

Exploring Relevant Artifacts of Release Notes: The Practitioners' Perspective

Sristy Sumana Nath

Computer Science, University of Saskatchewan
Saskatoon, Saskatchewan, Canada
sristy.sumana@usask.ca

Banani Roy

Computer Science, University of Saskatchewan
Saskatoon, Saskatchewan, Canada
banani.roy@usask.ca

Abstract—A software release note is one of the essential documents in the software development life cycle. The software release contains a set of information, e.g., bug fixes and security fixes. Release notes are used in different phases, e.g., requirement engineering, software testing and release management. Different types of practitioners (e.g., project managers and clients) get benefited from the release notes to understand the overview of the latest release. As a result, several studies have been done about release notes production and usage in practice. However, two significant problems (e.g., duplication and inconsistency in release notes contents) exist in producing well-written & well-structured release notes and organizing appropriate information regarding different targeted users' needs. For that reason, practitioners face difficulties in writing and reading the release notes using existing tools. To mitigate these problems, we execute two different studies in our paper. First, we execute an exploratory study by analyzing 3,347 release notes of 21 GitHub repositories to understand the documented contents of the release notes. As a result, we find relevant key artifacts, e.g., issues (29%), pull-requests (32%), commits (19%), and common vulnerabilities and exposures (CVE) issues (6%) in the release note contents. Second, we conduct a survey study with 32 professionals to understand the key information that is included in release notes regarding users' roles. For example, project managers are more interested in learning about new features than less critical bug fixes. Our study can guide future research directions to help practitioners produce the release notes with relevant content and improve the documentation quality.

Index Terms—Release notes, Exploratory study, Survey study, GitHub

I. INTRODUCTION

Regular releases of software promise to improve product quality and high customer satisfaction [12]. Release notes inform users about essential project changes, e.g., bug fixes, feature enhancements, source code changes, from the previous version to the latest version [13]. Practitioners (e.g., project managers, developers and clients) use release notes in various software development phases, e.g., requirements engineering, software programming, debugging and testing phase [5], [12]. Several empirical studies have been done to analyze the release note contents [4], [7], [13]. For example, Moreno et al. [13] identify 17 types of changes that can be included in the release notes, and Bi et al. [7] classify those changes into eight categories. Furthermore, Abebe et al. [4] discover additional six different types of information in release notes (described in Section II). Despite the existing studies, in reality practitioners

face difficulties in release note productions and usages because of *unclear contents with poor structural presentations*. The documented information in release notes is currently scattered and poorly organized, and the information is described vaguely [7]. Consequently, release notes help limited release note users in many cases. Therefore, identifying relevant software artifacts and linking them with release notes can help to resolve this issue [7].

The release management process consists of several activities from release initiating to closing [9]. Different target users are involved in the release process, and they are looking for different types of information regarding their project roles [12] in release notes. Bi et al. [7] classify users into two categories: (i) Release Note Producer and (ii) Release Note User. Moreover, the authors cover significant discrepancies between release note producers and users in perceiving release notes. For example, release note producers focus on high-level changes and design decisions of development. In contrast, release note users expect detailed descriptions of new features compared to the previous release. Therefore, *difficulties to use* is another problem of release notes usage. Because it is challenging to know the requirements for the release notes concerning the target users, conducting a survey study with practitioners can help understand the specific audiences' needs.

Our study has two motives: (1) to investigate the relevant software artifacts that can help to classify and structure the information in release notes; and (2) to understand the target users' requirements to tailor the release note contents.

- Exploratory Study: we extract and analyze 3,347 release notes of 21 GitHub projects and then separate release notes' contents (or sentences). We identify different artifacts related to the contents and classify the artifacts into three categories. The study design and result analysis are described in Section III-B and Section IV-A respectively.
- Survey Study: we gather practitioners' opinions on release notes in practice and receive responses from 32 participants. We classify these participants into two categories: internal and external team members. The study design and result analysis are described in Section III-C and Section IV-B respectively.

The key contributions of our study:

- We extract the data from GitHub and develop a dataset

TABLE I: The categories of the documented information in release notes

Study	Contents Category	Percentage
Moreno et al. [13]	Fixed Bugs	90%
	New Features	46%
	New Code Components	43%
	Modified Code Components	40%
	Modified Features	26%
	Refactoring Operations	21%
	Changes to Documentation	20%
	Upgraded Library Dep.	16%
	Deprecated Code Components	10%
	Deleted Code Components	9%
	Changes to Config. Files	8%
	Changes to Code Components Visibility	7%
	Changes to Test Suites	7%
	Known Issues	6%
	Replaced Code Components	5%
	Architectural Changes	3%
	Changes to Licenses	2%
Bi et al. [7]	Issues Fixed	79.3%
	New features	55.1%
	System internal changes	25.1%
	Non-functional requirements	10.3%
	Documentation update	9.5%
	Configuration	2.8%
	Required further actions	2.1%
	Refactoring and reuse	1.9%

for the contents of release notes.

- We identify essential software artifacts from the dataset those help to produce well-structured release notes and classify the contents.
- We analyze the response of the participants and summarize them, which can aid tailoring release notes automatically for different stakeholders.

II. BACKGROUND

A. Release Note Contents

Different communities produce release notes according to their own guidelines, and no common standards exist for writing release notes. In order to understand the release note contents (see Table I), several empirical studies have been done to investigate and categorize the documented information in release notes. For example, Moreno et al. [13] manually inspected 990 release notes to analyze and classify their content into 17 categories by focusing on information at finer level granularity. Similarly, Bi et al. [7] classified the documented information of release notes into eight main categories, comparatively higher-level classification.

On the other hand, Abebe et al. [4] identify six different information, e.g., titles, system overview, resource requirements, installation, addressed issues and caveat, that are included in release notes which contents are relatively high-level. Authors mainly focus on software end-users' perceptions of release notes. Klepper et al. [12] identify some additional information, for example, technical information and testing instructions, in the release notes. Our study explores the different software artifacts relevant to the release note contents.

B. Practitioners of Release Notes

Different role-based practitioners are involved in a software release process [12]. Bi et al. [7] classify the stakeholders

into two groups: release note producers and users. Architects and team managers are mainly responsible for producing release notes. Developers basically write release notes to reflect internal code changes, whereas testers and operators use release notes to perform their tasks. The purpose of release notes usages for different stakeholders (e.g., developers and testers) regarding the phases are described below:

- In the pre-alpha phase, *project managers* and *clients* use the release notes to discuss the project progress with new functionalities and significant issues.
- The release notes of an alpha version are used both by *developers* and *testers*. During this phase, developers first debug all the critical bugs found in a pre-alpha phase. Then, the testers assess the functionality of the application and compare the expected value with the final output value by utilizing the release notes.
- To test the system in the real environment, *clients* and *end-users* use the beta version of the release notes.
- *Team managers* release a stable version of the system and highlight activities in the release notes of the RC version.
- Before deploying the final version of product, the team members write the release notes for the *integrators* (who are using a library in their code) and *end-users*. Therefore, the release notes of the final version need to be concise and properly understandable for the target audience.

However, the documented ineffective and unnecessary information creates problems on the usage of release notes for the targeted audiences [7]. Therefore, our study investigates the valuable information of release notes depending on the target users' perceptions.

III. STUDY DESIGN

This section describes our research questions and study processes. Figure 1 represents the overview of this study.

A. Research Questions

We address the following research questions:

RQ1: *What software artifacts are important for preparing release notes?* To answer the research question, we conduct an exploratory study on 3,347 release notes from 21 GitHub project repositories for understanding the release note contents. Majority of the cases, we identify three crucial software artifacts, e.g., commits, issues and pull-requests, from GitHub and one artifact, e.g., common vulnerabilities and exposures (CVE) issues, is extracted from external sources. These valuable sources can help to prepare quality release notes. Moreover, we find other key artifacts that can assist in maintaining well-structured release notes.

RQ2: *What types of release note contents may vary depending on the software development role of practitioners?* Everyone involved in a software development project can be a practitioner of release notes., and the addressed contents of the release notes can be modified depending on the users' particular information needs [12]. To understand the targeted users' needs, we prepare an online survey study and learn users' opinions about release notes. This survey study may

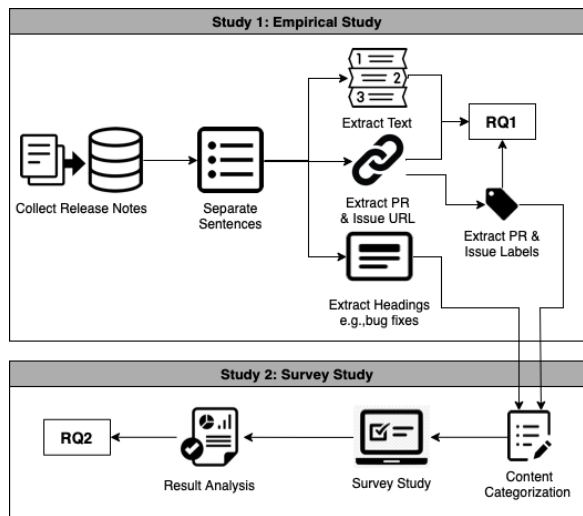


Fig. 1: Overview of our study

help produce high-quality release notes and tailor the documented release notes depending on users' needs.

B. Study 1: Exploratory Study

One key goal of our study is to identify the relevant artifacts of release notes (i.e., RQ1). Therefore, we targeted open-source projects and collected release notes from GitHub-hosted projects.

1) *Data Collection*: First, we collect release notes of open-source projects from GitHub. In this step, we use search option of GitHub by sorting the most number of stars of three-top languages, e.g., JavaScript, Java and Python, based on active repositories [2]. To eliminate the trivial projects, we define following criteria for a project selection from GitHub:

- **created**: minimum of three years ago and is active
- **forks and stars**: $\geq 1,000$ times forks and $\geq 8,000$ stars
- **contributors**: ≥ 30 committers
- **total commit**: $\geq 2,000$
- **release notes**: ≥ 30
- **total resolved issues**: ≥ 500 issues
- **total pull-requests**: ≥ 500 pull-requests

Then, we select total 21 projects, i.e., 8 JavaScript, 7 Java and 6 Python projects, based on the number of stars. Among the projects, we did not consider the non-engineering ones, e.g., `iluwatar/java-design-patterns` and `kdn251/interviews`. Table II represents the detail information about the selected repositories. After project selection, we extract the release notes using the data extraction tool. Data is available on GitHub [1].

2) *Data Analysis*: First, we filter out the empty release notes from the extracted data. Second, we eliminate some information, e.g., contributors' name, to analyze the release note contents. Third, we split the sentences, i.e., contents, and headings from release notes. For example, Fig. 2 represents a release note. Here, *Bug Fixes*, *Features* are heading and the bullet listed information are contents. Then, we extract the

TABLE II: Dataset Overview

SL.	Repository	Domain	# Releases
1	vuejs/vue	Web libraries and frameworks	210
2	facebook/react	Web libraries and frameworks	96
3	twbs/bootstrap	Web libraries and frameworks	73
4	axios/axios	Web libraries and frameworks	32
5	nodejs/node	System software	252
6	mrdoob/three.js	Web libraries and frameworks	125
7	mui-org/material-ui	Web libraries and frameworks	322
8	chartjs/Chart.js	Web libraries and frameworks	81
9	spring-projects/spring-boot	Software tools	112
10	elastic/elasticsearch	Web libraries and frameworks	60
11	ReactiveX/RxJava	System software	225
12	google/guava	Non-web libraries and frameworks	34
13	PhilJay/MPAndroidChart	Non-web libraries and frameworks	44
14	redisson/redisson	Non-web libraries and frameworks	103
15	jenkinsci/jenkins	System software	128
16	tensorflow/tensorflow	Software tools	152
17	tiangolo/fastapi	Non-web libraries and frameworks	113
18	getsentry/sentry	Non-web libraries and frameworks	38
19	pandas-dev/pandas	Non-web libraries and frameworks	81
20	apache/airflow	Software tools	41
21	home-assistant/core	Application software	847



Fig. 2: Artifacts of Release Notes

URLs from each sentences and maintain separate column in the dataset for further analysis.

TABLE III: Participants' Data

Category	Sub-Category	Participants	Total
Type	Internal User	19	32
	External User	13	
Experience	1-3 years	9	32
	3-6 years	16	
	More than 6 years	7	

C. Study 2: Survey Study

The purpose of this survey study is to learn about effective information of the release notes based on users' opinion. In this survey, we recruit participants who have minimum one year software development experience. In particular, we choose the users who have knowledge in software release and release notes. We adopt Non-Probabilistic Sampling methods, i.e., convenience sampling and snowball sampling, to invite participants of our study. We followed two steps to invite participants: (1) We sent email invitations to 200 contributors (only release note producers) of the selected GitHub projects in our study and received 11 (5.5%) responses. In GitHub, one contributor writes release notes in the specific release (marked as green color in Fig. 2). (2) We invited professionals, e.g., project managers, testers, integrators, clients, within our social network and asked them to circulate this invitation to other professionals using sampling methods. We get 21 responses by following this strategy.

We prepare set of questions to professionals about their role and experience and their opinions about release notes. We design a survey study using Google Form and asked

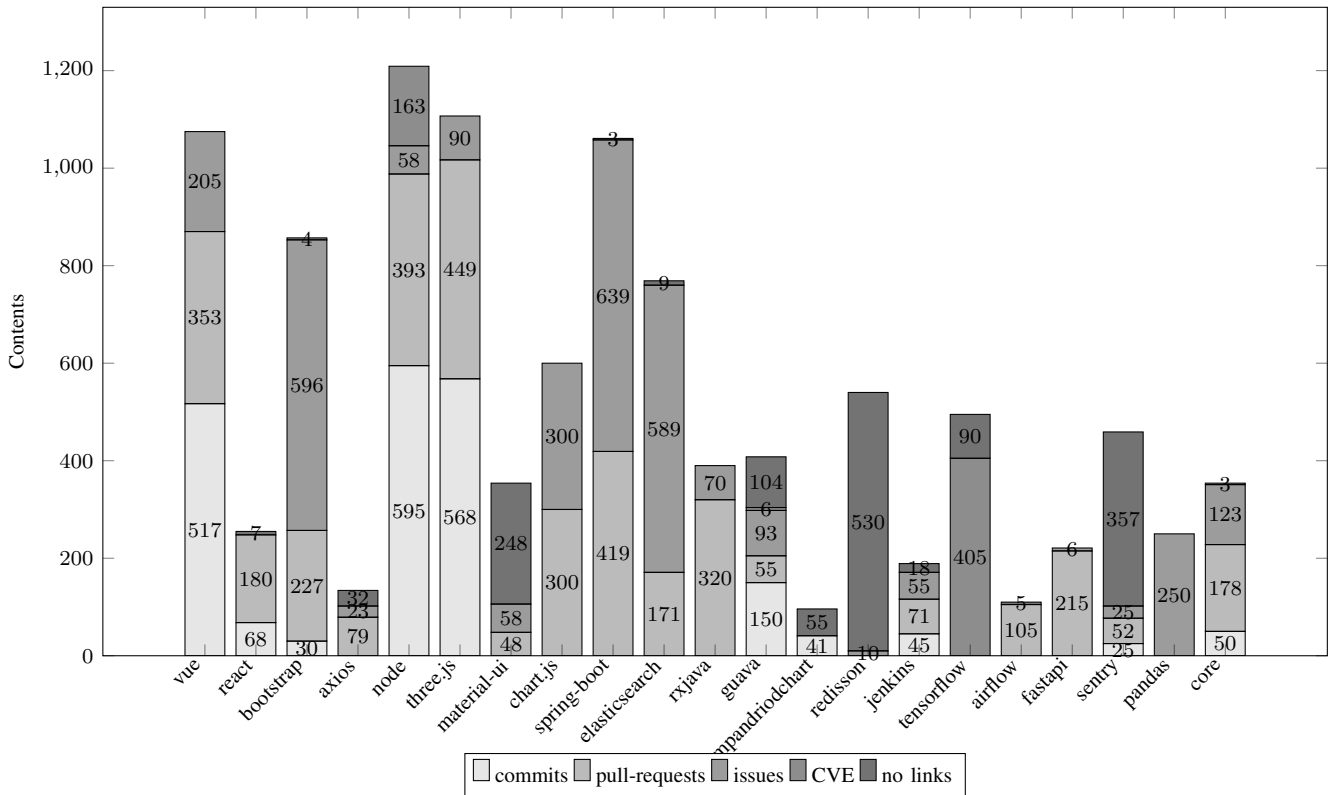


Fig. 3: Release Note Contents Analysis

participants about their roles and opinions in software development. Firstly, participants select their specific role (e.g., project manager, client) and write the number of years of experience in software development process. Finally, they provide their responses of the questions related to release notes, e.g., the content specification and structure of release note and comment. We use the 4-point Likert Scale (based on *Importance*) to understand the priority of the content of release notes based on a user role. Moreover, Abebe et al. [4] identify three different structures which are followed to prepare release notes, and we asked the participants to provide their feedback on the suitable structure of release notes. To response this question, participants select another 5-point Likert Scale (based on *Quality*). We have received responses from total 32 participants. Our questionnaires can be found on GitHub [1].

IV. RESULT ANALYSIS

We describe the results of the empirical study (to answer to RQ1) in Section IV-A. The results of professionals' opinions on release notes are shown in Section IV-B (to answer RQ2).

A. *RQ1: What software artifacts are important for preparing release notes?*

Several studies [4], [6], [7], [14], [15] analyze the release note production and usage; however, the exploration study about release note contents and identifying related software artifacts have not yet been investigated. To investigate the

relevant artifacts, we analyze the contents of our developed dataset. We identify two types of information are important in the release notes. One is *heading* of these contents, e.g., bug fixes and features. Second is *contents*, i.e., list of resolved issues, e.g., *v-slot: fix scoped slot normalization combined with v-if* (shown in Fig. 2). After pre-processing the text, our dataset contains 37,152 release note contents. We mainly focus on this study to find the relevant software artifacts with contents; therefore, we preserve the heading for future work to classify the contents that can help maintain the well-structured information in the release notes.

In the contents analysis step, we find some reference numbers along with URLs that are incorporated with contents (see in Fig. 2). We identify four different URLs pattern from the contents by extracting the URLs. Depending on these patterns, we detect four different software artifacts that can help to produce release notes. For example, issues, pull-requests, commits and CVE (Common Vulnerabilities and Exposures) are linked with software release in Fig. 2. For example,

- Issues: `/issues/issue_id`
- Pull-requests: `/pull/pull_id`
- Commits: `/commit/commit_id`
- CVE: `/cvename.cgi?name=cve_id`

Fig. 3 shows the comparison of these four type artifacts, and among them, the bar chart represents pull-requests (32%) are the most important artifacts to produce release notes.



Fig. 4: Labels

Interestingly, in 41% cases, the release note contents are linked with multiple artifacts, e.g., issues, pull-requests and commits. For example, the documented information of release notes is linked with multiple artifacts (see in Fig. 3). To improve the usability of release notes for practitioners, we need to consider the content selection approach from these artifacts in future.

Additionally, we explore the linked commits, pull-requests and issues and identify the labels of pull-requests and issues. For example, `docs` and `v5` are two labels of issues (see in Fig. 4). Cabot et al. [8] analyze the issue labels and categorize them into four categories based on the usages, e.g., priority, versioning, workflow and architecture. Our analysis finds the text similarity between content headings and issues/pull-requests label. The contents come from the issues, pull-requests and commits (developers cannot add labels in commit); therefore, we can use the labels to classify the information in release notes. Table V represents the example.

Finally, based on the analysis of result of RQ1, we classify these artifacts into three categories:

- **internal artifacts:** pull-requests, issue, commit
- **external artifacts:** CVE issues
- **other key artifacts:** tags of issues/pull-request, milestones, change-log.md, wiki

Moreover, around 14% cases, we do not find any related artifacts in the release note contents. In future, we need to analysis manually to get some more findings about release note contents.

B. RQ2: What types of release note contents may vary depending on the software development role of practitioners?

We conduct a survey study with professionals and define a set of questions about release notes contents and structure. In this study, we need expert opinions, and for that reason, we encourage those participants who have at least one year experience in software development. From the previous studies [7], [13] (details in Table I), we can see that bug fixes and new features are documented more in the release notes than the other types of contents. Our survey study includes these content types and asks participants what contents categories they are looking for along with the justifications. Table IV presents the content categories and summary of survey results.

According to their opinion, software release notes can be customized that will help for documenting for target users.

Participants choose option of 4-Likert Scale (*not important, slightly important, moderately important, important*). We received the responses of 32 participants of five different role-based users. The results of the online survey are summarised in Table IV. We have found the following observations:

- Bug fixes are essential content for *developers* and *testers* based on the avg. score (i.e., 3.4). Besides, *Project managers* and *clients* mention they are interested to know learn major or critical issues from the release notes rather than the all bugs. However, list of all bugs is not helpful.
- New features and feature enhancements are important content of release notes for both internal users and external users (Avg. score of likert distributions of Table IV).
- *Integrators* and *clients* concern about the documentation changes, e.g., installation instructions and UI changes, in the software development.
- The internal code changes include added/modified /deleted classes/methods in the source code. Developers must document these changes in release notes because it will help track changes, and new developers can benefit. Moreover, integrators want to know the source code changes that can help them integrate into their system.
- Architecture provides fundamental structure of software development and development teams follow the architectural process, e.g., model-view-controller model, to build the software. Refactoring is one kind of architectural changes; this process is restructuring the code without changing the external behavior. *Project managers* and *developers* give priority to know about architectural changes through the release notes.
- *Project managers* and *clients* want to know the non-functional changes, e.g., security issues or performance improvements, about the software. Because these changes help to improve the quality of software by solving some vulnerabilities and optimization issues.
- Dependency changes mean upgrade the APIs or libraries. This content essential for *developers* and *integrators* because without updating the dependency changes they cannot implement or use the updated version of the software.
- *Integrators* and *clients* want to learn about the required configuration changes (e.g., installation or hardware requirements) because without updating these requirements they cannot use the latest release.
- Testing information, e.g., test cases or suites, is useful content of release notes for *testers*. Because the internal testers receive unstable versions and can know which tasks to be tested or which to be ignored.

Additionally, we ask the participants about the suitable structure of release notes, which is easy to read. The following three styles are observed in [4]:

- Style 1: summarize or re-phrase the addressed issues in the current release
- Style 2: list of selected issues that were addressed in the

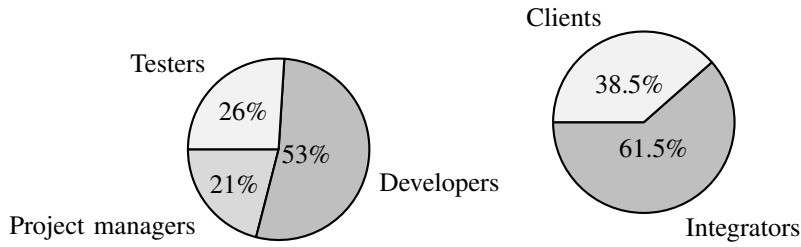


Fig. 5: Ratio of User’s Participation

TABLE IV: Survey Result

Types of Content	Practitioners’ Response (Likert Distributions & Avg. Score)									
	Project Managers		Developers		Testers		Integrators		Clients	
Bug Fixes		2.3		3.4		3.4		1.8		1.7
New Features		3.4		3.4		3.1		3.2		3.4
Feature Enhancements		3.3		3.3		3.1		3.0		3.4
Documentation Changes		2.8		2.4		2.1		3.0		3.2
Internal Code Changes		2.4		3.1		1.6		2.6		1.8
Architectural Changes		3.0		2.9		1.7		1.3		1.9
Security Issues		3.1		2.3		2.8		2.5		3.0
Performance Issues		3.2		2.8		2.8		2.4		3.1
Dependency Changes		2.7		3.2		1.7		3.2		2.5
Configuration Changes		2.4		2.4		1.8		3.1		3.3
Tests		3.2		2.4		3.3		2.4		3.1

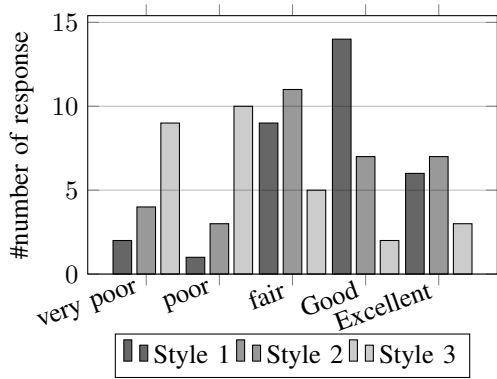


Fig. 6: Response about the Structure of Release Notes

current release (do not summarize issues)

- Style 3: list of all addressed issues in the current release

Participants provide score according to the quality structure (*very poor*, *poor*, *fair*, *good*, *excellent*) of release notes. We identified style 1, e.g., summary of related issues is more preferable (i.e., 14 and 6 users choose *Good* and *Excellent* option, respectively) than the other two structures (see Fig. 6). Example of statements:

- “Need to use few sentences with bullet points to let users

know what’s new and how to navigate these.”

- “Major issues of the release are required to include in release notes; however, sometimes technical contexts are made hard to read the release notes.”

V. DISCUSSIONS

We discuss our results analysis findings and threat to validity in this section.

A. Release Notes Generation

We discuss few concerns about release notes generation.

Content Selection. The results of RQ1 show that release note contents are linked with issues, pull requests, and commits. Generally, issues are created first, and developers solve those issues through commits and pull requests. Release notes of the latest version contain the resolved issues list from the immediate previous release. However, we identify that the documented contents can be different among issue titles, pull-request titles, and commit messages. We represent one content from the release note of version 2.6 [3] of Vue GitHub project.

- **Content:** *attrs: do not consider translate attribute as boolean*
- **Commit message:** *attrs: do not consider translate attribute as boolean* [10]

TABLE V: Examples of Common Content Types

Examples of headings	Examples of labels
Fixed, Bug fixes	bug, regression, type:bug, bug on dependency library
Breaking changes, Features	feat:ssr, feature request, type:feat
Development, Added, Modified	core, type:dev, type:chore
Docs, Documentation changes	docs, type:documentation
Performance Improvements	performance, type:performance
Dependency Changes, API Changes, Upgrades, Dependencies	dependencies, type:dependency-upgrades

TABLE VI: Problems of Existing Release Notes

Language	Num. of Projects	Duplication (%)	Inconsistency (%)
JavaScript	8	19	10
Java	7	22	7
Python	6	0	0

- **Issue title:** *isBooleanAttr for the HTML attribute 'translate' [11]*
- **Pull-request title:** *fix #11391: translate attribute had incorrectly the key as it's value [16]*

The above example shows that the documented content comes from the commit messages, and the titles of issue and pull-request are different. Content selection is a crucial part of producing good-quality release notes. Different automated release notes generation tools [6], [14] consider only issues and generate text from source code changes. We encourage researchers could consider the content selection from the relevant artifacts before developing new tools.

Content Classification & Structure. Bi et al. [7] encourage researchers to identify other software artifacts which help to classify and structure the information in release notes. We identify that the issues and pull-requests labels can help to classify the release note contents. Table V represents the similarity between contents' headings and tags. Labels or tags can help to classify the release note contents. Moreover, we identify another two artifacts, e.g., milestones and wiki, from *Three.js*, *redisson* and *jenkins* projects. Similarly, setting a milestone of upcoming releases can organize the structure of documenting information of release notes.

Content Tailoring. There are no worldwide standards or guidelines for documenting release notes [7], [13]. Furthermore, documenting all issues or changes in release notes makes it too lengthy. Consequently, readers lose interest in knowing the critical addressed issues in the current release [4]. Our second study finds that different practitioners look for different information in the release notes. Therefore, automated techniques need to integrate tailoring tools to customize the release note contents regarding target users' requirements.

B. Issues of Existing Release Notes

We identify two problems in the existing release notes in GitHub and that are discussed as follows.

Content Duplication. The first issue is same contents (or sentences) appear double or triple times in a release note. Fig. 7 represents an example of duplication problem and Table VI shows the duplication percentage in different GitHub projects. We investigate the reason behind this issue, and identify that

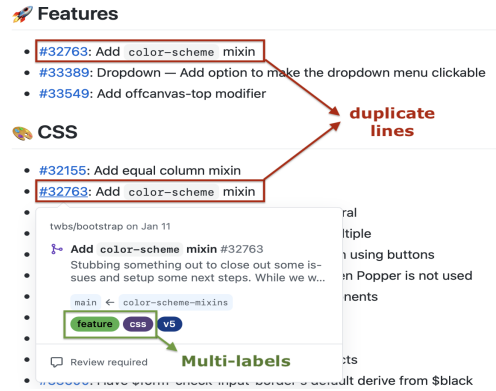


Fig. 7: Content Duplication



Fig. 8: Inconsistency

issues are tagged by multiple labels, e.g., feature and css. Surprisingly, we notice that the duplication issue has not appeared in Python projects' release notes. Therefore, we analyze the release notes of the python projects and find that release note producers prepare release note contents manually. For example, a *content* from the release note of version 1.3.0 of Pandas is *Performance improvement in DataFrame.corr() for method=kendall* and this content link with issue. However, the *issue title* is *DataFrame.corr(method="kendall") calculation is slow* However, producing release notes manually takes lengthy time around 2 to 4 hours [14]. In future, we need to consider the duplication problems to prepare the release notes using automated tools.

Inconsistency between Content Headings and Labels. The second problem is an inconsistency between the contents' headlines and tags. Our analysis identifies the similarity between the contents' headlines and tags. Table V shows the examples of headings and issue tags. We can see that those issues are labelled as *bug*, *type:bug*, usually those are listed under *Bug Fixes* or *Fixed* headings. However, in some cases, we find inconsistency. Fig. 8 represents an example of inconsistency issue. For example, an issue is labelled by *improvement* tag; however, this issue is written under *Bug Fixes* heading in the release note. It is possible to eliminate the inconsistency issue, while the automated release notes generation techniques will be considered the labels to classify the contents.

C. Threats to Validity

Internal validity- The content of release notes can be different based on the project domains, e.g., application software and software tools. Our study selects popular open-source projects' repositories in GitHub based on the most stars (i.e., more than 8,000 stars) and these projects belong to different

domains. We analyze the three top most popular languages' projects, e.g., JavaScript, Java and Python, (details in Table II) based on the active repositories to mitigate the internal threat. Moreover, we sort the projects based on the total number of releases and mitigate those with less than 30 releases.

External validity- The total number of participants of this survey study is may not be enough to understand the real-life scenarios about release notes usage in software development practice. To mitigate this threat, on average, our study participants have 3.5 years of experience in software development and use version control systems, e.g., Git, to keep track of the changes in the system.

VI. RELATED WORK

Empirical Studies. Many empirical studies have been done in prior works. Among them, some studies investigate the documented information types in release notes [4], [7], [13] (detailed discussion in Section II). Moreover, they identified that the provided information could differ in major and minor release notes from system to system. Klepper et al. [4] propose a technique for manually tailoring release notes by a release manager while considering three different kinds of stakeholders, e.g., users, customers and team members in agile software development.

Tools and Techniques. Several studies aim to generate release notes automatically or semi-automatically [6], [12], [14], [15]. A semi-automatic approach proposed by Klepper et al. [12] for generating audience-specific release notes. This approach used issue tracker, version control system and build server as data sources for release notes generation and the release manager tailored the contents for different target users. Moreno et al. [14] proposed another tool (i.e., ARENA) for Java projects. This tool uses external tools, e.g., change extractor, issue extractor, commits-issues linker and code change summarizer, to generate the complete release notes. Ali et al. [6] follow a similar approach to generate release notes for Node.js projects. Both are language-specific tools. Contrary, Nath et al. [15] proposed a release notes generation technique by applying text summarization techniques (i.e., TextRank) using related commits and pull-requests. This approach selects top-ranked contents. Therefore, some important sentences might be missed to generate automated release notes. From prior studies, we find that the proper investigation is required to understand the input sources for the documented contents of existing release notes. Therefore, we investigate more than 3,000 release note contents and structure from GitHub.

VII. CONCLUSION

We conduct an exploratory study and a user survey study on the release note contents to improve the quality of release notes production based on different practitioners' needs because, which are not explored in the existing studies. Our first study identifies several artifacts, e.g., commits, issues and pull-requests, from our empirical study which can be linked with release note contents. In addition, we detect other key artifacts, e.g., tags and milestones, that can assist in classifying and

maintaining an exemplary structure of the contents. Since different practitioners get benefits by using release notes, it is essential to understand what information needs to be included for different users. Keeping this in mind, we conduct a survey study to understand users' specific requirements. Moreover, we discuss our findings of release note production and existing issues of the release note contents. We are currently working on the three additional research questions to extend our research: (i) *What is the efficient way to generate release notes automatically using the relevant artifacts?*; (ii) *How can we integrate user-specific requirements to produce release notes?*; and (iii) *How to assess the quality of release notes?*

ACKNOWLEDGMENT

This research is supported by the Natural Sciences and Engineering Research Council of Canada (NSERC), and by two Canada First Research Excellence Fund (CFREF) grants coordinated by the Global Institute for Food Security (GIFS) and the Global Institute for Water Security (GIWS).

REFERENCES

- [1] Dataset. <https://github.com/sristorymana/Human22>.
- [2] Github. <https://github.info>.
- [3] Vue v2.6.13. <https://github.com/vuejs/vue/releases/tag/v2.6.13>.
- [4] Surafel Lemma Abebe, Nasir Ali, and Ahmed E. Hassan. An empirical study of software release notes. *Empirical Softw. Engg.*, 21(3):1107–1142, June 2016.
- [5] Emad Aghajani, Csaba Nagy, Mario Linares-Vásquez, Laura Moreno, Gabriele Bavota, Michele Lanza, and David C. Shepherd. Software documentation: The practitioners' perspective. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering, ICSE '20*, page 590–601, New York, NY, USA, 2020. Association for Computing Machinery.
- [6] M. Ali, A. Aftab, and W. H. Butt. Automatic release notes generation. In *2020 IEEE 11th International Conference on Software Engineering and Service Science (ICSESS)*, pages 76–81, 2020.
- [7] 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*, pages 1–1, 2020.
- [8] Jordi Cabot, Javier Luis Cánovas Izquierdo, Valerio Cosentino, and Belén Rolandi. Exploring the use of labels to categorize issues in open-source software projects. In *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, pages 550–554, 2015.
- [9] Simon Cleveland and Timothy J. Ellis. Orchestrating end-user perspectives in the software release process: An integrated release management framework. *Adv. Hum. Comput. Interact.*, 2014:805307:1–805307:15, 2014.
- [10] Vue Commit. Commit message. <https://github.com/vuejs/vue/commit/cd57393fd3e2c169d450607bc4f03652d106bcc2>.
- [11] Vue Issue. Issue title. <https://github.com/vuejs/vue/issues/11391>.
- [12] Sebastian Klepper, Stephan Krusche, and Bernd Brügge. Semi-automatic generation of audience-specific release notes. In *CSED@ICSE*, 2016.
- [13] Laura Moreno, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, Andrian Marcus, and Gerardo Canfora. Automatic generation of release notes. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2014*, page 484–495, New York, NY, USA, 2014. Association for Computing Machinery.
- [14] Laura Moreno, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, Andrian Marcus, and Gerardo Canfora. Arena: An approach for the automated generation of release notes. *IEEE Transactions on Software Engineering*, 43:106–127, 2017.
- [15] Sristory Sumana Nath and Banani Roy. Towards automatically generating release notes using extractive summarization technique. In *Proceedings of the 33rd International Conference on Software Engineering & Knowledge Engineering, (SEKE)*, pages 241–248, 2021.
- [16] Vue Pull-request. Pull-request title. <https://github.com/vuejs/vue/pull/11392>.