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

An Empirical Study of Software Release Notes

**Surafel Lemma Abebe, Nasir Ali,
and Ahmed E. Hassan**

the date of receipt and acceptance should be inserted later

Abstract Release notes are an important source of information about a new software release. Such notes contain information regarding what is new, changed, and/or got fixed in a release. Despite the importance of release notes, they are rarely explored in the research literature. Little is known about the contained information, *e.g.*, contents and structure, in release notes.

To better understand the types of contained information in release notes, we manually analyzed 85 release notes across 15 different software systems. In our manual analysis, we identify six different types of information (*e.g.*, caveats and addressed issues) that are contained in release notes. Addressed issues refer to new features, bugs, and improvements that were integrated in that particular release. We observe that most release notes list only a selected number of addressed issues (*i.e.*, 6-26% of all addressed issues in a release). We investigated nine different factors (*e.g.*, issue priority and type) to better understand the likelihood of an issue being listed in release notes. The investigation is conducted on eight release notes of three software systems using four machine learning techniques. Results show that certain factors, *e.g.*, issue type, have higher influence on the likelihood of an issue to be listed in release notes. We use machine learning techniques to automatically suggest the issues to be listed in release notes. Our results show that issues listed in all release notes can be automatically determined with an average precision of 84% and an average recall of 90%. To train and build the classification models, we also explored three scenarios: (a) having the user label some issues for a release and automatically suggest the remaining issues for that particular release, (b) using the previous release notes for the same software system, and (c) using prior releases for the current software system and the rest of the studied software systems. Our results show that the content of release notes vary between software systems and across the versions of the same software system. Nevertheless, automated techniques can provide reasonable support to the writers of such notes

Surafel Lemma Abebe, Nasir Ali, and Ahmed E. Hassan
Software Analysis and Intelligence Lab (SAIL)
School of Computing, Queen's University, Canada
E-mail: {surafel,nasir,ahmed}@cs.queensu.ca

with little training data. Our study provides developers with empirically-supported advice instead of simply relying on adhoc advice from on-line inquiries.

1 Introduction

Release notes are important software system trails [20]. Release notes are usually produced when a new version of a software system is released. The notes serve as a means of communication between the developers and the users of a software system. In addition, they help to establish expectations by providing information such as "known issues" [13, 33]. Consulting release notes is an essential best practice that is advocated in order to avoid field problems [13]. The importance and significance of release notes is echoed throughout several postings and news articles (*e.g.*, [33]). Despite the important role of release notes in providing quick and general information about a software release, little is known about release notes. Earlier versions of the *SWEBOK* had planned to tackle the content of release notes [7]. However, the most recent version of *SWEBOK* removed all references to the topic release notes [3].

Nevertheless, the content of release notes and best practices remain an important topic that is of interest to release notes writers. For example, the top 20 results of a Google search, using the query *"release notes" how to write*, shows 12 web pages of different websites which describe best practices to write such notes. Often, people ask about the best practices in writing release notes on question and answering websites. For example, on *StackOverflow*, we found the question "*How should release notes be written?*" [2], while on *Programmers StackExchange*, we found questions about "*good practices of writing release notes*" [1]. The answers to these questions, however, vary with each person sharing his or her own experiences and views. In addition, there exists a Wikipedia page on the topic of release notes [5]. While there are many listings of advices worldwide, they are ad-hoc personal views without a strong empirical backing to them. The differences in the contents of release notes could also be observed by looking at existing release notes. For example, Figure 1 shows release notes which have differences in the contained information. In particular, we find that the release notes for Eclipse often provide detailed information such as *target operating environment*, while the release notes for ThunderBird are usually much more concise and contain less information. In short, there appears to be little consensus on what is and what should be listed in release notes.

Despite the variation in the contents of release notes, we believe that release notes share the same objective and, hence, can be organized using a given set of information types. Identifying the different information types that are communicated in release notes is the first step to understand them. The identification of the information types commonly used in the release notes will help us derive best practices and guidelines to assist the writers of release notes. To the best of our knowledge, this paper is a first attempt to understand the different information types that are contained in release notes. We performed an empirical study on software release notes in order to identify the different challenges related to writing release notes.

In our empirical study, we explored the different information types that are included in release notes. Our study shows that:

Six different information types exist based on the analysis of 85 release notes taken from 15 software systems. Almost 70% of the manually evaluated release notes contain the title, system overview, addressed issues, and caveat information types.

We also observed that all too often, the notes omit listing many of the issues that were addressed in the release (*i.e.*, only 6-26% of the addressed issues are listed in eight release notes of the three software systems that we closely examined). In the second part of our empirical study, we analyzed the different factors which are related to issues and which could be used to explain the likelihood of an addressed issue appearing in the release notes. We find that:

Issue type, *number of comments*, *description size*, *days to address an issue*, *number of modified files* and *reporter's experience* are important factors in explaining the likelihood of an issue being listed in release notes.

The selection of issues requires a good knowledge of all addressed issues and the relevance of these issues to the release notes readers. We believe that the selection of the relevant addressed issues is one of the challenges that release notes writers must tackle everytime they prepare such notes. To address this challenge, we investigate automated approaches (through machine learning techniques) which suggest issues that should be listed in release notes. We find that:

Issues listed in all studied release notes can be determined with an average precision of 84% and an average recall of 90%.

Machine learning techniques use a training dataset to build a model. In our study, we investigated three training scenarios that could be used to build a model and suggest issues that should be listed in release notes. The training scenarios are: (a) having the user label some issues for a release and automatically suggesting the remaining issues for that particular release, (b) using a previous release notes for the same software system, and (c) using prior releases for the current software system and the rest of the studied software systems. The results show:

The first scenario, having the user label some issues for a release and automatically suggesting the remaining issues for that particular release, gives the best result as compared to the other training scenarios.

Our findings could be used as a starting point for further studies to create a guideline for writing release notes. The results show that our approach could also be used to help release notes writers automatically identify issues to be listed in release notes.

Paper organization: Section 2 presents our exploratory study on the contents of release notes. Sections 3 and 4 discuss factors that could be used to explain the likelihood of an issue being listed in release notes. Section 5 explores different scenarios that could be used to automatically suggest which issues should be listed in the release notes of a particular release. Sections 6 and 7 present threats to validity and related works, respectively. Section 8 discusses our conclusion and higlights avenues for future work.

2 Information types that are included in release notes

We collected release notes from 15 open source software systems of various domains (see Table 1). Table 1 shows the list of software systems, their type, release versions, and total, average and median of the number of words in release notes for each release type, *i.e.*, major and minor. When there is only one version of a software system, average and median values are not computed. The notes are collected from the Websites of the respective software systems. We collected more release notes of a software system if the release notes of that software system is small in terms of the number of words in the release notes, *e.g.*, Dropbox release notes. We believe that the larger number of small release notes helps us observe possibly different release notes contents for such systems.

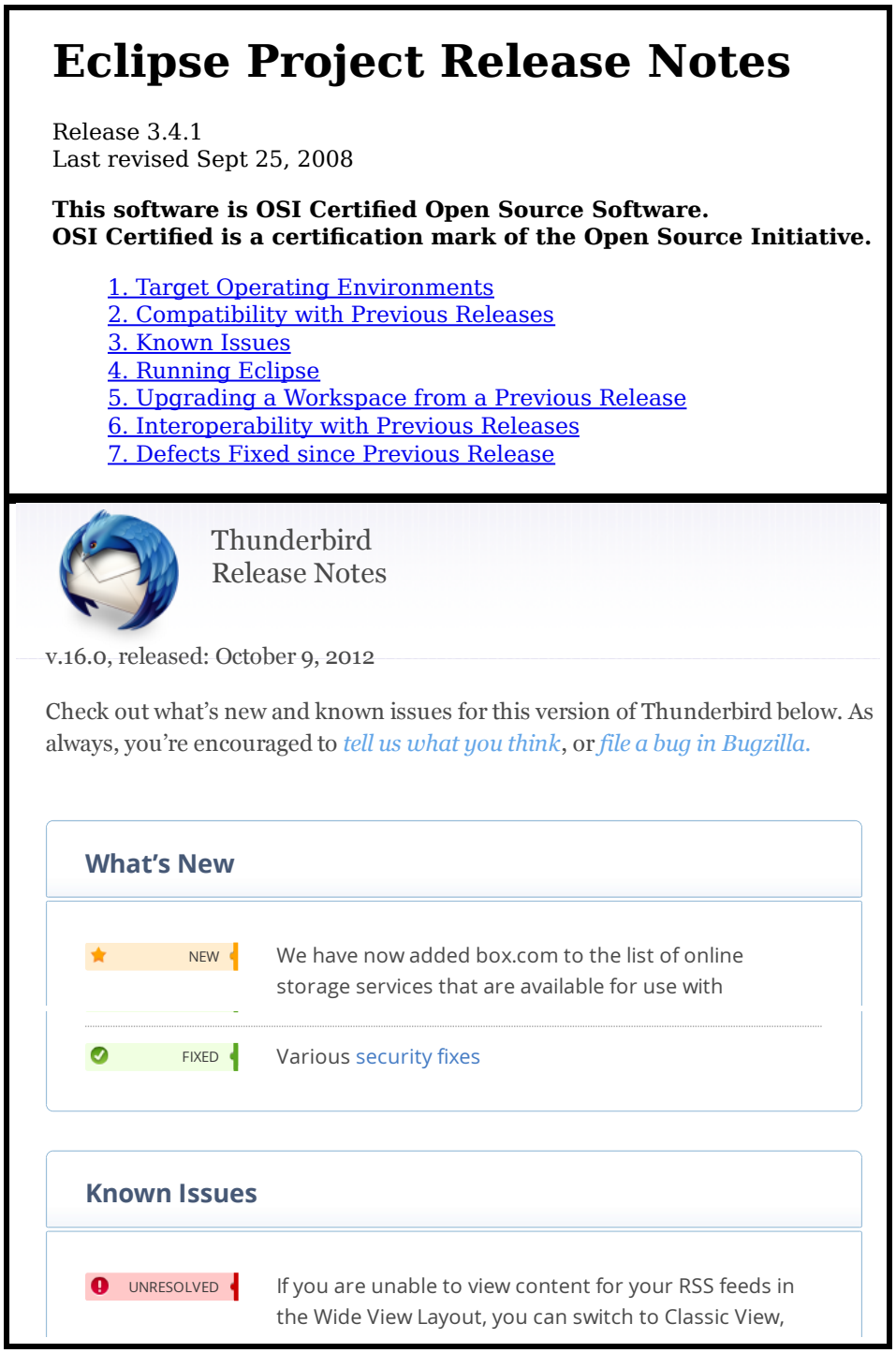
From each studied software system, we examined the release notes for at least one minor or one major release of the software system. To identify whether a release is a minor or major, we consulted several on-line system documentation and discussion forums.

We followed a grounded theory approach [16,30], to identify the types of information that is commonly included in the release notes. Grounded theory is a systematic approach where one derives abstract concepts from the subjects under observation. Grounded theory involves several steps where the observer incrementally collects data and identifies concepts from the data. For our case, the concepts are the types of information in the release notes. The concepts are then categorized and refined as additional data is introduced in the study. For example, if a release notes has information about *caveats* and *caveats* was not included in the list before then we added *caveats* in the information type pool. We followed the same process for all collected release notes. When related information types are found in the pool of information types, we combined the related information types into a new more general and representative information type. For example, the information types *new features*, *improvements* and *bug fixes* are combined into *addressed issues*. *New features*, *improvements* and *bug fixes* are usually found in different sections of release notes.

To evaluate whether the identified information types could be associated with the content of individual release notes, we manually classified the contents of 10 randomly selected release notes along the identified information types. We selected 10 release notes because we believed the results of the 10 release notes would give a good picture of the association. The 10 release notes cover more than 65% of the software systems that we studied. The classification was performed by the first author and verified by the second author. When there was a disagreement between the identified types, both authors held a discussion and consulted the third author to resolve the disagreement. The disagreement occurred 6 times, which is only 10% (6/60), while assigning the information types to release notes contents as shown in Table 3. It took 20 minutes to 3 hours in order to read and categorize the contents of one release notes.

2.1 Results and discussion

Six information types are identified in the release notes. In our study, we identified the information types that are presented in Table 2, along with a brief



Eclipse Project Release Notes

Release 3.4.1
Last revised Sept 25, 2008

This software is OSI Certified Open Source Software.
OSI Certified is a certification mark of the Open Source Initiative.

Thunderbird
Release Notes

v.16.0, released: October 9, 2012

Check out what's new and known issues for this version of Thunderbird below. As always, you're encouraged to *tell us what you think*, or *file a bug in Bugzilla.*

What's New

NEW — We have now added box.com to the list of online storage services that are available for use with

FIXED — Various security fixes

Known Issues

UNRESOLVED — If you are unable to view content for your RSS feeds in the Wide View Layout, you can switch to Classic View,

Fig. 1: Excerpts of the release notes for Eclipse 3.4.1 and ThunderBird 16.0.

Table 1: List of systems and versions for which we explored the release notes. Average, median, and St. deviation are not computed when only one release version is considered.

No.	System	System type	Release		Words in Release notes		
			Type	Versions	Total	Average	Median (St. Dev.)
1	Debian-32bit PC	OS	Minor	3.1	8,862		
			Major	7	8,427		
2	DropBox	Cloud storage service	Minor	2.0.2, 2.0.4, 2.0.5, 2.0.6, 2.0.7, 2.0.8, 2.0.10, 2.0.12, 2.0.16, 2.0.21, 2.0.22, 2.0.26, 2.2.0, 2.2.1, 2.2.2, 2.2.3, 2.2.8, 2.2.9, 2.2.10, 2.2.12, 2.2.13	1,047	50	28 (±52)
			Major	2.0.0	22		
3	Eclipse	IDE	Minor	3.4.1, 3.4.2, 3.5.1, 3.5.2, 3.6.1, 3.6.2, 3.7.1, 3.7.2,	127,608	15,951	16,033 (±1,012)
			Major	3.3.0, 3.4.0, 3.5.0, 3.6.0, 3.7.0, 4.2.0, 4.3.0	92,704	13,243	13,045 (±839)
4	Fedora	OS	Major	18, 19	19,029	9,515	9,515 (±1,084)
5	Firebug	Web development tool	Minor	1.12.2	1,433		
			Major	1.10.0, 1.11.0	2,623	1,312	1,312 (±450)
6	Firefox	Browser	Minor	23.0.1	1,267		
			Major	23, 24	2,322	1,161	1,161 (±62)
7	GCC	Compiler	Minor	3.1.1, 3.2.1, 3.2.2, 3.2.3, 3.3.1, 3.3.2, 3.3.3, 3.3.4, 3.3.5, 3.3.6	8,989	899	813 (±712)
			Major	3.2, 3.3, 4.8	9,029	3,010	3,873 (±2,212)
8	GIMP	Image editing tool	Major	2.4, 2.6, 2.8	5,634	1,878	1,529 (±689)
9	Hadoop	Software framework	Minor	0.18.0, 0.19.0, 0.20.0, 1.1.2	48,544	12,136	1,854 (±21,250)
10	JIRA	Issue tracking tool	Minor	5.2	2,680		
			Major	6.0	1,880		
11	LibreOffice	Office suite	Minor	4.0.4	627		
			Major	4.1.0	652		
12	Lucene	Information retrieval library	Major	3.4, 3.5	1,015	508	508 (±25)
13	Rubby on Rails	Web application framework	Major	3.0, 3.1, 3.2, 4.0	16,169	4,042	4,056 (±1,289)
14	Solr	Search platform	Minor	3.6.2, 4.2.1, 4.3.1	1,092	364	364 (±7)
			Major	3.4, 3.5, 3.6	1,463	488	458 (±53)
15	Thunderbird	Mail client	Minor	17.0.7	344		
			Major	16	485		

description of the information type and examples of each identified type. The identified information types are not dependent on a given domain as the types are identified using software systems taken from different domains. However, most of the information types could be shared by systems from the same domain. For example, the software systems from the OS domain, Debian-32bit PC and Fedora, share most of the information types (see Table 3). DropBox, on the other hand, has the least common information types with either Debian-32bit PC or Fedora. In addition to the domain of the software systems, other characteristics of the software systems (*e.g.*, software size and user base) could impact what is included in the release notes. Future studies should further investigate the information types in release notes with respect to the characteristics of the software system.

At least 70% of the manually examined release notes contain the title, system overview, addressed issues, and caveat information types. Table 3 shows the result of our manual evaluation of the identified information types on the ten randomly selected release notes. An information type corresponding to the release

Table 2: Information types in Release Notes

Information type	Description	Example
Title	Title provides specific information about the released version. Title is used to present unique identifiers of the released software. The information included in this information type are: name and version of the released software. In some release notes, the date on which the release notes is created, last updated, or revised is also included in the title.	In Debian 7.0 the title of the release notes is *Debian 7.0 "Wheezy" released*. LibreOffice and ThunderBird have the created, last updated, or revised information as well.
System overview	System overview provides general information about the released software system. The overview is used to present a summary of the main functionalities of the software system. In some release notes, it also includes information about licenses and copyrights. In most release notes, overview is one of the first sections of the release notes. Some release notes do not include this information type.	LibreOffice release notes states that LibreOffice source code is licensed under the GNU Lesser General Public License (LGPLv3).
Resource requirement	Resource requirement provides information about supported platform (*i.e.*, hardware and software requirements). The descriptions about the environment or type of machine required to install and use the software system are categorized under this information type. Resource requirement refers to information such as the operating system, hardware, or architecture of the machine supported by the software. In addition no longer supported platforms are described under this information type.	Eclipse 3.7.1 release notes indicates that Eclipse 3.7.1 supports x86 32-bit, x86 64-bit, Power 64-bit, SPARC 32-bit, ia64 32-bit, Universal 32-bit, and Universal 64-bit hardware platforms.
Installation	Installation provides instructions on how to make the released software operational. Instructions, which are provided on how to install, configure, upgrade the software, are categorized under this information type. We observed that such information is not always provided in release notes.	JIRA 6.0 release notes provides instructions on how to upgrade to JIRA 6.0 from JIRA 5.2.x installations.
Addressed issues	Addressed issues provides information about the integrated changes, *i.e.*, work done, in the released software. Addressed issues include *new features*, *bug fixes*, and *improvements*.	Firefox 23 release notes describes two new features: options panel created for Web Developer Toolbox, and a user protection mechanism from man-in-the-middle attacks and eavesdroppers on HTTPS pages.
Caveat	Caveat provides information about open issues and problems of which a user should be aware. Caveat information type usually indicates the causes of the issues and problems, and future plans to address them.	Eclipse 3.7.1 release notes describes installation and configuration issues that can cause Eclipse to fail to start.

notes is marked with a check mark (✓), if the release notes contain information which in part or fully corresponds to the description of the information type. Otherwise the information type corresponding to the release notes is left blank.

Table 3: Classification of release notes contents based on information types.

Release notes			Title	System overview	Resource requirement	Installation	Addressed issues	Caveats
System	Version	Type						
Debian-32bit PC	7	Major	✓	✓	✓	✓	✓	✓
DropBox	2.2.10	Minor	✓				✓	
Eclipse	3.7.1	Minor	✓	✓	✓	✓	✓	✓
Fedora	18	Major	✓	✓	✓		✓	✓
Firefox	23	Major	✓	✓			✓	✓
GCC	4.8	Major	✓				✓	✓
Hadoop	0.19.0	Minor	✓	✓			✓	
JIRA	6.0	Major	✓	✓		✓	✓	✓
LibreOffice	4.1.0	Major	✓	✓			✓	✓
ThunderBird	17.0.7	Minor	✓	✓			✓	
Percentage of ✓			**100**	**80**	**30**	**30**	**100**	**70**

Addressed issues and title information types are found in all of the ten studied release notes. In our manual analysis, we observed that all release notes have a title and list either of the three types of addressed issues: *new features*, *bug fixes*, or *improvements*. Resource requirement and installation information types are found in only 30% of the manually analyzed release notes. The absence of such information types are due to the fact that release notes are not the only sources of information for software systems. When a software is released, additional sources of information such as user manuals and *README* files which could contain such information types are also prepared. Hence, the information types which are not included in release notes of some software systems could be found in the additional documents.

Release notes are user-oriented. In our manual evaluation, we observed that all release notes are mainly written for users who would not read through the source code of the software system. Only 30% of the manually analyzed release notes have additional information for developers. The additionally provided information for developers include newly introduced subroutines and debugging features, and developer-relevant incompatibilities. The release notes which have additional information for developers are Fedora 18, Firefox 23, and Hadoop 0.19.0.

The information types provided in major and minor releases differ from system to system. While identifying the information types, we observed that release notes of some software systems provide different information in their release notes for major versus minor releases, while in others they provide the same information types. For example, in the release notes for major releases of Eclipse, addressed issues are not provided; while the release notes for minor releases provide such information. However, both the release notes for the major and minor versions of ThunderBird include the addressed issues information type. Looking into the number of words in the release notes of major and minor versions of Eclipse, the release notes for the minor versions have 38% more words than the release notes for the major versions. For ThunderBird, however, the release notes for the mi-

nor version has 29% less words than the release notes for the major release (see Table 1).

Table 4: Temporal changes of the information types across several release notes for Eclipse. Other than Addressed issues, all types appear across all releases.

Eclipse release		Information types					
Version	**Type**	**Title**	**System overview**	**Resource req.**	**Installation**	**Addressed issues**	**Caveat**
3.3.0	Major	✓	✓	✓	✓		✓
3.4.0	Major	✓	✓	✓	✓		✓
3.4.1	Minor	✓	✓	✓	✓	✓	✓
3.4.2	Minor	✓	✓	✓	✓	✓	✓
3.5.0	Major	✓	✓	✓	✓		✓
3.5.1	Minor	✓	✓	✓	✓	✓	✓
3.5.2	Minor	✓	✓	✓	✓	✓	✓
3.6.0	Major	✓	✓	✓	✓		✓
3.6.1	Minor	✓	✓	✓	✓	✓	✓
3.6.2	Minor	✓	✓	✓	✓	✓	✓
3.7.0	Major	✓	✓	✓	✓		✓
3.7.1	Minor	✓	✓	✓	✓	✓	✓
3.7.2	Minor	✓	✓	✓	✓	✓	✓
4.2.0	Major	✓	✓	✓	✓		✓
4.3.0	Major	✓	✓	✓	✓		✓

Temporal changes of information are dependent on the type of release. To investigate if there are temporal changes of information as the software is upgraded, we analyzed GCC, Eclipse, and DropBox (see Table 1). The systems are selected because we have a large number of versions for them. Table 4 shows the results for Eclipse. In GCC and Eclipse, we observed that the information types contained in the release notes rarely change between major and minor releases. For example, Eclipse has addressed issues only in the minor releases (see Table 4). For DropBox, however, no change of information type is observed across the versions investigated.
Release notes writers follow three different styles when describing the addressed issues. Addressed issues is one of the main information types in release notes. The addressed issues information type incorporates information about three types of addressed issues in the released software: *new features*, *bug fixes*, and *improvements*. While exploring the release notes, we observed that most release notes writers follow three different styles when describing the addressed issues in a release. We describe each style below:

- ***Style I - Consolidated summary of selected issues:*** Release notes writers rephrase and summarize the addressed issues in the current release. In some release notes, writers consolidate the summaries of related issues into paragraphs (*e.g.*, GCC 4.3, JIRA 6.1 release notes), while in others they select issues and provide a summary for each issue (e.g. DropBox 2.2.12 release notes). These summaries do not include any references to the corresponding issues in the repository where issues are tracked. In 80% of the release notes that are shown in Table 3, release notes writers follow *Style I* when documenting the issues in the release notes.

- ***Style II - List of selected issues:*** Release notes writers do not summarize issues but simply list some selected issues that were addressed in the current release. As a description of the issue, writers usually use the title of the issue as provided in the issue repository (*e.g.*, Hadoop 0.19.0 release notes). In 10% of the release notes that are shown in Table 3, release notes writers follow *Style II* when documenting the issues in the release notes.
- ***Style III - List of all issues:*** Release notes writers provide a list of all addressed issues in the current release without providing any information on a selected list of addressed issues which are considered worth expanding on. An example of release notes with such style is the release notes of Eclipse 3.4.1. In this release notes, the writers list all the addressed issues with the issue ID and title. Release notes which provide only a link to the issue repository without any highlighting of the selected issues are also grouped under this style. In 10% of the release notes that are shown in Table 3, the release notes writers follow *Style III* when documenting the addressed issues in the release notes.

3 Why are only some of the issues listed in the release notes?

From our exploratory study in Section 2, we notice that, all too often, not all addressed issues are listed in the release notes. Usually, adding all the addressed issues in release notes, as described in *Style III* above, makes the release notes too long. Consequently, lengthy release notes might be less useful for readers who are interested in knowing the important addressed issues in the current release. Release notes writers often try to list the most important issues in the release notes. A reply to a good practice question about writing release notes on the *Stack Exchange* supports our statement. The reply states the following: *"The most important thing about release notes is to be aware that every additional sentence loses another 10% of the readers. So you must strictly prioritize what your current users need to know about the release"*[1].

To strike a balance between being informative and not too lengthy, release notes writers commonly follow the aforementioned *Style I* and *Style II*. In these styles, selected issues are listed or summarized in the release notes. From the ten software release notes listed in Table 3, only the release notes for Eclipse 3.7.1 lists all addressed issues in the given version. The remaining nine release notes provide information on a selected number of addressed issues in the corresponding version. For example, for Lucene 3.5, 9 of 72 addressed issues are listed as *highlights* in the corresponding release notes. Selecting which issues to list from a long list of addressed issues in a given version is a non-trivial decision for the writers of release notes. The selection process requires complete knowledge of the issues and their importance from both the user and developer point of view.

3.1 Factors which influence the likelihood of addressed issues being listed in the release notes

To the best of our knowledge, there are no studies which explores the rationale for listing or not listing an addressed issue in the release notes. We define nine different factors relating to issues in order to understand the rationale for an issue

being listed in the release notes. We, then, compute different analysis to see if the factors could be used to explain the likelihood of an addressed issue being listed in the release notes.

Table 5: Systems and number of listed issues in the corresponding release notes (RN).

System	Version	Addressed Issues		
		Listed in Release Notes	Total	Ratio of listed in Release Notes (%)
Hadoop	0.18.0	65	253	25.69
	0.19.0	63	274	22.99
	0.20.0	39	259	15.06
Lucene	3.4	7	50	14.00
	3.5	9	72	12.50
Solr	3.4	6	44	13.64
	3.5	5	40	12.50
	3.6	7	113	6.19

Investigated factors: Table 6 shows the factors that are investigated in our study. We selected factors which we felt could intuitively explain the importance of an issue from different dimensions. The selected factors have also been studied to address other software maintenance tasks [8, 18, 19, 28, 31, 35, 37, 41]. The factors are categorized along three dimensions: *issue information*, *change effort*, and *experience of people*.

Issue information dimension refers to factors which are collected from an issue report. The different values in the issue report could be used for different purposes, such as prioritizing an issue [19, 28, 37, 41]. For example, issues which are bug fixes and have higher priority are likely to be addressed ahead of others. We investigate if the factors under the issue information dimension can also be used to explain the likelihood of an issue being listed in release notes.

Change effort represents the effort put in addressing an issue [8, 18, 31]. We conjecture that issues which require more effort are more likely to be listed in the release notes. We measure change effort using two proxy factors: *days to address an issue* and *number of modified files*.

Experience of people dimension refers to the experience of the people involved in reporting and addressing an issue [14, 19, 32, 35]. People who are well-experienced usually identify and address major issues which are more likely to be listed in the release notes. We use reporter experience,*i.e.*, the number of issues that are reported by a reporter, committer experience, *i.e.*, the number of files that are committed by a developer, to quantify the experience of individuals.

Table 6 presents the factors that were collected for the three dimensions.

Studied release notes: For our study, we selected three release notes from Hadoop, two release notes from Lucene, and three release notes from Solr (see Table 5). These release notes are selected because they include the IDs of the addressed issues that are listed in the release notes as described in the aforementioned *Style II*. The issue IDs help to accurately map the addressed issues that are listed in release notes to the corresponding issues in the issue repository. Release notes that use *Style I* have a consolidated summary of selected issues that does not include

Table 6: Studied factors across the three dimensions along with the used measure and the rationale for each factor.

Dimension	Factor	Measure	Rationale
Issue information	Issue type	count of New features, bugs, improvements	Different issue types are given different importance and, hence, could influence the likelihood of an issue to be listed in release notes.
	Priority	count of Blocker, critical, major, minor, trivial	Issues with higher priorities are more likely to be listed in release notes.
	Number of comments	Count of comments per issue	Long discussions could indicate difficult or important issues which are more likely to be listed in release notes.
	Summary size	Count of words	When listing all the addressed issues, one might actually opt for shorter titles to prevent the release notes from getting too long.
	Description size	Count of words	A long description could indicate a complex and difficult issue that is likely to be listed in release notes.
Change effort	Days to address an issue	Number of days	Issues which take longer time to be addressed could be a major and, hence, important issue that are likely to be listed in release notes.
	Number of modified files	Count of modified files to address an issue	Issues requiring modification of a number of files could be major ones, and hence, important issues that are more likely to be listed in release notes.
Experience of people	Reporter experience	Number of issues reported by a reporter	Issues reported by an experienced reporter are more likely to be listed in the release notes.
	Committer experience	Number of files committed	Experienced developers are likely to work on important issues which are more likely to be listed in the release notes.

issue IDs. The absence of issue IDs makes mapping the selected issues with the entries in the issue repository difficult. Release notes that use *Style III*, on the other hand, list all addressed issues in the release notes which makes the notes unsuitable for our investigation. From the studied release notes, only Hadoop, Lucene, and Solr have IDs of addressed issues in their release notes. The existence of such mapping ensures accurate and unbiased mapping of issues to the corresponding content in the release notes. Table 5 presents the number of issues that are listed in the release notes and the total number of addressed issues in the corresponding version. The ratio of issues listed in the release notes ranges from 6 to 26%. All the three systems use the Apache JIRA issue repository to manage their issues.

Data collection: To compute the factors for each issue, we mined the issue repository of each system. The issue repository provides us with various details about an issue, *e.g.*, all comments for an issue, days to address an issue, and issue priority. Some information about an issue, *e.g.*, the number of modified files to address an issue, are not available in issue repositories. All information regarding added, modified, or deleted files is stored in the commit logs of the corresponding

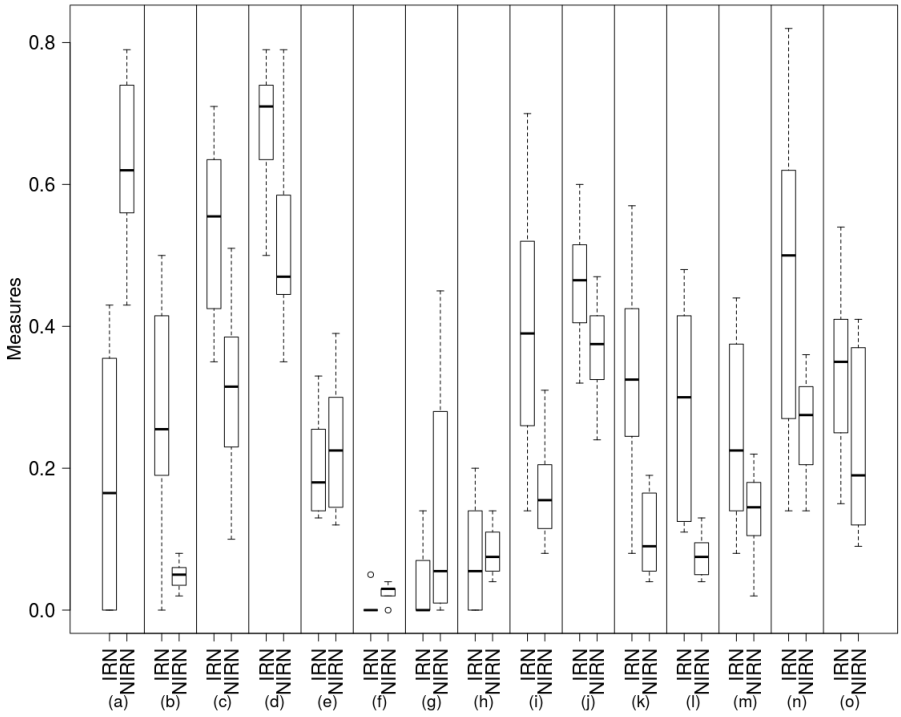


Fig. 2: Comparison of issues in release notes (IRN) and not in release notes (NIRN) for each factor. The x-axis labels correspond to the labels listed in Table 7 and y-axis shows the average of measures.

version control system of the studied software systems. Thus, we link commit logs with issue reports to gather more information about an issue.

Developers often add an issue tracking number in the commit log message when they address an issue that is reported in the issue tracking system. We use regular expressions, *i.e.*, a simple text matching approach, to link an issue with a commit log. On average, we accurately linked 89% of the issues to their commit logs. Nevertheless, more complex linking techniques, such as [10, 42], could be used.

After linking each issue with the commit logs, we collected the measures for each factor related to an issue as described in Table 6. For issue types, *e.g.*, feature requests, and issue priority, *e.g.*, major, we counted the number of occurrences of each issue type and issue priority of an addressed issue listed in release notes and not listed in release notes. We took the percentage of each issue type and issue priority. For example, if there are ten issues in release notes and four of them are feature requests then we divide four by ten, *i.e.*, 0.4, to get the percentage of the issue type of new features. For all other factors, we first normalize the values. For example, we divided the total number of comments of an issue by the maximum number of comments that are posted across all the issues in the same release of a software system. Hence, the values of all the factors range between 0 and 1.

3.2 Results and discussion

Issue type, number of comments, description size, days to address an issue, number of modified files and reporter experience explain the likelihood of listing an addressed issue in the release notes. Figure 2 shows the distribution of the measures in release notes (IRN) and not in release notes (NIRN) for each factor. In comparison with issue type of bugs, issue types of new features and improvements have higher likelihood to be listed in a release notes. Issue priority type *major* better explains if an issue will be listed in release notes. The higher value of issue priority type *major* shows that developers tend to list issues with issue type *major* in release notes more often than other issue priority types, *e.g.*, minor or blocker. Figure 2 shows that longer discussions, *i.e.*, total number of comments, and issue reporter's experience tend to explain the likelihood of an issue being listed in the release notes. The addressed issues by more experienced developers, *i.e.*, the developer who has made more commits than other developers, get listed in release notes more often than the addressed issues by less experienced developers.

A factor is said to explain whether an issue will be listed in release notes or not, if the difference between the IRN and NIRN average values is statistically significant. A statistical test assures that a difference between two distributions is not by chance or some extreme values. To test if the difference is significant, we performed a paired, two sided Wilcoxon signed rank test for each factor across the systems. A paired-wise statistical test compares two distribution to measure the difference between them. A difference is considered statistically significant if the p-value is less than 0.05. To test whether an issue will be listed in release notes, we pose a null hypothesis, *i.e.*, there is no statistical difference between the factor measures for issues listed in release notes and the issues that are not listed, is rejected.

Table 7 reports the p-values and the positive (+) and negative (-) likelihood of listing an addressed issue in release notes. In 8/15 of the cases, we reject our null hypothesis because the p-value is below the significance level, *i.e.*, 0.05. For example, the issue type of bugs has a negative (-) likelihood, hence issues that address bugs are likely not be listed in release notes.

Different software systems have different trends for each factor. Some factors could better explain the likelihood of an issue to be listed in release notes than others (see Figure 3). A higher bar shows that a factor explains better the likelihood of an issue to be included in the release notes over other factors. For example, major bugs are mostly included in a release notes. In addition, committers' experience has more influence on including an issue in the release notes than the experience of the issue reporter.

In the case of Hadoop, an issue was listed in the release notes if it belongs to the new features/improvements issue type, it has major issue priority, it has longer issue description size, and an issue took longer time, *i.e.*, number of days, to address. In contrast, release notes writers did not list an issue if the issue type or priority was a bug or trivial respectively. Other factors, *i.e.*, minor, critical, blocker issue types, number of comments per issue, summary counts of an issue, number of modified files to fix an issue, reporter experience, committer experience, do not have consistent trend in the three releases of Hadoop.

In the case of Lucene, developers listed an issue in the release notes if it is an improvement, it has higher number of comments, and long description. The

Table 7: Results of two sided Wilcoxon signed rank test and (+) and (-) likelihood of an addressed issue. (+) likelihood represents if a release notes writer will use the factor to list an addressed issue in the release notes. (-) likelihood represents if release notes writer will use the factor to exclude an addressed issue from the release notes.

Factor	**Labels**	**Measures**	**P-Values**	**Likelihood**
Issue Type	(a)	Count of bugs	***0.0078***	(-)
	(b)	Count of new features	***0.016***	(+)
	(c)	Count of improvements	***0.0078***	(+)
Issue Priority	(d)	Count of major	0.055	
	(e)	Count of minor	0.074	
	(f)	Count of critical	0.073	
	(g)	Count of trivial	0.14	
	(h)	Count of blocker	0.67	
Number of comments	(i)	Count of comments per issue	***0.021***	(+)
Summary size	(j)	Count of words	0.11	
Description size	(k)	Count of words	***0.014***	(+)
Days to address an issue	(l)	Number of days	***0.0078***	(+)
Number of modified files	(m)	Count of modified files to address an issue	***0.034***	(+)
Reporter experience	(n)	Number of issues reported by a reporter	***0.042***	(+)
Committer experience	(o)	Number of files committed	0.068	

developers did not list bugs in the release notes. All other factors do not have any consistent trend.

In the case of Solr, developers do not list bugs, and critical issues in the release notes. The trivial issues and blocker type of issues have no consistent trends. Issues related to all the other factors have higher likelihood of being listed in the release notes.

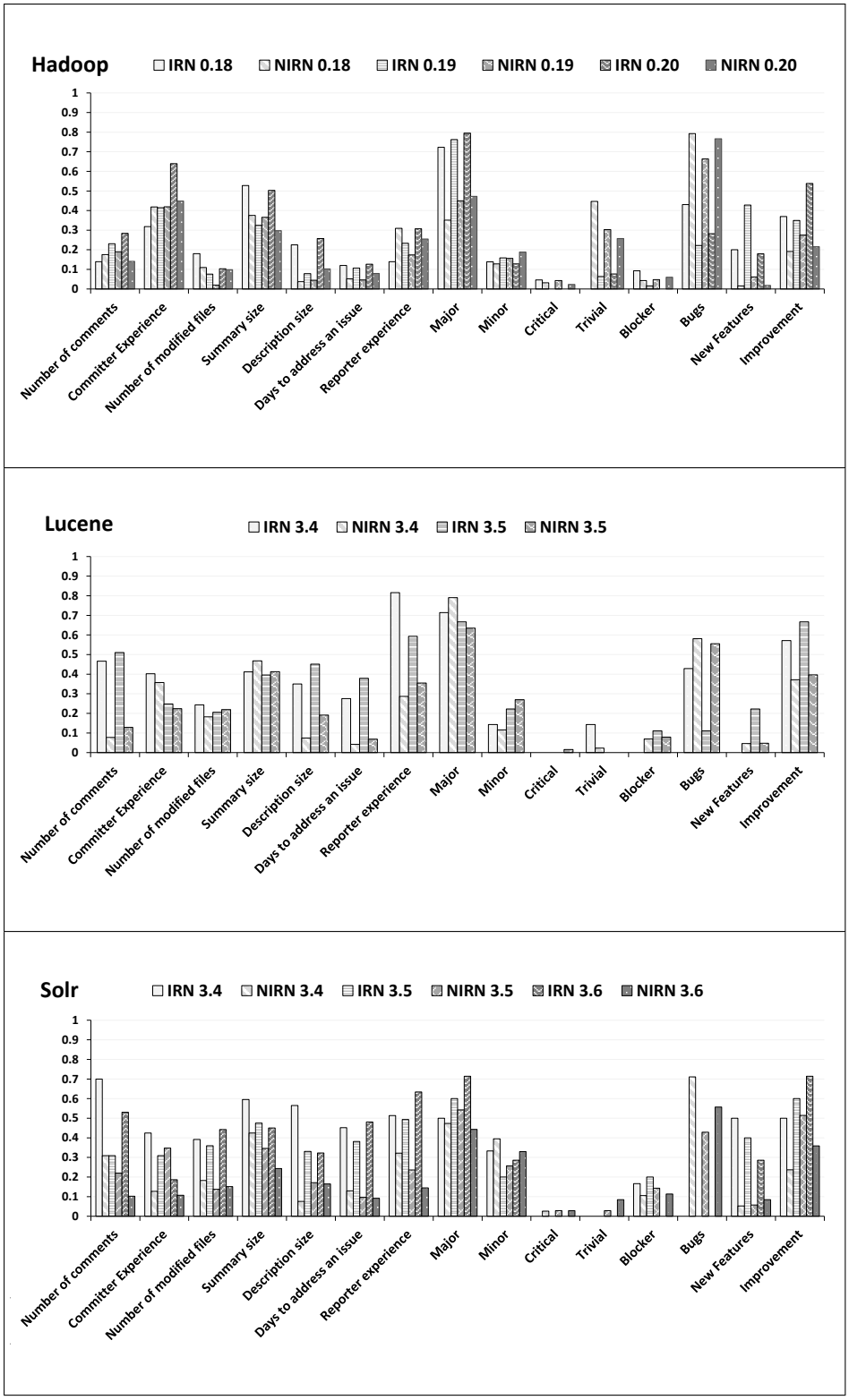
4 Automatically suggesting issues to be listed in release notes

Release notes writers usually select and list only a subset of the addressed issues in release notes (see Table 5). The selection of issues, however, requires a good knowledge of all issues that were addressed in that particular release of the software system and the relevance of these issues to the release notes readers. The selection of issues is not a one time task, however it must be conducted for a new release of a software system. Hence, we believe that automating this step helps to save the time and efforts needed to create release notes.

The results in Section 3 show that factors that are related to the addressed issues in the software system could be used to explain why some issues are selected and listed in release notes. In this Section, we investigate if the factors (see Table 6) can be used to build models that can automatically suggest issues to be listed in the release notes. We select three software systems, *i.e.*, Hadoop, Lucene, and Solr, to automatically classify issues to be listed in the release notes. The selection criteria is based on the availability of issue IDs in the studied release notes and commit logs. The availability of issue IDs helps us link issue reports to commit logs so we can compute our studied factors.

To automatically identify issues to be listed in release notes, we built models using four machine learning techniques: two *decision tree classifiers*, *simple logistics*,

Fig. 3: Measures for each factor (IRN = in release notes, NIRN = not in release notes). X-axis shows the factors and y-axis shows the measures for each factor.



and *support vector machine* (SVM). We used the implementations of these techniques that are provided in WEKA [21]. WEKA is a suite of tools for machine learning techniques. The implementations of the decision tree classifier used in this study are J48 [34] and Random Forest [15], while *sequential minimal optimization* (SMO) [26] is used for SVM. The implementation used for simple logistics is described in [40]. Below, we describe the variables and settings that we used to build the models.

Independent variables: are the factors which are listed in Table 6. We collected measures for each factor from the issue repositories and commit logs as described in the Section 3. To see if the factors are independent of each other, we measured the correlation between them. The correlation between the factors is measured using the Spearman rank correlation. The results show that the correlation numbers are low. Table 8 shows the Spearman rank correlation result for Hadoop 0.18.0. We also computed the p-values for the Spearman rank correlation. The results show that the correlation result for more than half of the 36 pairs is not significant. Hence, we considered all factors as our independent variables.

Table 8: Spearman rank correlation results for Hadoop 0.18.0

	Issue type	**Issue priority**	**Number of comments**	**Committer experience**	**Number of modified files**	**Summary size**	**Description size**	**Days to address an issue**	**Reporter experience**
Issue type	1								
Issue priority	0.30	1							
Number of comments	0.22	-0.04	1						
Committer experience	0.02	-0.09	0.03	1					
Number of modified files	0.31	-0.04	0.39	0.06	1				
Summary size	-0.04	0.04	0.09	0.20	-0.01	1			
Description size	-0.10	-0.11	0.33	-0.03	0.09	0.12	1		
Days to address an issue	0.27	0.23	0.52	0.08	0.34	0.20	0.24	1	
Reporter experience	0.09	0.01	0.24	-0.14	0.13	-0.03	0.12	0.06	1

Dependent variable: is the dichotomous variable *IRN* (in release notes) which indicates whether or not an addressed issue in a given version is listed in the corresponding release notes. The values for the dependent variable are *YES*, if the issue is listed in the release notes, and *NO* otherwise.

The distribution of the two values for the dependent variable is unbalanced. Such data imbalance biases the model to the majority and affects the results [11,36,37]. To address this problem, we used re-sampling. To carry out the re-sampling, we used the supervised re-sampling implementation provided in WEKA. The re-sampling technique gives a random balanced subsample of the dataset. The implementation gives the option to do the sampling with or without replacement and to bias the dependent class distribution towards a uniform distribution. In our experiment, we used sampling without replacement and biased the dependent class to a uniform distribution to balance the dataset. We did not re-sample the testing data.

To build and test the models, we used a 10-fold cross validation.

4.1 Evaluation metrics

Classifying the value of a dichotomous dependent variable is a classification problem. Hence, to assess the classification capability of the models that are built using the factors, we use the confusion matrix that is shown in Table 9. From the confusion matrix we compute the precision (P), recall (R), and F-measure (F).

Table 9: Confusion matrix (TP=True positive, TN=True negative, FP=False positive, FN=False negative, IRN= in the release notes, NIRN= not in the release notes).

		Actual	
		IRN	**NIRN**
Classified	**IRN**	TP	FP
	NIRN	FN	TN

Precision (P): measures the correctness of a model in classifying issues. A classification is considered correct, if an issue classifed to be in the release notes is actually in the release notes. Precision is computed as the ratio of issues which are correctly classified to the total number of classified issues to be in the release notes $P = TP/(TP + FP)$. A classification model is considered precise if all issues that are classified to be in the release notes are actually in the release notes, i.e. if $P = 1$.

Recall (R): measures the completeness of a model. A model is considered complete, if all the issues which are actually in the release notes are classified to be in the release notes. Recall is computed as the ratio of the number of correctly classified issues (*i.e.*, classified issues which are in the release notes) to the total number of issues which are actually in release notes, $R = TP/(TP + FN)$.

F-measure (F): is the harmonic mean of precision and recall ($F = (2 * P * R)/(P + R)$). F-measure is used to combine the inversely related precision and recall values into a single value. Combining the two inversely related values into one simplifies the comparison of the models that are used.

Precision, recall, and F-measure are computed for issues that are classified as either to be or not to be in release notes. In addition, we computed the weighted values of the two types of classifications, *i.e.*, IRN and NIRN, for each model. The weighted values are computed using the proportion of instances in each class (IRN and NIRN) as a weight. A high precision and recall value of the classification models indicates that release notes writers could identify most issues that need to be included in the release notes with minimal effort.

To evaluate the degree of discrimination achieved by each model, we also computed the ROC (Receiver Operating Characteristics) area. The ROC area is the area below the curve plotted using the fraction of true positive rate ($TP/(TP + FP)$) versus false positive rate ($FP/(FP+TN)$). The value of ROC area is between 0 and 1. An area greater than 0.5 shows that the classification model outperforms random guessing.

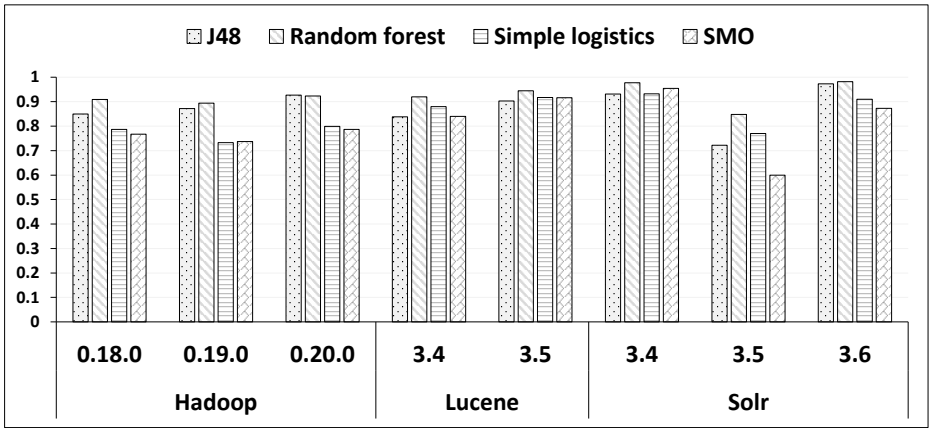


Fig. 4: Weighted F-measures.

4.2 Results and discussion

Automatically suggesting issues that should be listed in release notes has a precision of 84% and a recall of 90%, and almost all models have an F-measure of at least 72% on average. Table 10 shows the precision, recall, and F-measure achieved while using the classification models built using the four machine learning techniques. The lowest F-measure, 58%, is obtained for Solr 3.5 while using SMO to build the model. The F-measures and ROC area for Solr 3.5 are always lower than the F-measures and ROC areas of the others when using the J48, random forest, and SMO classification model. For simple logistics, however, the F-measure for Solr 3.5 is higher than Hadoop 0.18.0, 0.19.0, and 0.20.0. Looking closely into the datasets, we observe that Solr 3.5 has the smallest number of addressed issues. Hence, the low F-measure for Solr 3.5 in the three models, could be due to the small number of addressed issues in the release.

High precision and recall values are observed for both IRN and NIRN classifications. We also computed the precision and recall for the classification of issues that are not listed in the release notes (see Table 10). The results show that all models classfied issues which are not listed in release notes with average precision and recall of 90% and 83%, respectively. The high precision and recall values for both types of classifications, *i.e.*, IRN and NIRN, show that the models are not biased. Similar to the obtained results for issues that are listed in release notes, random forest gives the highest F-measure in all studied release notes except for Hadoop 0.20.0. The weighted F-measures of both types of classifications, listed in the release notes and not listed in the release notes, is also consistent to the individual F-measures (see Figure 4).

All classification models have a ROC area greater than the area achieved by random guessing. The SMO model built for Solr 3.5 release notes has the lowest ROC area, *i.e.*, 0.6. For the other models and release notes, the area is between 0.73 and 1. The area range shows that the models are better than a random guessing in discriminating issues that are listed in release notes.

Table 10: Results of classification within the same version (NIRN=not in the release notes, IRN=in the release notes).

System	Version	Metric	J48			Random Forest			Simple logistics			SMO		
			NIRN	IRN	Weighted	NIRN	IRN	Weighted	NIRN	IRN	Weighted	NIRN	IRN	Weighted
Hadoop	0.18.0	Precision	.86	.84	.85	.95	.88	.91	.76	.81	.79	.73	.81	.77
		Recall	.82	.88	.85	.85	.96	.91	.79	.78	.79	.81	.73	.77
		F-Measure	.84	.86	.85	.90	.92	.91	.78	.80	.79	.77	.77	.77
		ROC Area		.87			.97			.86			.77	
	.19.0	Precision	.90	.85	.87	.93	.87	.90	.68	.80	.74	.70	.77	.74
		Recall	.82	.92	.87	.84	.95	.89	.81	.67	.73	.77	.71	.74
		F-Measure	.86	.88	.87	.88	.90	.89	.74	.73	.73	.73	.74	.74
		ROC Area		.89			.96			.80			.74	
	.2.0	Precision	.98	.88	.93	.98	.87	.93	.81	.78	.80	.79	.78	.79
		Recall	.89	.98	.93	.87	.98	.92	.81	.78	.80	.82	.75	.79
		F-Measure	.93	.93	.93	.92	.92	.92	.81	.78	.80	.81	.77	.79
		ROC Area		.93			.99			.88			.79	
Lucene	3.4	Precision	.90	.80	.85	.92	.92	.92	.88	.89	.88	.83	.85	.84
		Recall	.75	.92	.84	.92	.92	.92	.88	.89	.88	.83	.85	.84
		F-Measure	.82	.86	.84	.92	.92	.92	.88	.89	.88	.83	.85	.84
		ROC Area		.80			.97			.92			.84	
	3.5	Precision	.97	.84	.91	.97	.91	.95	.95	.88	.92	.91	.93	.92
		Recall	.85	.97	.90	.93	.97	.94	.90	.94	.92	.95	.88	.92
		F-Measure	.91	.90	.90	.95	.94	.95	.92	.91	.92	.93	.90	.92
		ROC Area		.90			.98			.95			.91	
Solr	3.4	Precision	1.00	.89	.94	.96	1.00	.98	.95	.92	.93	1.00	.92	.96
		Recall	.86	1.00	.93	1.00	.96	.98	.91	.96	.93	.91	1.00	.96
		F-Measure	.92	.94	.93	.98	.98	.98	.93	.94	.93	.95	.96	.95
		ROC Area		.89			.98			.97			.95	
	3.5	Precision	.81	.67	.74	1.00	.76	.89	.93	.69	.82	.62	.58	.60
		Recall	.62	.84	.73	.71	1.00	.85	.62	.95	.78	.62	.58	.60
		F-Measure	.70	.74	.72	.83	.86	.85	.74	.80	.77	.62	.58	.60
		ROC Area		.75			.95			.79			.60	
	3.6	Precision	1.00	.95	.98	1.00	.97	.98	1.00	.86	.92	1.00	.81	.90
		Recall	.94	1.00	.97	.96	1.00	.98	.81	1.00	.91	.74	1.00	.88
		F-Measure	.97	.98	.97	.98	.98	.98	.90	.92	.91	.85	.90	.87
		ROC Area		.97			1.00			.92			.87	
Median (± standard deviation)		Precision	.94 (±.07)	.85 (±.08)	.89 (±.07)	.96 (±.03)	.90 (±.07)	.93 (±.04)	.90 (±.11)	.83 (±.07)	.85 (±.07)	.81 (±.14)	.81 (±.11)	.81 (±.12)
		Recall	.84 (±.10)	.95 (±.06)	.89 (±.08)	.89 (±.09)	.97 (±.03)	.92 (±.04)	.81 (±.09)	.91 (±.11)	.84 (±.08)	.81 (±.10)	.80 (±.15)	.81 (±.11)
		F-Measure	.88(±.08)	.89 (±.07)	.89 (±.08)	.92 (±.05)	.92 (±.04)	.92 (±.04)	.84 (±.08)	.84 (±.08)	.84 (±.08)	.82 (±.11)	.81 (±.12)	.81(±.11)
		ROC Area		.89(±.07)			.97 (±.02)			.90 (±.07)			.81 (±.11)	

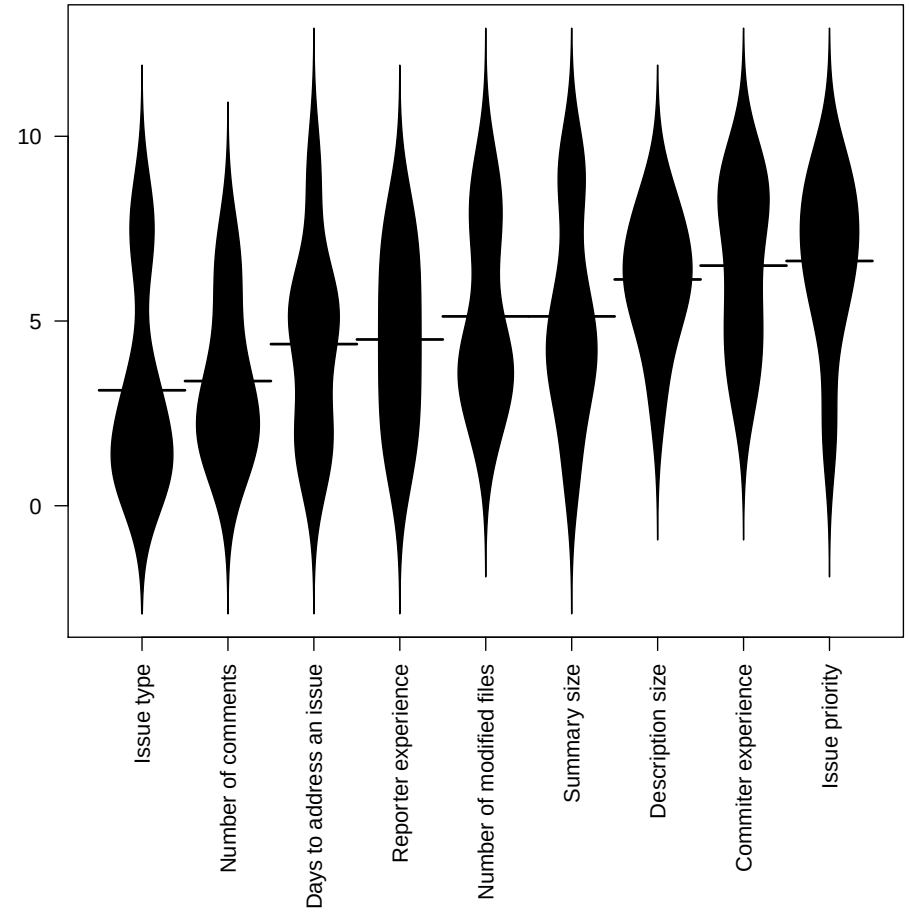


Fig. 5: Ranked list of important variables distribution. Issue type has the highest rank, while issue pirority has the lowest rank.

Random forest has the best performance among the four used machine learning techniques. The highest F-measures in classifiying issues to be listed in release notes is obtained by random forest, while most of the lowest scores are obtained by SMO. The F-measures for random forest range between 86% and 98%. The closest result to random forest is obtained using J48. The result shows that not all machine learning techniques are equally good in identifying and learning important characteristics from the factors. The result is also consistent with ROC area measures which are above 0.95.

Issue type, days to address an issue, and number of comments are ranked in the top five important factors in 75% of the release notes. The Beanplot in Figure 5 shows the distribution of the rank for each factor between 1 and 9 (with 1 being the most important). Beanplots are boxplots which also show the distribution of the data using the vertical curves. The horizontal black line indicates the median rank. To identify the factors which play an important role

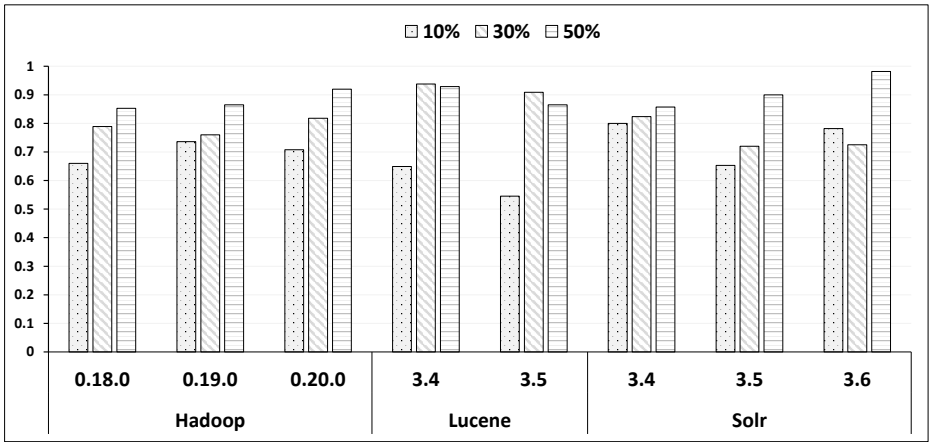


Fig. 6: F-measures of issues classified to be in the release notes while using only 10%, 30%, and 50% of the data for training using random forest.

in the classification, we computed conditional variable importance using *varimp*. *varimp* computes conditional variable importance for the $cforest$ implementation of random forest in R [4]. The implementation of $cforest$ is the same as the original implementation of random forest [15], but differs on the base learners used and the aggregation scheme. The base learners in $cforest$ are conditional inference trees [23] while the aggregation scheme averages observation weights extracted from each of the trees built [24]. The *number of modified files* and *reporter experience* are also identified as important factors in more than half of the release notes. These results are consistent with what we found in Section 3. In both evaluations *issue type*, *days to address an issue*, *number of comments*, and *reporter experience* are identified to explain issues that are listed in release notes in the majority of the release notes.

From the F-measure and ROC area values obtained for the majority of the release notes, we can conclude that the factors are able to build models which suggest issues that should be listed in release notes with reasonable accuracy. Based on our results, we recommend the use of all the factors except issue priority, summary size, and committer experience. The use of a random forest classifier leads to the best performing models.

5 Different scenarios for training and building models

The training dataset that is used in building classification models can be created by: (a) having the user label some issues for a release, (b) using the previous release notes for the same software system, and (c) using prior releases for the current software system and the rest of the studied software systems. Similar scenarios are used in building different classification models in various studies. Kamei *et al.* and Abebe *et al.* used infomation within the same version and previous releases to train and build a classification model [6, 25]. Bell *et al.* and D'Ambros *et al.* also used historical data to train and build a classification model [12, 17]. Using cross-project

data to train and build a classification model is investigated in several studies [22, 44, 45]. Below we describe the obtained results while using the classification models built following the three types of scenarios.

Table 11: Examples of correctly and incorrectly classified addressed issues which are listed in release notes.

Training scenario	**Model**	**System**	**IssueID**	**Correctly classified?**
10% of data for training	Random Forest	Hadoop 0.18.0	HADOOP-3336	No
			HADOOP-3337	Yes
Previous version	Simple Logistics	Lucene 3.5	LUCENE-2215	Yes
			LUCENE-2564	No
Previous versions and the rest of the studied systems	SMO	Solr 3.5	SOLR-1023	No
			SOLR-1926	Yes

5.1 Training using some labeled issues

Small training data could be used to train the model and classify issues to be listed in the release notes with satisfactory accuracy. Figure 6 shows that the F-measures obtained while using random forest to build a model using 10%, 30%, and 50% of each release notes data collected for training. The data used in this scenario is the same as what is used in the previous section. The results show that while using only 10% of the release notes' data for training, the model correctly classifies more than 60% of the issues in seven of the eight studied release notes. The first two rows in Table 11 show examples of incorrectly and correctly classified issues. *HADOOP-3336*, for example, is actually listed in the release notes but the issue is classified by the model as not listed in the release notes. Looking at the classification models built using 30% and 50% of the release notes data collected for training, *HADOOP-3336* is classified correctly. Hence, for some issues to be correctly classified, we need access to some additional information. *HADOOP-3337*, on the other hand, is correctly classified by all classification models irrespective of the size of the training data. The result indicates that the model can be trained with a minimal effort and that it gives satisfactory results. Hence, in a real setting, a release notes writer needs to manually classify a small number of issues that should be listed in the release notes for training and use the model to suggest the remaining issues.

To further understand why some issues are correctly classified while others are not, we conducted a manual analysis. In our manual analysis, we observed that for some issues the reporter name is the same as the name of the person who is assigned to address the issue. In an issue, the same reporter and assignee name could indicate that the issue is internal and release notes writers might not need to have the issue reported in the release notes. To see if our conjuncture helps to improve the classification, we added a boolean factor that tells if the issue reporter name is the same as the assignee name and we built our models. We refer to such

Table 12: Median F-measure of 10 random run classification results.

System	Version	Without boolean factor			With boolean factor		
		10%	30%	50%	10%	30%	50%
Hadoop	**0.18.0**	0.76	0.82	0.87	0.75	0.81	0.87
	0.19.0	0.70	0.83	0.86	0.71	0.81	0.87
	0.20.0	0.69	0.83	0.87	0.69	0.83	0.88
Lucene	**3.4**	0.60	0.88	0.90	0.67	0.92	0.95
	3.5	0.60	0.82	0.86	0.63	0.83	0.89
Solr	**3.4**	0.70	0.89	0.93	0.69	0.82	0.91
	3.5	0.63	0.77	0.82	0.57	0.79	0.86
	3.6	0.82	0.89	0.94	0.80	0.90	0.93

models as *models with boolean factor*. The classification results of *models with boolean factor* are then compared with results of models built with only the factors that are listed in Table 6. We refer to the models built with only the factors that are listed in Table 6 as *models without boolean factor*. To make sure that the results are consistent, we performed 10 runs using 10%, 30%, and 50% of each release notes data that is collected for training (30 training sets with two runs for each training set (with and without boolean factor)). For each run, the training set is selected randomly.

Overall, the results show that having the boolean factor in addition to the factors listed in Table 6 did not help improve the model performance. Table 12 shows the median of the F-measures for each training set. We also found that the median conditional variable importance rank of the boolean factor is 7 out of 10. The conditional variable importance is computed using *varimp*. To see if there is statistically significant difference between the results of the 10 runs for each training set and versions, we performed a paired, two sided Wilcoxon signed rank test. The p-value is less than 0.05 for only Solr 3.4 while using 30% of the data for training. For all other versions and training sets the p-value is greater than or equal to 0.05. Hence, we did not reject the null hypothesis, *i.e.*, there is no statistical difference between the performance of the *model with boolean factor* and *model without boolean factor*.

5.2 Training using a previous release

On average, the classification models suggested issues to be listed in the next version of a release notes with a precision of 30% and recall of 67%. We conducted an experiment to see if a release notes writer can use the data from the previous release notes in order to train a model and use it in the current version. For example, we used Hadoop 0.18.0 data to train a model and classify the issues to be listed in Hadoop 0.19.0. The results are shown in Table 13. The F-measures obtained are low due to the low precision values. The low precision values indicate that the models incorrectly suggested a high number of issues which are not listed in release notes. However, all ROC values are above 0.5 which indicates that the models are better than a random guess in discriminating issues that are listed in the release notes. The result indicates that release notes writers who use data from previous release notes to train their model can get most of the issues to be

Table 13: Results of classification on next version release notes.

Classification on	Model	Precision	Recall	F-Measure	ROC Area
Hadoop 0.19.0	**J48**	.36	.76	.49	.72
	Random Forest	.39	.73	.51	.77
	Simple Logistics	.38	.84	. 52	.78
	SMO	38	.79	.51	.70
Hadoop 0.20.0	**J48**	.27	.46	.34	.58
	Random Forest	.33	.36	.34	.72
	Simple Logistics	.33	.62	.43	.78
	SMO	38	.56	.45	.70
Lucene 3.5	**J48**	.24	.78	.37	.72
	Random Forest	.28	.56	.37	.57
	Simple Logistics	.39	.56	.46	.79
	SMO	.46	.56	.50	.73
Solr 3.5	**J48**	.23	.60	.33	.66
	Random Forest	.29	.80	.42	.79
	Simple Logistics	.25	1.00	.40	.85
	SMO	.24	1.00	.39	.77
Solr 3.6	**J48**	.23	.60	.33	.66
	Random Forest	.13	.29	.18	.72
	Simple Logistics	.18	.71	.29	.71
	SMO	.20	.71	.31	.76
Median (± Standard deviation)	**J48**	.24 (±.05)	.60 (±.13)	.34 (±.07)	66 (±.06)
	Random Forest	.29 (±.09)	.56 (±.22)	.37 (±.12)	.72 (±.09)
	Simple Logistics	.33 (±.09)	.71 (±.18)	.43 (±.09)	.78 (±.05)
	SMO	.38 (±.11)	.71 (±.18)	.45 (±.08)	.73 (±.03)

listed however they need to remove a relatively high number of issues. Examples of correctly and incorrectly classified issues are shown in Table 11 (middle rows).

5.3 Training using prior releases and other software systems

The classification using prior releases and other software systems is conducted using: (a) global model, and (b) ensemble model. To build the global model, we used the datasets from previous releases and the rest of the studied software systems to build a model. We merged the datasets because our earlier observations and findings (see Sections 3.2 and 3.1) indicate high variance in the size of the data and the factors that influence each release. The model is then used to classify issues for the release notes of the current release.

Models built using different datasets are usually different because each dataset could have different characteristics. To exploit the diversity and see the impact on classification, we built ensemble model using majority voting (*i.e.*, an ensemble approach). To build the ensemble model, one model is built per dataset and the classification of the models is combined using majority voting. A model is built for each previous release of the current software system and the rest of the studied systems, and is used to classify the issues to be listed in the remaining release notes. An issue is to be listed in the release notes, if the issue is classified to be listed in the release notes by majority of the models. We used an R package, ROCR, to compute precision, recall, F-measure, and ROC area of the voting result [39].

Global models suggested issues to be listed in the release notes with an average precision of 33% and an average recall of 67%. Table 14 shows the results of the global models. The classification is conducted on the system and version that are indicated in Table 14, while the data from previous versions and other systems

in the study are used for training. Despite the low F-measures, the ROC values are above 0.5 which indicates that the models are better than random guessing. Similar to classification on the next version, global models have low precision. Hence, release notes writers who use across system dataset to construct one training dataset and build classification models need to conduct manual removal of a relatively high number of issues. The last two rows of Table 11 show examples of incorrectly and correctly classified issues.

Table 14: Results of classification models trained and built using prior releases and other software systems.

System	Version	Metric	J48	Random Forest	Simple Logistics	SMO
Hadoop	**0.18.0**	**Precision**	.68	.71	.53	.59
		Recall	.32	.26	.45	.46
		F-Measure	.44	.38	.48	.52
		ROC Area	.60	.72	.68	.68
	0.19.0	**Precision**	.40	.48	.42	.43
		Recall	.52	.62	.78	.76
		F-Measure	.45	.54	.54	.55
		ROC Area	.67	.75	.79	.73
	0.20.0	**Precision**	.24	.39	.38	.36
		Recall	.59	.41	.69	.72
		F-Measure	.34	.40	.49	.48
		ROC Area	.63	.69	.85	.75
Lucene	**3.4**	**Precision**	.20	.30	.24	.20
		Recall	.43	.43	.71	.57
		F-Measure	.27	.35	.36	.30
		ROC Area	.56	.64	.68	.60
	3.5	**Precision**	.24	.33	.36	.30
		Recall	.44	.67	.89	.89
		F-Measure	.31	.44	.52	.44
		ROC Area	.54	.79	.86	.79
Solr	**3.4**	**Precision**	.25	.36	.22	.36
		Recall	.67	.67	.67	.83
		F-Measure	.36	.47	.33	.50
		ROC Area	.66	.72	.72	.80
	3.5	**Precision**	.18	.27	.26	.22
		Recall	.60	.80	1.00	1.00
		F-Measure	.27	.40	.42	.36
		ROC Area	.58	.81	.82	.74
	3.6	**Precision**	.14	.16	.15	.14
		Recall	.86	.86	1.00	1.00
		F-Measure	.25	.27	.26	.24
		ROC Area	.71	.86	.87	.79
Median (± Standard deviation)		**Precision**	.24 (±.17)	.35 (±.16)	.31 (±.12)	.33 (±.14)
		Recall	.56 (±.16)	.64 (±.20)	.75 (±.19)	.80 (±.19)
		F-Measure	.32 (±.08)	.40 (±.08)	.45 (±.10)	.46 (±.11)
		ROC Area	.62 (±.06)	.73 (±.07)	.80 (±.08)	.75 (±.07)

Classifying issues to be listed in a release notes using ensemble models has an average precision of 38% and an average recall of 61%. Table 15 shows the results of the approach which uses majority voting to classify issues to be listed in the release notes. The results are similar to the classification by global models and classification on the next version. On average, all models classified issues to be listed in release notes with an F-measure below 44%. The models have high ROC area (0.70 on average) which indicates that the models are better than random guessing. The precision of the models, however, is relatively low which indicates

that release notes writers might need to conduct manual removal of a relatively high number of issues.

Table 15: Classification performance for majority vote ensemble models.

Classification on	Model	Precision	Recall	F-Measure	ROC Area
Hadoop 0.18.0	**J48**	.48	.35	.41	.61
	Random Forest	.81	.2	.32	.59
	Simple Logistics	.67	.22	.33	.59
	SMO	.71	.34	.46	.65
Hadoop 0.19.0	**J48**	.47	.62	.53	.71
	Random Forest	.62	.4	.49	.66
	Simple Logistics	.56	.52	.54	.7
	SMO	.54	.6	.57	.73
Hadoop 0.20.0	**J48**	.43	.51	.47	.7
	Random Forest	.59	.26	.36	.61
	Simple Logistics	.52	.41	.46	.67
	SMO	.36	.51	.43	.68
Lucene 3.4	**J48**	.27	.43	.33	.62
	Random Forest	.38	.43	.40	.66
	Simple Logistics	.25	.57	.35	.65
	SMO	.22	.57	.32	.62
Lucene 3.5	**J48**	.33	.89	.48	.82
	Random Forest	.46	.67	.55	.78
	Simple Logistics	.29	.56	.38	.68
	SMO	.42	.89	.57	.86
Solr 3.4	**J48**	.17	.50	.25	.55
	Random Forest	.31	.83	.45	.77
	Simple Logistics	.33	.67	.44	.73
	SMO	.33	.50	.40	.67
Solr 3.5	**J48**	.22	.80	.35	.70
	Random Forest	.31	.80	.44	.77
	Simple Logistics	.25	1.00	.40	.79
	SMO	.25	1.00	.40	.79
Solr 3.6	**J48**	.15	1.00	.26	.82
	Random Forest	.16	.71	.26	.73
	Simple Logistics	.18	.86	.29	.80
	SMO	.17	.86	.29	.79
Median (± Standard deviation)	**J48**	.30 (±.13)	.57 (±.23)	.38 (±.10)	.70 (±.10)
	Random Forest	.42 (±.21)	.55 (±.25)	.42 (±.09)	.70 (±.08)
	Simple Logistics	.31 (±.18)	.57 (±.25)	.39 (±.08)	.69 (±.07)
	SMO	.35 (±.18)	.59 (±.23)	.42 (±.10)	.71 (±.08)

6 Threats to validity

Some threats limit the validity of our findings. To comprehend the strengths and limitations of our empirical study, we now discuss the potential threats to the validity of our findings and how we control or mitigate them.

Construct validity: We defined some factors (see Table 6) that could explain the likelihood of an issue to be listed in release notes. However, it is quite possible that using different factors and/or different methods to compute values for each factor could yield different results. To mitigate the selection and measurement of factors construct validity threat, we used nine different factors to measure the

likelihood of an issue to be listed in release notes. In addition, we also used a basic normalization method to normalize the values for each factor (see Section 3.1). The information types that are identified in Section 2 (see Table 3) are identified manually, and hence are subject to bias. To mitigate the manual analysis threat, one of the authors manually categorized contents of release notes for ten systems (see Table 3), while another author verified all the results. When there was a disagreement between the identified types, both authors held a discussion and consulted third author to resolve the disagreement. We conjecture that helping developers to write release notes could save some of the development/maintenance effort. However, it is quite possible that the cost of release notes writing is lesser than the other development/maintenance task. A more detailed study of software development/maintenance effort is required to better understand the effort saved in automatic release notes generation. Nevertheless our study is the first study to ever shed some light into the rationale for including an issue in release notes. Our study provides developers with empirically-supported advice instead of simply relying on adhoc advice from on-line inquiries.

Internal validity: The software system size could directly impact our results in Section 3. For example, a large software systems may have more commits to fix an issue than a smaller software system. To mitigate the impact of software system size on results, for issue types and issue priorities, we calculated the percentages and for all other factors, we normalized the values of measures (see Section 3.1). Another internal threat to validity could be the parameters tuning for classification models. To mitigate this threat, we used default parameters of the machine learning techniques and explicitly stated the non-default parameters that we used. Using different optimized parameters could produce different results than the results achieved in this paper. Thus, more experiments are required to analyze the impact of different parameters on our classification models. In addition, the factors that we used might not be the most optimal factors. For example, we use the opening and closing dates of an issue to measure the days spent on issue. Developers spend adequate amount of time to understand an issue before assigning it to a specific developer. Thus, using an assignment date as a starting date to compute "days spend on an issue" could be less effective. Other factors that capture other aspects might be explored and their impact by future research. However, the default parameters and proposed factors provide satisfactory results in our empirical study. For example, adding a new boolean factor, that we observed during our manual analsysis, did not improve the performance of the classification models. The boolean factor tells if the issue reporter name is the same as the assignee name.

External validity: In our exploratory study, we only analyzed 85 release notes taken from 15 systems. To evaluate the identified information types, we used 10 randomly selected release notes that cover more than 65% of the systems we analyzed. The selection of the studied software systems could bias the results of our study. However, the studied software systems vary in nature, *e.g.*, software system size, number of issues, commit logs, size of release notes. Thus, the selection criteria mitigates this threat. In addition, the approach used to build the classification models is applicable to any other system. However, we cannot claim that the same results would be achieved with other systems. Different systems with different values for the factor, *e.g.*, comment size, number of modified files for an

issue etc., may lead to different results. The collected values for the factors of the eight release notes are also different. Thus, the variation in our dataset reduces the threat to external validity of our empirical study. We only identified six information types of the studied software systems. There could be more information types than the ones that we identified in this paper. To the best of our knowledge, this paper is a first attempt to understand release notes. More studies could build on this paper to generalize the results.

We do note that based on our case study results (*i.e.*, within project models have the best performance), it appears that the decision of including issues varies between projects and even within releases. Hence, in lieu of seeking a global model that supports developers, it is probably more valuable to support developer by doing within release suggestions (*i.e.*, given a small set of issues that are marked for inclusion, we would provide the release notes writer with the other remaining issue instead of her/him having to manually go through all the issues that were integrated in that particular release).

7 Related work

Due to the rapidly changing user requirements and competition, many of the companies are moving towards shorter release cycles [27]. The shorter releases of a software system could leave more and rapid software trails [20], *e.g.*, logs, documentation, and release notes. Thus, keeping track and managing all of the software trails becomes an effort and a resource intensive task. The software repositories, *e.g.*, issue repositories, commit logs, and mailing lists, are used to keep track of software trails. Thus, over time, software repositories become an important asset for software developers to monitor the software trails. Several studies have been conducted to characterize software system trails, extract information from them, and automate their generation [9,29,37,38,43].

Release notes are one of the important software trails. Developers use release notes to communicate with their software users about new changes/updates in the software system. Developers could extract information from software repositories to summarize the changes made in the new release, *i.e.*, release notes. In this paper, we also used some software trails extracted from issue repositories and commit logs to identify issues listed in release notes.

Yu [43] conducted a study using release notes and software change logs to understand the evolution and maintenance of a software system. To support understanding, Yu proposed a keyword based mining technique to extract keywords from release notes and change logs. The results of the study show that the majority of maintenance and evolution activities are completed in one release. In addition, Yu observed that the number of new maintenance and evolution activities are linearly correlated with the number of recurring maintenance and evolution activities such as updating a feature. Maalej and Happel [29] studied work descriptions which are informal texts produced by developers. They, in particular, investigated personal work notes, commit messages and code comments, and identified information entities and granularities used in these descriptions. This work has a similar objective to our work, identifying general characteristics to support automation, but differs on the type of data investigated and results obtained. They studied work descriptions while we studied software release notes. The result of their study shows that

in different work descriptions there are many similarities in the way vocabularies are used, most frequent terms, and categories of described works. Maalej and Happel also observed that there are some recurrent description patterns. In our study, we identified six different information types contained in release notes and proposed an approach to automatically populate one of the information types, *i.e.*, addressed issues.

To the best of our knowledge, this paper is the first attempt towards understanding release notes, their characteristics, and automated recommendation of the contents of release notes.

8 Conclusions

Release notes are an important source of information for users as well as developers of a software system to get to know the new changes/updates of the software system. Release notes writers provide important information about a software system update in release notes. No specific guidelines for release notes writers on how to write release notes. The lack of guidelines causes variations in release notes contents of different software systems. Several people ask for such guidelines over question and answering websites. However, responders to such questions only answer based on their personal experience.

In this paper, we identify and summarize six information types that are contained in release notes as a first step towards providing more concrete guidelines for release notes writers. We find that release notes writers do not follow a consistent pattern across the software systems to write release notes. Release notes writers use different styles to list the addressed issues of a software system in the release notes. In addition, information types that are listed in release notes vary from system to system. The only common information types in all the release notes are the addressed issues and title (see Table 3). To help release notes writers, we proposed a machine learning approach to automatically suggest important addressed issues to be listed in release notes.

To the best of our knowledge, this is the first study towards analyzing and understanding the nature of release notes. Additional empirical studies are needed to generalize our results and to better understand release notes.

Future studies should investigate the evolution of release notes. Such an evolution study will help us better understand whether developers produce more precise and informative release notes over time or they follow the same strategy. Moreover, additional studies of more diverse software systems would help provide more concrete guidelines to developers for writing release notes. A qualitative user study is needed to understand the perspectives of developers and users regarding release notes and what important information types should appear in release notes.

References

1. Good practices of writing release notes, http://programmers.stackexchange.com/questions/167578/good-practices-of-writing-release-notes.
2. How should release notes be written, http://stackoverflow.com/questions/638423/how-should-release-notes-be-written.
3. A new swebok guide, http://www.computer.org/portal/web/swebok/.

4. R project, http://www.r-project.org/.
5. Release notes, http://en.wikipedia.org/wiki/release_notes.
6. Surafel Lemma Abebe, Venera Arnaoudova, Paolo Tonella, Giuliano Antoniol, and Yann-Gael Gueheneuc. Can lexicon bad smells improve fault prediction? In *Proceedings of the 2012 19th Working Conference on Reverse Engineering*, WCRE '12, pages 235–244, Washington, DC, USA, 2012. IEEE Computer Society.
7. Alain Abran, Pierre Bourque, Robert Dupuis, James W. Moore, and Leonard L. Tripp. *Guide to the Software Engineering Body of Knowledge - SWEBOK*. IEEE Press, Piscataway, NJ, USA, 2004 version edition, 2004.
8. Syed Nadeem Ahsan, Javed Ferzund, and Franz Wotawa. Program file bug fix effort estimation using machine learning methods for oss. In *SEKE*, pages 129–134, 2009.
9. A. Alali, H. Kagdi, and J.I. Maletic. What's a typical commit? a characterization of open source software repositories. In *16th IEEE International Conference on Program Comprehension, 2008. ICPC 2008.*, pages 182–191, 2008.
10. Adrian Bachmann, Christian Bird, Foyzur Rahman, Premkumar Devanbu, and Abraham Bernstein. The missing links: bugs and bug-fix commits. In *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering*, pages 97–106. ACM, 2010.
11. Ricardo Barandela, José Salvador Sánchez, Vicente García, and E. Rangel. Strategies for learning in class imbalance problems. *Pattern Recognition*, 36(3):849–851, 2003.
12. Robert M. Bell, Thomas J. Ostrand, and Elaine J. Weyuker. Does measuring code change improve fault prediction? In *Proceedings of the 7th International Conference on Predictive Models in Software Engineering*, Promise '11, pages 2:1–2:8, New York, NY, USA, 2011. ACM.
13. Ray Bernard, PSP, and CHS-III. Convergence q&a: The answer is in black and white. `http://goo.gl/VMZG2k`, 2012. Accessed September, 2014.
14. Christian Bird, David Pattison, Raissa D'Souza, Vladimir Filkov, and Premkumar Devanbu. Latent social structure in open source projects. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, pages 24–35. ACM, 2008.
15. Leo Breiman and E. Schapire. Random forests. In *Machine Learning*, pages 5–32, 2001.
16. J. W. Creswell. *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches*. Sage Publications Ltd., 3 edition, 2008.
17. M. D'Ambros, M. Lanza, and R. Robbes. An extensive comparison of bug prediction approaches. In *Proceedings of the 7th IEEE Working Conference on Mining Software Repositories (MSR)*, MSR '10, pages 31–41, May 2010.
18. Nguyen Duc Anh, Daniela S Cruzes, Reidar Conradi, and Claudia Ayala. Empirical validation of human factors in predicting issue lead time in open source projects. In *Proceedings of the 7th International Conference on Predictive Models in Software Engineering*, page 13. ACM, 2011.
19. Jon Eyolfson, Lin Tan, and Patrick Lam. Do time of day and developer experience affect commit bugginess? In *Proceedings of the 8th Working Conference on Mining Software Repositories*, pages 153–162. ACM, 2011.
20. Daniel German. Using software trails to rebuild the evolution of software. *Journal of Software Maintenance and Evolution: Research and Practice*, 2004.
21. Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: an update. *SIGKDD Exploration Newsletter*, 11(1):10–18, November 2009.
22. Zhimin He, Fengdi Shu, Ye Yang, Mingshu Li, and Qing Wang. An investigation on the feasibility of cross-project defect prediction. *Automated Software Engg.*, 19(2):167–199, 2012.
23. Torsten Hothorn, Kurt Hornik, and Achim Zeileis. Unbiased recursive partitioning: A conditional inference framework. *Journal of Computational and Graphical Statistics*, 15(3):651–674, 2006.
24. Torsten Hothorn, Berthold Lausen, Axel Benner, and Martin Radespiel-Tröger. Bagging survival trees. *Statistics in Medicine*, 23(1):77–91, Jan 2004.
25. Yasutaka Kamei, Shinsuke Matsumoto, Akito Monden, Ken-ichi Matsumoto, Bram Adams, and Ahmed E. Hassan. Revisiting common bug prediction findings using effort-aware models. In *Proceedings of the 2010 IEEE International Conference on Software Maintenance*, ICSM '10, pages 1–10, Washington, DC, USA, 2010. IEEE Computer Society.

26. S. S. Keerthi, S. K. Shevade, C. Bhattacharyya, and K. R. K. Murthy. Improvements to platt's smo algorithm for svm classifier design. *Neural Comput.*, 13(3):637–649, 2001.
27. Foutse Khomh, Tejinder Dhaliwal, Ying Zou, and Bram Adams. Do faster releases improve software quality? an empirical case study of mozilla firefox. In *Mining Software Repositories (MSR), 2012 9th IEEE Working Conference on*, pages 179–188. IEEE, 2012.
28. Kaiping Liu, Hee Beng Kuan Tan, and Hongyu Zhang. Has this bug been reported? In *Reverse Engineering (WCRE), 2013 20th Working Conference on*, pages 82–91. IEEE, 2013.
29. W. Maalej and H.-J. Happel. Can development work describe itself? In *7th IEEE Working Conference on Mining Software Repositories*, pages 191–200, 2010.
30. Patricia Yancey Martin and Barry A. Turner. Grounded theory and organizational research. *The Journal of Applied Behavioral Science*, 22(2):141–157, 1986.
31. Thilo Mende and Rainer Koschke. Effort-aware defect prediction models. In *Software Maintenance and Reengineering (CSMR), 2010 14th European Conference on*, pages 107–116. IEEE, 2010.
32. Audris Mockus and James D. Herbsleb. Expertise browser: A quantitative approach to identifying expertise. In *Proceedings of the 24th International Conference on Software Engineering*, ICSE '02, pages 503–512, New York, NY, USA, 2002. ACM.
33. LaTonya Pearson. The benefit of software release notes and why your company should use them. `http://goo.gl/lsCKtw`, 2013. Accessed September, 2014.
34. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, San Mateo, CA, 1993.
35. Foyzur Rahman and Premkumar Devanbu. Ownership, experience and defects: a fine-grained study of authorship. In *Proceedings of the 33rd International Conference on Software Engineering*, pages 491–500. ACM, 2011.
36. Jelber Sayyad Shirabad. *Supporting software maintenance by mining software update records*. PhD thesis, Ottawa, Ont., Canada, Canada, 2003. AAINQ79317.
37. E. Shihab, A. Ihara, Y. Kamei, W.M. Ibrahim, M. Ohira, B. Adams, A.E. Hassan, and K.-i. Matsumoto. Predicting re-opened bugs: A case study on the eclipse project. In *17th Working Conference on Reverse Engineering (WCRE)*, pages 249–258, 2010.
38. Emad Shihab, Akinori Ihara, Yasutaka Kamei, Walid M. Ibrahim, Masao Ohira, Bram Adams, Ahmed E. Hassan, and Ken ichi Matsumoto. Studying re-opened bugs in open source software. *Empirical Software Engineering*, 18(5):1005–1042, 2013.
39. Tobias Sing, Oliver Sander, Niko Beerenwinkel, and Thomas Lengauer. Rocr: visualizing classifier performance in r. *Bioinformatics*, 21(20):3940–3941, 2005.
40. Marc Sumner, Eibe Frank, and Mark Hall. Speeding up logistic model tree induction. In *Proceedings of the 9th European conference on Principles and Practice of Knowledge Discovery in Databases*, PKDD'05, pages 675–683, Berlin, Heidelberg, 2005. Springer-Verlag.
41. Yuan Tian, David Lo, and Chengnian Sun. Drone: Predicting priority of reported bugs by multi-factor analysis. In *Software Maintenance (ICSM), 2013 29th IEEE International Conference on*, pages 200–209. IEEE, 2013.
42. Rongxin Wu, Hongyu Zhang, Sunghun Kim, and Shing-Chi Cheung. Relink: recovering links between bugs and changes. In *Proceedings of the nineteen ACM SIGSOFT international symposium on Foundations of software engineering*, pages 15–25. ACM, 2011.
43. Liguo Yu. Mining change logs and release notes to understand software maintenance and evolution. *CLEI Electronic Journal*, 12(2), 2009.
44. Feng Zhang, Audris Mockus, Iman Keivanloo, and Ying Zou. Towards building a universal defect prediction model. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 182–191, New York, NY, USA, 2014. ACM.
45. Thomas Zimmermann, Nachiappan Nagappan, Harald Gall, Emanuel Giger, and Brendan Murphy. Cross-project defect prediction: A large scale experiment on data vs. domain vs. process. In *Proceedings of the the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, ESEC/FSE '09, pages 91–100, New York, NY, USA, 2009. ACM.