.54%	33.48%	42.15%	34.80%
Persuasive[c]	In addition to presenting the content of related commits/PRs/issues, RN producers provide additional information to help developers understand the changes, such as the rationale behind the changes, the impact of the changes, and guides for the upgrades.	59.26%	36.56%	16.14%	35.13%

[a] Example Reference: https://github.com/Homebrew/brew/releases/tag/3.5.5
[b] Example Reference: https://github.com/iina/iina/releases/tag/v1.3.0
[c] Example Reference: https://github.com/GoogleContainerTools/kaniko/releases/tag/v1.0.0

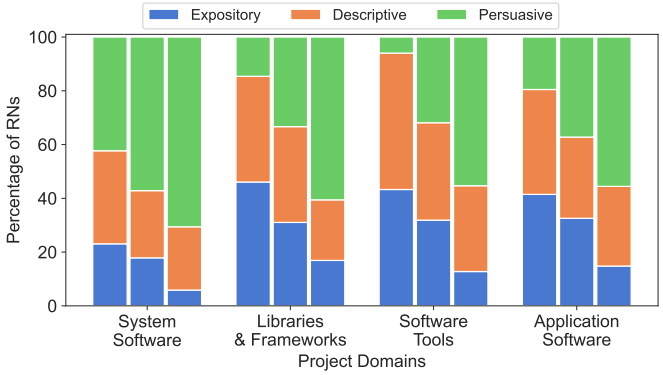


Fig. 3. Distribution of Writing Style under four domains for release types. The bars from left to right represent Patch, Minor, and Major releases, respectively.

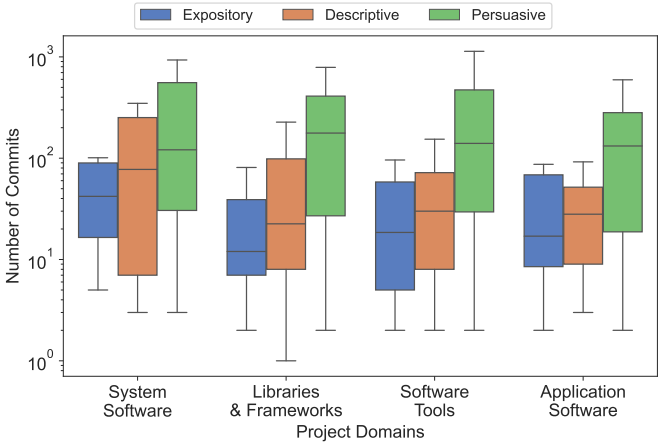


Fig. 4. Distribution of the number of commits (log) between releases under four domains for three Writing Style categories.

For example, as a well-known JavaScript compiler, Babel offers well-crafted RNs for the major release v7.0.0, including a summary of the whole changes, a highlight of their significant impact on users, usage case, migration guidelines for major breaking changes, and detailed background of each other change accompanied with links [47]. However, they only classify the PRs into different sections without modifying the description in the patch release 7.18.8 [48]. This is probably because, on the one hand, patch releases usually contain a few bug fixes, and the content of change-related commits/PRs/issues (i.e., *Expository* style) are sufficiently self-explanatory. Major releases often introduce massive and complex changes, which if not explained in a clear manner, may confuse users. Therefore, producers devote greater efforts (e.g., taking *Persuasive* style) to refining RNs of major releases. On the other hand, as revealed in prior work [4], users state RNs of major releases should be improved regarding the description. Indeed, a well-compiled RN of major releases indicates that the software is under active development and maintenance.

Also, we notice that the writing style of RNs is influenced by automated tools used to generate RNs, which projects adopt for simplicity. For example, Saleor [49] uses the popular release-it [23] tool and Rasa employs a script [50] to generate RNs for patch releases. However, popular RN generation tools on GitHub can only produce RNs with *Expository* style, because they can only present the related commits/PRs/issues description and contributors without any additional informative information, which however requires manual efforts and can not be automated at present.

In Figure 3, we can observe that the three writing styles follow similar distribution for **Application Software** and **Libraries & Frameworks** projects. For the patch release, the most frequently adopted writing style is *Expository*, taking 41.46% of RNs in Application Software and 46.07% of RNs in Libraries & Frameworks projects. For RN of the patch and minor releases for **Software Tools** projects, the most common writing style is *Descriptive*. It reflects that RN producers of Software Tools projects usually re-phrase and combine the content of related commits/PRs/issues. RN producers of **System Software** projects make the most efforts to compile

their RNs for better clarity and readability, with a higher percentage of *Persuasive* writing style than the other three domains, taking 42.31%, 57.14% and 70.59% for patch, minor and major releases respectively. In the case of etcd, which is a distributed consistent key-value database, it describes notable enhancements for solving the scalability issues after upgrading the v3 API in detail and the impact of the upgrade on users, e.g., improved latency and throughput.

From Figure 4, we can observe more clearly that the writing styles for RNs across four domains are all related to the number of commits between versions. More commits between releases indicate that RN producers have to devote more time and efforts to improving the quality of the RNs' writing style.

Summary for Writing Style:

We find three types of writing styles: *Expository*, *Descriptive*, and *Persuasive* with increasing manual involvement, taking 30.07%, 34.80%, and 35.13% of RNs respectively. 83.10% of RNs of System projects adopt *Descriptive* and *Persuasive* style. 59.26% of RNs of major releases provide additional information for changes, e.g., rationales and impacts of changes to improve clarity and readability.

C. Content

Bi et al. [4] reported the top three documented categories. For different domains, the three categories are often the same, i.e., new features, fixed bugs, and system internal changes, but have different proportions, which advanced our knowledge of how different domains of projects document the content of changes. In this paper, we further reveal that different domains of projects show unique preferences for the other types of content, which are equally important.

In Figure 5, we present the percentage of RNs containing each content type across different project domains and release types. As there are only two RNs containing Licence Changes information (transferring to MIT license), we omit it in Figure 5. Consistent with previous work [4], [7], we find *Fixed Bugs* and *New Features* are the most frequently documented information in all four project domains. It suggests that RN producers commonly recognize two types of information essential in RNs. Besides, we also find *Dependency/Environment Changes* are also frequently documented, appearing in approximately half of RNs in the four domains. Intuitively, the first step for users to use these software projects is to successfully install and configure them. So RN producers usually record *Dependency/Environment Changes* in RNs to inform users to avoid installation failures.

Libraries and Frameworks projects have the highest percentage of RNs containing Documentation Changes (26.32%) among the four domains. Additionally, over 1/3 of their RNs contain *Breaking Changes*. Libraries & Frameworks projects provide APIs that allow users to reuse their functionalities.

[5]We use the abbreviations from left to right to represent Breaking Changes, New Features, Fixed Bugs, Enhancements, Document Changes, Dependency/Environment Changes and Security Changes respectively.

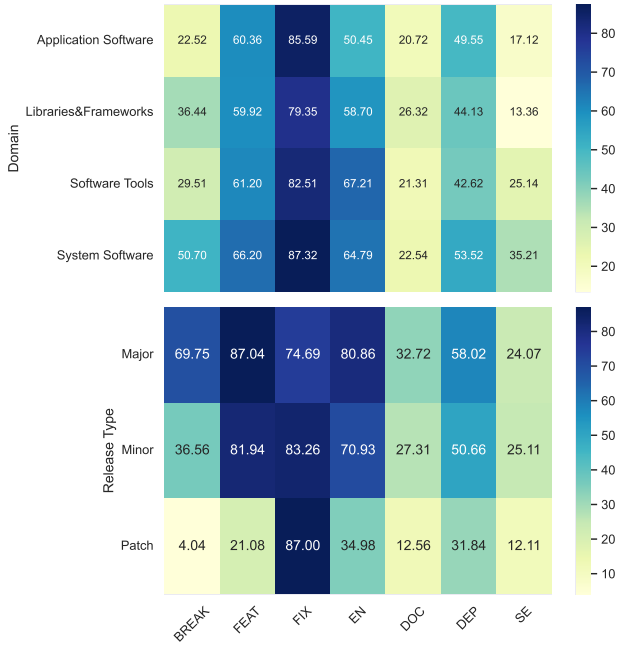


Fig. 5. Percentage distribution of Content categories across the project domains (upper) and release types (lower).[5]

Documentation Changes are necessary for users to be able to utilize these APIs, and *Breaking Changes* may cause downstream software to crash. Both pieces of information are critical to inform users whether to upgrade or not. **System Software** projects are often complex providing basic services and infrastructure to other projects. Their audiences are broad including both end-users and developers. To serve different users, their RNs contain the most comprehensive information. Specifically, *Breaking Changes* and *Security Changes* account for the highest percentage of RNs for System Software projects. There are two possible reasons. On the one hand, due to the complexity of System Software projects, they are more likely to introduce *Breaking Changes* and suffer from security vulnerabilities. On the other hand, System Software projects usually act as the basis for other software, which means that most of their users are concerned about their breaking changes and security vulnerabilities. Both **Software Tools** and **Application Software** projects usually provide many user-oriented functionalities and the frequency of *New Feature* and *Fixed Bugs* for RNs of both domains is almost identical. RNs of Software Tools have a higher percentage of *Breaking Changes*, *Enhancements*, and *Security Changes*, while RNs of Application Software projects contain more Document and Dependency/Environment information for ease of installation and upgrade for end-users. Only 50.45% of the RNs for Application Software projects introduce Enhancement, while 67.21% of RNs in Software Tools projects include *Enhancements*. Such differences may be attributed to the audiences of the project domains, i.e., the major users of the **Software Tools** and **Application Software** are developers and end-users, respectively. Users of some Application Software projects may

not be developers, thus technical details are not necessary and even confuse users. While most users of Software Tools projects are developers, who benefit from technical details.

As shown in Figure 5, we can observe that for a patch release, 21.08% of RNs introduce some new features in the patch release, such as the version 4.3.4 of Redis-py [51]; even for minor releases, 36.56% of RNs contain breaking changes, for example, the version 1.8.0 of Packer [52], which violates the SemVer rule [32] We observe that the frequency of most content categories, e.g., *New Features* and *Breaking Changes*, increases significantly with a shift from minor to major releases), which indicates that major/minor releases tend to contain more updates and prefer to introduce more content. It should be noted, however, that the percentage of *Fixed Bugs* and *Security Changes* in major releases is even lower than in minor releases, which illustrates the producer's strategy: only introduce information that tantalizes users into upgrading. For example, spaCy, an NLP library, only introduces *New Features*, *Enhancements*, *Breaking Changes* in the major version 3.0.0 [53], while recording the fixed bugs and security vulnerabilities in its patch, and minor releases.

Summary for **Content**:

System Software and Libraries & Frameworks projects record *Breaking Changes* more frequently, while Software Tools projects emphasize *Enhancements*. RNs of System Software contain the most comprehensive information to serve different users. *Fixed Bugs* and *Security Changes* are less common in RNs of major releases.

V. Implication

Our results provide rich implications for RN production from the dimensions of Structure, Writing Style, and Content.

A. Release Note Structure

As an essential reference source for software upgrades, RNs should be well-structured in a way that readers can extract necessary information easily. We refer to Information Architecture (IA) models that focus on effectively structuring, organizing, labeling the content [54] to further understand our structure strategies in RNs. There are four IA models: tunnel, flat (matrix), deep hierarchy (tree), and hybrids to convey information more effectively and efficiently. Our results align with IA models. Specifically, *Plain List* corresponds to tunnel, the three hierarchical strategies correspond to deep hierarchy (tree), and multiple strategies correspond to hybrids. Tunnel architecture in IA is a simple straightforward structure for seeking information and we observe that RNs for patch releases generally adopt a *Plain List* structure, while RNs of minor and major releases often choose hierarchy list structures to avoid visual chaos. Therefore, we suggest RN producers use a *Plain List* structure when the release only contains a few changes and a hierarchy list for releases with many changes.

Hierarchy architecture organizes information in a top-down manner in order to allow users to review increasingly detailed content [55]. We observe that different hierarchical strategies are preferred by projects from different domains since each of these hierarchical structures has its own unique characteristics. According to IA, the structure should take user and content into account. Therefore, we presume that RNs organized by *Change Type* and *Change Priority* are more friendly to end users, while those by *Affected Module* are more developer-friendly. For the software projects, such as Application Software, mainly targeted at end-users, we recommend that they organize RNs by *Change Type* or *Change Priority*, which will facilitate locating relevant contents, such as new features and highlighted changes. For Libraries & Frameworks and Software Tools projects, their users are primarily developers who expect to know more low-level technical information [4]. Except for providing change type information, RN producers can also provide information on affected modules to assist users in understanding whether and to what extent their software might be affected by this new release. For example, System Software projects may consist of multiple components, provide basic services and infrastructure to both developers and end-users, and have a more complex development process, e.g., each component of Linux is maintained by specific maintainers. [56]. *Affected Module* or *Hybrid* strategies can be used to facilitate the localization of the information. However, *Hybrid* strategy is not a silver bullet. On the one hand, it requires additional information about changes, which will impose extra labour on contributors or RN producers. On the other hand, multiple strategies may scatter information throughout the RN and confuse users who are interested in certain types of information, such as changes affecting a specific module scattered throughout several sections.

B. Release Note Writing Style

Another purpose that RNs serve is to help users understand the changes. Therefore, writing styles also play a vital role. We borrow classical writing theory [45] and apply it to gain a deep understanding of the writing style characteristics of RNs. According to the survey conducted by [2], RN producers are usually core members of the team (e.g., architects, project managers), who are also responsible for other routine tasks, such as scheduling the development process. Thus, choosing a proper writing style for RNs is also a trade-off between development responsibility and producing RNs with well-compiled descriptions to meet users' needs. Our results can help producers strike a balance. To shorten the time for producing RNs, we recommend that RN producers adopt the *Expository* style when the changes are limited and RN producers can also resort to existing RN generation tools mentioned in Section II-B. However, if the number of changes increases, users may become lost in the raw content of change-related commits/PRs/issues. They have to spend plenty of time understanding the effect, rationale, and goals of the changes. RN producers should pay greater attention to the *Descriptive* and *Persuasive* styles in this case to make RNs clear and compelling. As far as we know, there are no tools that automatically generate RNs in *Persuasive* style.

Additionally, we provide a reference on how to adopt a proper writing style by the number of commits contained in releases for the four domains in Figure 4. We also recommend that RNs of System Software projects provide more informative explanations of the changes to serve different users, thereby reducing the cost of upgrading. Note that when reading carefully compiled RNs, users can sense the sincerity of RN producers [57], which motivates them to become more loyal and more willing to upgrade [58].

C. Release Note Content

In Section IV-C, we find the distribution of content types varies with project domains and release types. Specifically, RNs of Libraries & Frameworks projects need to pay more attention to *Document Changes* to facilitate the usage of downstream projects, while RNs of Software Tools projects may focus more on performance and security improvements to strengthen developers' confidence when they develop software with it. We recommend that RN producers of System Software projects provide a more comprehensive introduction of various categories to serve a variety of audiences. When developers have to balance the information abundance and RN length for major or minor releases, *New Features*, *Enhancements*, and *Breaking Changes* can be prioritized while the number of occurrences for *Fixed Bugs* and *Security Changes* can be minimized. A developer states that *"it is a best practice to separate bug fixes (patches) from new features (minor) and breaking changes (major), into separate releases"* [59].

In particular, our results indicate that some software projects violate SemVer in practice, which could lead to inaccurate assessments of the upgrade risk by users [2]. To minimize the impact of introducing incompatible changes, RN producers should further evaluate the software update/upgrade and adhere strictly to the SemVer convention.

D. How far We Are to Generate Release Notes Automatically

Extensive works in Section II-B have explored automated generation from the perspective of contained information. For structure, however, none of the popular RN generation tools and studies mentioned in Section II-B can fully automate the customization process. These tools require strict commit rules to categorize the changes such as Angular Commit Message Conventions [42], or a PR/issue labeling system during the review process. Thus, we recommend that a dedicated classifier be designed to automatically categorize the changes into the various types of content and link the changes with affected modules. Besides, the process of automatically organizing changes by their priority is challenging though crucial. RN producers can further filter out trivial changes by combining the location and type of change.

For Writing Style, researchers have attempted to semi-automatically generate RNs, e.g., using pre-defined templates [1]. However, these methods have never been applied to actual production environments [4]. To generate rationales and goals for the changes, as well as the effect of the changes, we suggest: (1) exploring and mining diverse types of relevant information, e.g., commits, PRs, issues, and CVEs between releases, milestones and wiki for the project [14]; (2) resorting to state-of-the-art NLP techniques, such as key information extraction [60], text summarization [61] and style transfer [62].

VI. Threats to Validity

Internal Validity concerns the threats to how we perform our study. The subjectivity of inspection is a crucial threat to our work. Our categories construction and labeling process for Content, Structure, and Writing Style is based entirely on manual analysis. Due to the inspector's experience, the classification of the projects' domain can introduce errors. To minimize these threats, two authors are involved in inspecting RNs and reaching an agreement with the help of a third author through discussions. Finally, we respectively measure the kappa values for content ($k \geq 0.76$), structure ($k \geq 0.87$), and writing style ($k \geq 0.74$), demonstrating the reliability of the coding schema and procedure.

External Validity refers to the threats to generalizing our findings. The first threat relates to the selection of data sources. Our works use GitHub projects as the only data source to characterize the software RNs. There may be valuable insights that are overlooked from other sources that have been used in previous studies [7]. To mitigate it, we invite industry developers to validate whether our categories can cover the RN structures and writing styles they have encountered. We believe our results reveal valuable insights into RN production, as well as practical characteristics. The selection of criteria is a second external validity. We design several criteria to collect high popularity and representative projects in Section III-A1, because these projects are more likely to develop good and mature practices of producing RNs. However, only 955 projects on GitHub meet our strict criteria for filtering projects. After sampling and classifying the domains, the number of their RNs across the various domains is not evenly distributed, which may introduce bias into our analysis. Besides, since the size of our dataset is comparable with previous studies [63], [64], [65], [66], [67], we consider this threat is reasonably reduced.

VII. Conclusion

In this paper, we manually analyze 612 latest RNs from 233 popular GitHub Software Projects and characterize software RNs from the dimensions of Structure, Writing Style, and Content. We identify three strategies to organize changes into hierarchical lists: by *Change Type*, *Affected Module*, and *Change Priority*. We identify three levels of Writing Style: *Expository*, *Descriptive*, and *Persuasive*. We investigate how the distributions of different structure strategies, writing styles, and content types vary with project domains and release types. Our results clarify confusion regarding RN production, e.g., software versioning number specification. We discuss how far we are to generate practical and informative RNs and provide a research roadmap for further improvement that we believe will benefit the community.

Acknowledgments. This work is sponsored by the National Natural Science Foundation of China 61825201 & 62142201.

REFERENCES

[1] L. Moreno, G. Bavota, M. D. Penta, R. Oliveto, A. Marcus, and G. Canfora, "Arena: An approach for the automated generation of release notes," *IEEE Transactions on Software Engineering*, vol. 43, no. 2, pp. 106–127, 2017.

[2] J. Wu, H. He, W. Xiao, K. Gao, and M. Zhou, "Demystifying software release note issues on github," in *2022 IEEE/ACM 30th International Conference on Program Comprehension (ICPC)*, 2022, pp. 602–613.

[3] "Release early, release often from wikipedia," en.wikipedia.org/wiki/Release_early,_release_often.

[4] T. Bi, X. Xia, D. Lo, J. Grundy, and T. Zimmermann, "An empirical study of release note production and usage in practice," *IEEE Transactions on Software Engineering*, 2020.

[5] M. Zhou, X. Hu, and W. Xiong, "openeuler: Advancing a hardware and software application ecosystem," *IEEE Software*, vol. 39, no. 2, pp. 101–105, 2022.

[6] "Good practices of writing release notes in stackexchange," https://softwareengineering.stackexchange.com/questions/167578/good-practices-of-writing-release-notes.

[7] S. L. Abebe, N. Ali, and A. E. Hassan, "An empirical study of software release notes," *Empirical Software Engineering*, vol. 21, no. 3, pp. 1107–1142, 2016.

[8] E. Aghajani, C. Nagy, M. Linares-Vásquez, L. Moreno, G. Bavota, M. Lanza, and D. C. Shepherd, "Software documentation: the practitioners' perspective," in *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*. IEEE, 2020, pp. 590–601.

[9] G. A. A. Prana, C. Treude, F. Thung, T. Atapattu, and D. Lo, "Categorizing the content of github readme files," *Empirical Software Engineering*, vol. 24, no. 3, pp. 1296–1327, 2019.

[10] "User guide - wikipedia," https://en.wikipedia.org/wiki/User_guide.

[11] F. Khomh, T. Dhaliwal, Y. Zou, and B. Adams, "Do faster releases improve software quality? an empirical case study of mozilla firefox," in *2012 9th IEEE working conference on mining software repositories (MSR)*. IEEE, 2012, pp. 179–188.

[12] M. Zhou and A. Mockus, "What make long term contributors: Willingness and opportunity in oss community," in *2012 34th International Conference on Software Engineering (ICSE)*, 2012, pp. 518–528.

[13] "The firefox release notes process from mozillawiki," https://wiki.mozilla.org/Release_Management/Release_Notes, 2021.

[14] S. S. Nath and B. Roy, "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, 2022, pp. 1270–1277. [Online]. Available: https://doi.org/10.1109/SANER53432.2022.00152

[15] A. Z. H. Yang, S. Hassan, Y. Zou, and A. E. Hassan, "An empirical study on release notes patterns of popular apps in the google play store," *Empir. Softw. Eng.*, vol. 27, no. 2, p. 55, 2022. [Online]. Available: https://doi.org/10.1007/s10664-021-10086-2

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

[17] M. Ali, A. Aftab, and W. H. Buttt, "Automatic release notes generation," in *2020 IEEE 11th International Conference on Software Engineering and Service Science (ICSESS)*. IEEE, 2020, pp. 76–81.

[18] S. S. Nath and B. Roy, "Towards automatically generating release notes using extractive summarization technique," in *International Conference on Software Engineering & Knowledge Engineering, SEKE*, 2021, pp. 241–248.

[19] H. Jiang, J. Zhu, L. Yang, G. Liang, and C. Zuo, "Deeprelease: Language-agnostic release notes generation from pull requests of open-source software," in *2021 28th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 2021, pp. 101–110.

[20] H. Kamezawa, N. Nishida, N. Shimizu, T. Miyazaki, and H. Nakayama, "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)*, 2022, pp. 8718–8735.

[21] "Semantic release repository on github," github.com/semantic-release/semantic-release, 2022.

[22] "github-changelog-generator repository on github," github.com/github-changelog-generator/github-changelog-generator, 2022.

[23] "Release it repository on github," github.com/release-it/release-it, 2022.

[24] "Release-drafter repository on github," github.com/release-drafter/release-drafter, 2022.

[25] "About on github," github.com/about, 2022.

[26] Y. Zhang, M. Zhou, A. Mockus, and Z. Jin, "Companies' participation in OSS development-an empirical study of openstack," *IEEE Trans. Software Eng.*, vol. 47, no. 10, pp. 2242–2259, 2021. [Online]. Available: https://doi.org/10.1109/TSE.2019.2946156

[27] X. Tan, K. Gao, M. Zhou, and L. Zhang, "An exploratory study of deep learning supply chain," in *Proceedings of the 44th International Conference on Software Engineering*, ser. ICSE '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 86–98. [Online]. Available: https://doi.org/10.1145/3510003.3510199

[28] K. Gao, Z. Wang, A. Mockus, and M. Zhou, "On the variability of software engineering needs for deep learning: Stages, trends, and application types," *IEEE Transactions on Software Engineering*, 2022.

[29] C. Wang, H. He, U. Pal, D. Marinov, and M. Zhou, "Suboptimal comments in java projects: From independent comment changes to commenting practices," *ACM Transactions on Software Engineering and Methodology*, 2022.

[30] "The openapi specification repository on github," github.com/oai/openapi-specification, 2022.

[31] "Software versioning from wikipedia," https://en.wikipedia.org/wiki/Software_versioning, 2022.

[32] "Semantic versioning from semver.org," https://semver.org/, 07 2022.

[33] L. Ochoa, T. Degueule, J.-R. Falleri, and J. Vinju, "Breaking bad? semantic versioning and impact of breaking changes in maven central," *Empirical Software Engineering*, vol. 27, no. 3, pp. 1–42, 2022.

[34] "maxwellito/vivus," github.com/maxwellito/vivus/tree/v0.4.0.

[35] "Sentry repository on github," github.com/getsentry/sentry/tree/22.7.0.

[36] Y. Zhang, M. Zhou, A. Mockus, and Z. Jin, "Companies' participation in oss development–an empirical study of openstack," *IEEE Transactions on Software Engineering*, vol. 47, no. 10, pp. 2242–2259, 2019.

[37] R. He, H. He, Y. Zhang, and M. Zhou, "Automating dependency updates in practice: An exploratory study on github dependabot," *CoRR*, vol. abs/2206.07230, 2022. [Online]. Available: https://doi.org/10.48550/arXiv.2206.07230

[38] D. Spadini, M. F. Aniche, and A. Bacchelli, "Pydriller: Python framework for mining software repositories," in *Proceedings of the 2018 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2018, Lake Buena Vista, FL, USA, November 04-09, 2018*, G. T. Leavens, A. Garcia, and C. S. Pasareanu, Eds. ACM, 2018, pp. 908–911. [Online]. Available: https://doi.org/10.1145/3236024.3264598

[39] H. Borges, A. Hora, and M. T. Valente, "Understanding the factors that impact the popularity of github repositories," in *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2016, pp. 334–344.

[40] S. E. Hove and B. Anda, "Experiences from conducting semi-structured interviews in empirical software engineering research," in *11th IEEE International Software Metrics Symposium (METRICS'05)*. IEEE, 2005, pp. 10–pp.

[41] N. Humbatova, G. Jahangirova, G. Bavota, V. Riccio, A. Stocco, and P. Tonella, "Taxonomy of real faults in deep learning systems," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, 2020, pp. 1110–1121.

[42] "Description for angular.js convention," github.com/angular/angular.js/blob/master/DEVELOPERS.md\#commits, 2022.

[43] "Release dgraph v21.12.0 from graph-io/dgraph," github.com/dgraph-io/dgraph/releases/tag/v21.12.0.

[44] "Release 4.0.0 from slimphp/slim," github.com/slimphp/Slim/pull/2769.

[45] W. Grabe, "Narrative and expository macro-genres," *Genre in the classroom: Multiple perspectives*, pp. 249–267, 2002.

[46] "Writing styles every writer should know - egetleads," https://egetleads.com/6-web-writing-styles-every-writer-should-know/, (Accessed on 10/24/2022).

[47] "Babel 7," babeljs.io/blog/2018/08/27/7.0.0.

[48] "Release v7.18.8 from babel/babel," github.com/babel/babel/releases/tag/v7.18.8.

[49] "Release-it.json from saleor/saleor," github.com/saleor/saleor/blob/main/.release-it.json.

[50] "Release_notes.py from rasahq/rasa," github.com/RasaHQ/rasa/blob/3.2.2/scripts/publish_gh_release_notes.py.

[51] "Version 4.3.4 from redis/redis-py," github.com/redis/redis-py/releases/tag/v4.3.4.

[52] "Release v1.8.0 from hashicorp/packer," github.com/hashicorp/packer/releases/tag/v1.8.0.
[53] "Release v3.0.0 from explosion/spacy," github.com/explosion/spaCy/releases/tag/v3.0.0.
[54] M. Li, R. Gao, X. Hu, and Y. Chen, "Comparing infovis designs with different information architecture for communicating complex information," *Communication Design Quarterly Review*, vol. 5, no. 1, pp. 43–56, 2017.
[55] B. G. Danaher, H. G. McKay, and J. R. Seeley, "The information architecture of behavior change websites," *Journal of medical Internet research*, vol. 7, no. 2, p. e406, 2005.
[56] "List of maintainers in linux," kernel.org/doc/html/latest/process/maintainers.html.
[57] "Issue #5913 from prisma/prisma," github.com/prisma/prisma/issues/5913/\#issuecomment-788326709.
[58] "Release notes best practices — beamer," www.getbeamer.com/blog/release-notes-best-practices.
[59] "Reason for minor vs patch rules in semver," https://stackoverflow.com/questions/46720398/reason-for-minor-vs-patch-rules-in-semver.
[60] Y. Hou, C. Chen, X. Luo, B. Li, and W. Che, "Inverse is better! fast and accurate prompt for few-shot slot tagging," in *Findings of the Association for Computational Linguistics: ACL 2022, Dublin, Ireland, May 22-27, 2022*, S. Muresan, P. Nakov, and A. Villavicencio, Eds. Association for Computational Linguistics, 2022, pp. 637–647. [Online]. Available: https://doi.org/10.18653/v1/2022.findings-acl.53
[61] S. Tu, J. Yu, F. Zhu, J. Li, L. Hou, and J. Nie, "UPER: boosting multi-document summarization with an unsupervised prompt-based extractor," in *Proceedings of the 29th International Conference on Computational Linguistics, COLING 2022, Gyeongju, Republic of Korea, October 12-17, 2022*, N. Calzolari, C. Huang, H. Kim, J. Pustejovsky, L. Wanner, K. Choi, P. Ryu, H. Chen, L. Donatelli, H. Ji, S. Kurohashi, P. Paggio, N. Xue, S. Kim, Y. Hahm, Z. He, T. K. Lee, E. Santus, F. Bond, and S. Na, Eds. International Committee on Computational Linguistics, 2022, pp. 6315–6326. [Online]. Available: https://aclanthology.org/2022.coling-1.550
[62] E. Reif, D. Ippolito, A. Yuan, A. Coenen, C. Callison-Burch, and J. Wei, "A recipe for arbitrary text style transfer with large language models," in *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, S. Muresan, P. Nakov, and A. Villavicencio, Eds. Association for Computational Linguistics, 2022, pp. 837–848. [Online]. Available: https://doi.org/10.18653/v1/2022.acl-short.94
[63] Z. Chen, Y. Cao, Y. Liu, H. Wang, T. Xie, and X. Liu, "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*, 2020, pp. 750–762.
[64] T. Zhang, C. Gao, L. Ma, M. Lyu, and M. Kim, "An empirical study of common challenges in developing deep learning applications," in *2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2019, pp. 104–115.
[65] E. Aghajani, C. Nagy, O. L. Vega-Márquez, M. Linares-Vásquez, L. Moreno, G. Bavota, and M. Lanza, "Software documentation issues unveiled," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 2019, pp. 1199–1210.
[66] S. Beyer, C. Macho, M. Di Penta, and M. Pinzger, "Automatically classifying posts into question categories on stack overflow," in *2018 IEEE/ACM 26th International Conference on Program Comprehension (ICPC)*. IEEE, 2018, pp. 211–21 110.
[67] Y. Zhang, Y. Chen, S.-C. Cheung, Y. Xiong, and L. Zhang, "An empirical study on tensorflow program bugs," in *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2018, pp. 129–140.