ARENA: An Approach for the Automated Generation of Release Notes

Laura Moreno, *Member, IEEE,* Gabriele Bavota, *Member, IEEE,* Massimiliano Di Penta, *Member, IEEE,* Rocco Oliveto, *Member, IEEE,* Andrian Marcus, *Member, IEEE,* Gerardo Canfora

Abstract—Release notes document corrections, enhancements, and, in general, changes that were implemented in a new release of a software project. They are usually created manually and may include hundreds of different items, such as descriptions of new features, bug fixes, structural changes, new or deprecated APIs, and changes to software licenses. Thus, producing them can be a time-consuming and daunting task. This paper describes ARENA (**A**utomatic **RE**lease **N**otes gener**A**tor), an approach for the automatic generation of release notes. ARENA extracts changes from the source code, summarizes them, and integrates them with information from versioning systems and issue trackers. ARENA was designed based on the manual analysis of 990 existing release notes. In order to evaluate the quality of the release notes automatically generated by ARENA, we performed four empirical studies involving a total of 56 participants (48 professional developers and 8 students). The obtained results indicate that the generated release notes are very good approximations of the ones manually produced by developers and often include important information that is missing in the manually created release notes.

Index Terms—Release notes, Software documentation, Software evolution

1 Introduction

RELEASE notes summarize the main changes that occurred in a software system since its previous release, such as, the addition of new features, bug fixes, changes to licenses under which the project is released, and, especially when the software is a library used by others, relevant changes at code level. Different stakeholders might benefit from release notes. *Developers*, for example, use them to learn what has changed in the code, which features have been implemented and which features are left for future releases, which bugs have been fixed and which ones are still open, whether or not the latest release introduced new legal constraints, etc. Similarly, *integrators*, who are using a library in their code, use the library release notes to decide whether or not such a library should be upgraded to a new release. Finally, *end users* read the release notes to decide whether it would be worthwhile to install a new release of a software (*e.g.,* a mobile app).

In general, release notes are produced manually. According to a survey we conducted among open-source and professional developers (see Section 4.2), creating a release note by hand is a difficult and effort-prone activity that can take as much as eight hours. Some issue trackers partially support this task by generating simplified release notes (*e.g.,* the *Atlassian OnDemand release note generator*[1]), yet such notes are limited to list closed issues that developers have manually associated with a release.

This paper proposes ARENA (**A**utomatic **RE**lease **N**otes gener**A**tor), an automated approach for the generation of release notes. ARENA identifies changes occurred in the commits performed between two releases of a software project, such as, structural changes to the code, upgrades of external libraries used by the project, and changes in the licenses. Then, ARENA summarizes the code changes through an approach derived from code summarization [28], [38]. These changes are linked to information that ARENA extracts from commit notes and issue trackers, which is used to describe fixed bugs, new features, and open bugs related to the previous release. Finally, the release note is organized into categories and presented as a hierarchical and interactive HTML document, where details on each item can be expanded or collapsed, as needed.

Currently, there is no industry-wide standard on the content and format of release notes, hence software companies adopt independent release note guidelines based on their own information and technical needs. For this reason, ARENA has been designed based on the *manual analysis of 990 project release notes* identifying what elements they typically contain. Also, it is important to point out that the release notes generated by ARENA include information that is *useful mainly to developers and integrators*, rather than to end-users.

From an engineering standpoint, ARENA leverages existing approaches for code summarization and for linking code changes to issues; yet, ARENA is novel and unique for two reasons: (i) it generates summaries and descriptions

- *L. Moreno and A. Marcus are with the University of Texas at Dallas, Richardson, TX, USA.*
  E-mail: lmorenoc, amarcus@utdallas.edu
- *G. Bavota is with the Free University of Bozen-Bolzano, Bolzano, Italy.*
  E-mail: gabriele.bavota@unibz.it
- *M. Di Penta and G. Canfora are with the University of Sannio, Benevento, Italy.*
  E-mail: dipenta, canfora@unisannio.it
- *R. Oliveto is with the University of Molise, Pesche (IS), Italy.*
  E-mail: rocco.oliveto@unimol.it

Manuscript received April 19, 2005; revised September 17, 2014.

1. http://tinyurl.com/atlassian-jira-rn. All URLs last verified on 06/10/2015.

of code changes at different levels of granularity than what was done in the past; and (ii) *for the first time*, it combines code analysis, summarization, and mining approaches together to address the problem of release note generation. To the best of our knowledge, no current tool or approach automatically generates release notes with the rich content of ARENA's release notes. Our conjecture is that such a rich content (describing, for example, the code that was changed to implement a bug fix or feature) of the release note, can be beneficial to developers during evolutionary tasks.

In order to evaluate ARENA, we performed four different empirical studies evaluating the release notes generated by ARENA from complementary points of view.

This paper extends our previous work on the automatic generation of release notes [29]. In particular, novel contributions of this paper are the following:

- We developed and publicly released ARENA[2], a tool implementing the approach for the automatic generation of release notes described in our previous work [29].
- Thanks to the tool availability, we performed a six-month in-field study, where a team of developers from a software company used ARENA to generate the release notes of a medical software system. This study allowed us to investigate the usefulness of ARENA and the quality of its release notes within an industrial setting, in which developers used it in their everyday routine for a substantial time period.

Replication package. A replication package is available online[3]. It includes: (i) the complete results of the survey; (ii) the code summarization templates used by ARENA; (iii) all the generated release notes; and (iv) the material and working data sets of the four evaluation studies.

Paper structure. Section 2 reports results of our initial survey to identify requirements for generating release notes. Section 3 introduces ARENA, while Section 4 presents the four evaluation studies and their results. The threats that could affect the validity of the results achieved are discussed in Section 5. Finally, Section 7 concludes the paper and outlines directions for future work, after a discussion of the related literature (Section 6).

2 What Do Release Notes Contain?

Since no industry-wide common standards exist for writing release notes, different communities produce release notes according to their own guidelines. In order to define the content and structure of the ARENA-generated release notes, we performed an exploratory study aimed at understanding the structure and content of existing release notes. Specifically, we manually inspected 990 release notes from 55 open source projects (see Table 1) to analyze and classify their content. Note that to improve the generalizability of the release notes generated by ARENA, we considered projects from different open-source communities. The analyzed notes belong to 608 releases of 41 open-source projects from the Apache ecosystem (*e.g.,* Ant, log4j, *etc.*), and 382 releases of 14 open-source projects developed by other communities (*e.g.,* jEdit, Selenium, Firefox, *etc.*).

2. https://seers.utdallas.edu/ARENA/
3. http://utdallas.edu/~lmorenoc/research/tse2015-arena/

TABLE 1
Systems used in the exploratory study.

Project type	Software project	Number of releases
Open-source (Apache community)	Abdera	8
	Accumulo	7
	Ant	16
	Cayenne	65
	Click	45
	Commons Attributes	2
	Commons BeanUtils	13
	Commons Betwixt	5
	Commons BSF	6
	Commons Chain	2
	Commons CLI	2
	Commons Codec	10
	Commons Collections	9
	Commons Compress	5
	Commons Configuration	7
	Commons Daemon	15
	Commons DBCP	4
	Commons DbUtils	4
	Commons Digester	15
	Commons Discovery	3
	Commons Email	3
	Derby	20
	Geronimo Eclipse	10
	Ivy	36
	JAMES	15
	Karaf	21
	Lenya	1
	log4j	60
	log4net	11
	Lucene Core	51
	Maven	28
	MINA	3
	MyFaces	3
	OpenJPA	12
	Pivot	8
	POI	22
	Qpid	7
	Regexp	5
	Santuario	10
	ServiceMix	21
	ZooKeeper	18
Subtotal # of projects	*41*	*608*
Open-source (other communities)	FindBugs	52
	Firefox	21
	Google Web Toolkit	41
	GTGE	3
	Hibernate	16
	ImageJ	63
	jEdit	31
	JHotDraw	4
	JUnit	10
	Netty	50
	ProGuard	31
	Selenium Java	34
	SLF4J	12
	Struts	14
Subtotal # of projects	*14*	*382*
Total # of projects	*55*	*990*

Release notes are usually presented as a list of items, each one describing some types of changes. Table 2 reports the 17 types of changes we identified as most frequently included in release notes, the number of release notes containing such information, and the corresponding percentage. Note that *Code Components* may refer to classes, methods, or instance variables. The raw data including detailed information for each one of the 990 analyzed release notes can be found in our replication package.

Bug fixes stand out as the most frequent item included in the release notes (in 888 release notes—90% of our sample). Typically, the information is reported as a simple bullet list containing for each fixed bug, its ID and a very short summary (often the bug's title as stored in the bug tracker).

TABLE 2
Contents of the 990 release notes.

Content Type	#Rel. Notes	%
Fixed Bugs	888	90%
New Features	455	46%
New Code Components	424	43%
Modified Code Components	397	40%
Modified Features	262	26%
Refactoring Operations	206	21%
Changes to Documentation	200	20%
Upgraded Library Dep.	157	16%
Deprecated Code Components	97	10%
Deleted Code Components	88	9%
Changes to Config. Files	84	8%
Changes to Code Components Visibility	72	7%
Changes to Test Suites	70	7%
Known Issues	64	6%
Replaced Code Components	47	5%
Architectural Changes	29	3%
Changes to Licenses	18	2%

For example, in the release note of Apache Lucene 4.0.0, the LUCENE-4310 bug fix is reported as follows:

LUCENE-4310: MappingCharFilter was failing to match input strings containing non-BMP Unicode characters.

Other frequently reported changes in release notes are new features (46%) and new code components (43%). These two types of changes are often found together, explaining what code components were added to implement the new features. Also, when available, the ID of the issue where developers discussed the implementation of the new feature is reported. An example of such an item reported in the Apache Lucene 4.0.0 release note is:

LUCENE-1888: Added the option to store payloads in the term vectors (IndexableFieldType.storeTermVectorPayloads()). Note that you must store term vector positions to store payloads.

Modified code components (*i.e.,* classes, methods, fields, parameters) are also frequently reported (40%). Note that we include here all cases where the release notes report that a code element has been changed, without specifying how. We do not include here deprecated code components or changes to code components' visibility that are classified separately (see Table 2).

Explanations of modified features are quite frequent in release notes (26%) and are generally accompanied by the code components that were added/modified/deleted to implement the feature change. An example from the release note of the Google Web Toolkit 2.3.0 (M1) is:

Updated GPE's UIBinder editor (i.e., class UIBinder of the Google Plug-in) to provide support for attribute auto-completion based on getter/setters in the owner type.

Refactoring operations are also included in release notes (21%), generally as simple statements specifying the impacted code components, *e.g.,* "*Refactored the WebDriverJs*"—from Selenium 2.20.0.

Changes in documentation are present in 20% of the analyzed release notes, although, more often than not, they are rather vaguely described with generic sentences like "*more complete documentation has been added*" or "*documentation improvements*".

We also found: 157 release notes (16%) reporting upgrades in the libraries used by the project (*e.g.,* "*The Deb Ant Task now works with Ant 1.7*"—from Jedit 4.3pre11); 97 (10%) reporting deprecated code components; and 88 (9%) including deleted code components (*e.g.,* "*Removed GWTShell, an obsolete way of starting dev mode*"—from Google Web Toolkit 2.5.1 (RC1)).

Other changes performed in the new release are less common in the analyzed release notes (see Table 2). We must note that rarely summarized types of changes are not necessarily less important than the frequently reported ones. It may be the case that some types of changes occur less frequently than others, hence they are reported less. For example, changes to licenses are generally rare and thus, only 18 release notes (2%) contain this information. We do not equate frequency with importance. Future work will answer the *importance* question separately.

Based on the results of this survey and on our assessment of what can be automatically extracted from available sources of information (*i.e.,* release archives, versioning systems, and issue trackers), we have formulated requirements for what ARENA should include in release notes:

1) a description of fixed bugs, new features, and improvement of existing features, complemented with a description of what was changed in the code (*i.e.,* classes, methods, fields, parameters, etc.);
2) a summary of other source code changes, including (i) description of code changes (*e.g.,* added, removed, and modified code elements, or changes to their visibility); (ii) deprecated methods/classes; and (iii) refactoring operations;
3) changes to the libraries used by the system;
4) changes to licenses and documentation; and
5) open issues.

In the subsequent sections we use the term "item" to refer to elements belonging to the above listed categories of changes and reported in the release note generated by ARENA (*e.g.,* a fixed bug described in the release note, a refactoring operation, *etc.*).

Note that in the current version of ARENA we did not consider changes to the high-level system architecture, because existing architecture recovery approaches (*e.g.,* [19], [22], [25]) usually require manual effort to produce usable results. Also, as it will be detailed in Section 3, we do not capture all changes performed to build files, only those for which it is possible to provide a specific rationale (*i.e.,* the change in dependencies).

3 ARENA Overview

In a nutshell, ARENA works as depicted in Fig. 1. The process to generate release notes for a release r_k is composed of four main steps.

During the **first** step (Section 3.1), the *Change Extractor* is used to capture changes performed between releases r_{k-1} and r_k in terms of: (i) fine-grained source code changes (*e.g.,* changes to methods visibility, deprecated classes, new classes, *etc.*); (ii) changes to libraries; (iii) changes to documentation; and (iv) changes to licenses. All these fine-grained source code changes are then linked to information extracted from the versioning system (*e.g.,* commit-id, date, *etc.*).

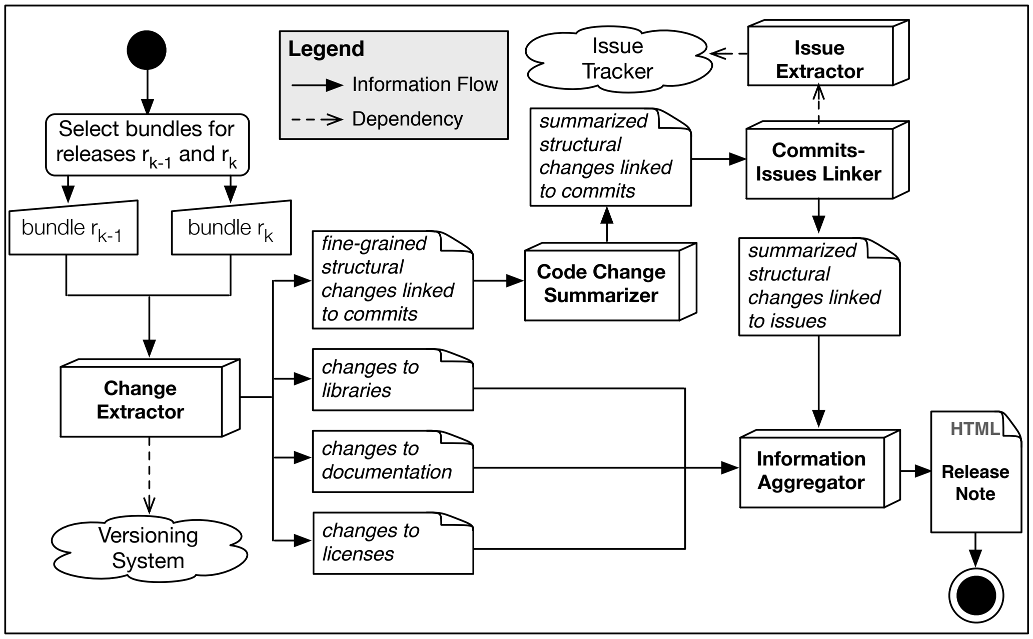


Fig. 1. Overview of the ARENA approach.

In the **second** step (Section 3.2), the *Code Change Summarizer* describes the fine-grained changes captured by the *Change Extractor*, with the goal of obtaining a higher-level view of what changed in the code. To this aim, the structural changes are hierarchically organized and prioritized to select the changes to be included in the summaries. An itemized, natural-language description is generated for the top changes. The summarized structural code changes are then linked to the extracted commit information from the previous step.

In the **third** step (Section 3.3), the *Commit-Issue Linker* uses the *Issue Extractor* to mine information (*e.g.,* fixed bugs, new features, improvements, *etc.*) from the issue tracker of the project to which r_k belongs. Each fixed bug, implemented new feature, and improvement is linked to the code changes performed between releases r_{k-1} and r_k. Therefore, in a generated release note, the summarized structural code changes represent *what* changed in the new release, while the information extracted from the issue tracker explains *why* the changes were performed.

Finally, in the **fourth** step (Section 3.4), the extracted information is provided as input to the *Information Aggregator*. This component is in charge of organizing the extracted information as a hierarchy and creating an HTML version of the release note for r_k, where each item can be expanded/collapsed to reveal/hide its details. An example of a release note generated by ARENA can be found at http://tinyurl.com/arena-lucene-4-0.

3.1 Extraction of Changes

ARENA extracts code changes, as well as changes to other entities, from two different sources: (i) the versioning system; and (ii) the source archives of the releases to be compared.

3.1.1 Identification of the Time Interval to be Analyzed

ARENA aims at identifying the subset of commits that pertains to the release for which the release note needs to be generated, say release r_k. Intuitively, the simplest way to analyze code changes occurring between the two releases r_{k-1} and r_k is to analyze the content of the source distribution archives of the two releases. Such analysis would enable the identification of structural changes, while it does not allow to properly link source code changes onto their rationale (to be mined from the issue tracker). In this sense, we opt for analyzing the commits in the versioning system, which contain each change performed to the system. To identify changes between the releases r_{k-1} and r_k, we consider all commits occurred—in the main trunk of the versioning system—starting from the r_{k-1} release date t_{k-1}, until the r_k release date t_k. Note that these dates could be approximations, as developers could start working on the release r_{k+1} even before t_k is issued, *i.e.,* changes committed before t_k could belong to the release r_{k+1} and not to the release r_k. It is worth noting that such a date approximation is needed only if dates are not specified by an end-user of ARENA. Indeed, in a real usage scenario (as explained in Section 3.5), a developer in charge of creating a release note using ARENA can simply provide the best time interval to analyze.

3.1.2 Analysis of Code Changes

Once the commits of interest are identified, ARENA's *Change Extractor* analyzes them using a code analyzer developed in the context of the *Markos* EU project[4]. The analyzer parses the source code using the *srcML* toolkit [11] and extracts a set of facts concerning the files that have been added, removed, and changed in each commit. Information

4. http://www.markosproject.eu

on the commits performed between the releases r_{k-1} and r_k is extracted from the versioning system (*e.g., git, svn*). Given the set of files in a commit, the following kinds of changes are identified:

- Files added, removed, and moved between packages;
- Classed added, removed, renamed, or moved between files.
- Methods added, removed, renamed, or moved between classes;
- Methods changed, *i.e.,* changes in the signature, visibility, source code, or set of thrown exceptions;
- Instance variables added, removed, and with visibility changes;
- Deprecated classes, methods, and instance variables.

Added and removed files. These changes are the easiest to identify, since they are explicitly reported in the versioning system log. For example, the commit:

```
F.java,<commit_id>,<date>,<author>,A,
<commit message>
```

indicates that the file `F.java` has been *added* to the repository, as suggested by the literal `A` in the fifth value of the commit. This value explicitly marks the type of change performed on the file object of the commit. Other possible literals for this value are `D`, indicating that the file has been deleted, and `M`, indicating its modification.

Moved files. A file `F.java` is considered as moved between two packages when a commit contains two operations related to `F.java`, one reporting `D` as type of change (*i.e.,* the file has been deleted from its previous folder) and the other one reporting `A` as type of change (*i.e.,* the file has been added to a new folder).

Added, removed, renamed, and moved classes. Added and removed classes are identified by comparing the content of each source code file in the commit before and after the commit was performed. Classes that have been renamed or moved between files are identified through a metric-based fingerprinting approach [3], [17], where a class is characterized by a set of metrics or *fingerprint* that allows to trace the class when its name or location change.

Added, removed, renamed, and moved methods. Similar to the previous case, added and removed methods are identified by comparing the content of each source code file before and after the analyzed commits. Likewise, renamed and moved methods are detected by using the fingerprinting-based approach.

Changed methods. For each method found in the modified files of the commit under analysis, a comparison between the before- and after-commit versions of the source code of the method is performed. Through this comparison, ARENA is able to identify: (i) changes to the method visibility (*e.g.,* a method converted from private to public); (ii) changes to the method signature (*e.g.,* parameters added/removed, changes to the exceptions thrown by the method, *etc.*); and (iii) changes to the method body.

Added, removed and changed instance variables. These changes are captured by comparing before- and after-commit versions of the instance variables of each class contained in modified code files.

Deprecated classes, methods, and instance variables. Deprecation of code elements is identified by detecting new `@Deprecated` Java tags on classes, methods, and instance variables in the modified code files of the commit under analysis.

All the previous changes are automatically linked to the commit in which they have been performed, by storing: (i) the commit-id in which the change occurred; (ii) the author of the change; (iii) the date of the change; and (iv) the commit message describing the change.

3.1.3 Analysis of Licensing Changes

In order to identify license changes, ARENA's *Change Extractor* analyzes the content of the source distribution of r_{k-1} and r_k, extracting all source files (*i.e.,*`.java`) and all text files (*i.e.,*`.txt`). Then, it uses the *Ninka* license classifier [16] to identify and classify licenses contained in such files. *Ninka* uses a pattern-matching based approach to identify statements characterizing the various licenses and, given any text file (including source code files) as input, it outputs the license type (*e.g.,* GPL) and its version (*e.g.,* 2.0) with a 95% precision. ARENA highlights a license change in r_k if: (i) a new file declaring a previously unused license is added in r_k, (ii) a file changes license type and/or version between r_{k-1} and r_k, or (iii) a file that declared a license l_i in r_{k-1} is deleted in r_k and no other files declaring l_i are present in r_k.

3.1.4 Identification of Changes in Documentation

This analysis is done on the release archives of r_{k-1} and r_k. Although release archives can contain any kind of documentation, we only focus on changes to documentation describing source code entities. ARENA identifies documentation changes using the approach described below:

1) Identify text files, *i.e.,*`.pdf`, `.txt`, `.rdf`, `.doc`, `.docx`, and `.html`, and extract the textual content from them using the *Apache Tika*[5] library.
2) If a text file (say doc_i) has been added in r_k, then verify whether it references code file names, class names, and method names. We use a pattern matching approach similar to the one proposed by Bacchelli *et al.* [4]. If such entities are found in a file, then check whether these files, classes, or methods have been added, removed, or changed in the source code, so that ARENA can generate an explanation of why the documentation was added, *e.g.,* if the added text file contains a reference to class C_j added in r_k, ARENA describes the change as *"The file doc_i has been added to the documentation to reflect the addition of class C_j"*.
3) If a text file (say doc_i) has been removed in r_k, then check if it references deleted methods, classes, or code

5. http://tika.apache.org

files. If that is the case, ARENA generates an explanation *"The file doc_i has been deleted from the documentation due to the removal of* <involved_code_components> *from the system"*.

4) If a text file has been modified between r_{k-1} and r_k, we use a similar approach as above, but we search for references to code entities only in the portions of the text file that were changed.

Note that ARENA is reporting changes in documentation artifacts in a slightly different way compared to what we observed in the release notes manually analyzed in the context of the study reported in Section 2. In particular, we observed that developers often describe changes to documentation at a very high level (*e.g.*, "more complete documentation has been added"). Instead, ARENA only looks for changes related to documentation files linked to code components. A simplistic approach, more in line with the findings of our exploratory study (Section 2), would have been to describe changes to all textual artifacts (*i.e.*, `.pdf`, `.txt`, `.rdf`, *etc.*) despite their linkage to code elements. However, as we tested this approach we observed that it resulted in the generation of some irrelevant information. This is because software repositories often contain textual files (*e.g.*, the project website, advertising material, list of contributors) that are not relevant when describing changes in a newly issued release. For this reason, we decided to only consider textual documents explicitly referring to code elements, since those are the ones likely representing relevant documentation artifacts.

3.1.5 Identification of Changes in the Used Libraries

ARENA's *Change Extractor* analyzes whether: (i) r_k uses new libraries compared to r_{k-1}; (ii) libraries are no longer required; and (iii) libraries have been upgraded to new releases. The analysis is performed in two steps:

1) Parse the files describing the inter-project dependencies. In Java projects, these are usually *property* files (`libraries.properties` or `deps.properties`) or Maven Project Object Model (POM) files (`pom.xml`). The information contained in such files allows ARENA to detect the libraries used in both releases r_k and r_{k-1}. Specifically, ARENA detects the name and used versions of each library, *e.g., ant* v. *1.7.1*.
2) Identify the *jar* files contained in the two release archives and—by means of regular expressions—extracting name and version from each jar file name, *e.g.,*`ant_1.7.1.jar` is mapped to library *ant* v. *1.7.1*.

With the list of libraries used in both releases, ARENA verifies and reports if: (i) new libraries have been added in r_k; (ii) libraries are no longer used in r_k; or (iii) new versions of libraries previously used in r_{k-1} are used in r_k.

3.1.6 Identification of Refactoring Operations

ARENA's *Change Extractor* also identifies refactoring operations performed between the releases r_{k-1} and r_k, by using two complementary sources of information:

1) Refactoring operations documented in the commit notes. Such operations are identified by matching regular expressions in commit notes—*e.g.,refact*, *renam*—as done in previous work [10], [36].
2) Class/method renaming and moving (those not already identified by means of their commits using the heuristic above). Such refactoring changes are identified by means of fingerprinting analysis.

In principle, ARENA could describe other kinds of refactoring operations by integrating refactoring identification tools like *RefFinder* [33]. For the time being, we prefer to keep a light-weight approach.

3.2 Summary of Code Changes

The fine-grained code changes captured by the *Change Extractor* are provided to the *Code Change Summarizer* to obtain a higher-level view of what changed in the code (see Fig. 1). ARENA's *Code Change Summarizer* follows three steps in order to generate such a view: (i) it hierarchically organizes the code changes; (ii) it selects the changes to be described; and (iii) it generates a set of natural-language sentences for the selected changes.

In the first step, a hierarchy of changed artifacts is built by considering the kind of artifact (*i.e.,* files, classes, methods, or instance variables) affected by each change. In Object-Oriented (OO) software systems, files contain classes, which in turn consist of methods and instance variables. Therefore, changes are grouped based on these relationships, *e.g.,* changed methods and instance variables that belong to the same class are grouped under that class.

In the second step, ARENA analyzes the hierarchy of changed artifacts in a top-down fashion, to select the code changes to be included in the summaries, in the following way:

1) If a file is associated to class-level changes, then the class changes are selected instead of the file changes, since in OO programming languages classes are the main decomposition unit, rather than source files. For example, consider the new file `SearcherLifetimeManager.java` in Apache Lucene 3.5. Together with this file, the `SearcherLifetimeManager` class was added. This class has a specific role within the system that is very likely to provide more information than the file.
2) If the change associated to a class is its addition or deletion, then such change is selected instead of any other change in the class (*e.g.,* the addition/deletion of its methods). Otherwise, changes on the visibility or deprecation of the class are selected. If the change associated to a class is its modification, then the changes associated to instance variable and method level are selected.
3) If the change associated to an instance variable or method is its addition, deletion, renaming, or deprecation, then such change is selected. Changes to parameters, return types, or exceptions are marked to be excluded.

Finally, in the **third** step, the *Code Change Summarizer* generates a natural language description of the selected changes, presented as a list of paragraphs. For this, ARENA defines a set of templates according to the kind of artifact and kind of change to be described. In this way, one sentence is generated for each change. For example, an added file is

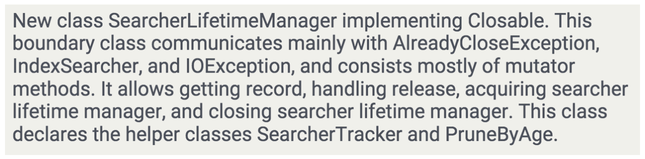

New class SearcherLifetimeManager implementing Closable. This boundary class communicates mainly with AlreadyCloseException, IndexSearcher, and IOException, and consists mostly of mutator methods. It allows getting record, handling release, acquiring searcher lifetime manager, and closing searcher lifetime manager. This class declares the helper classes SearcherTracker and PruneByAge.

Fig. 2. Summary of the `SearcherLifetimeManager` class from Lucene 3.5.

described as *New file <file_name>*, whereas a deleted file is reported as *File <file_name> has been removed*.

As stated above, the focus of OO systems is on classes. Thus, for added classes ARENA provides more detailed sentences than for other changes, by adapting *JSummarizer* [30], a tool that automatically generates natural-language summaries of classes. Each summary consists of four parts: (i) a general description based on the superclass and interfaces of the class; (ii) a description of the role of the class within the system; (iii) a description of the class behavior based on the most relevant methods; and (iv) a list of the inner classes, if they exist. We adapted *JSummarizer* by modifying some of the original templates and by improving the filtering process when selecting the information to be included in the class summary. For example, when filtering the elements that interact with the analyzed class, we excluded the classes in the package `java.lang`, since they are invariably used in every Java system. Fig. 2 shows part of an automatically generated summary for the `SearcherLifetimeManager` class from Lucene 3.5.

Deleted classes are reported in a similar way as deleted files. Changes regarding the visibility of a class are described by providing the added or removed specifier, *e.g., Class <class_name> is now <added_specifier>*. The description of modified classes consists of sentences reporting methods and instance variables added, deleted or modified. For example, a change in a method's name is reported as: *Method < old_method_name> was renamed as <new_method_name>*.

The generated sentences are concatenated according to the priority previously assigned to the changes. At this point, there may be some redundancies due to similar sentences reporting the same type of change on same-level artifacts. For example, two or more methods could be removed from the same class or, as another example, the visibility of a set of classes could change to the same level. ARENA groups similar changes in single sentences to avoid text redundancies, *e.g., Methods <method_name$_1$>, . . . , and <method_name$_n$> were removed from <class_name>*, rather than list them one at a time. A complete list of the templates that ARENA uses to generate the text descriptions is available in our replication package.

3.3 Extraction of Issues and Linkage to Commits

ARENA uses the versioning system to extract various kinds of changes to source code and other system entities, *i.e.,* to explain *what* in the system has been changed. In addition, it relies on the issue tracker to extract change descriptions, *i.e.,* to explain *why* the system has been changed. To this aim, ARENA's *Issue Extractor* (see Fig. 1) extracts the following type of issues from the issue tracker:

- *Issues describing bug fixes:* Issues with type=*"Bug"*, status=*"Resolved"* or *"Closed"*, resolution=*"Fixed"*, and resolution date included in the $[t_{k-1}, t_k]$ period.
- *Issues describing new features:* Same as above, but considering issues with type=*"New Feature"*.
- *Issues describing improvements:* Same as for bug fixes, but considering issues with type=*"Improvement"*.
- *Open issues*. Any issue with status=*"Open"* and open date in the period $[t_{k-1}, t_k]$.

Note that open issues are collected to present in the release note r_k's *Known Issues*. Based on the fields described above, ARENA has been implemented for the *Jira* issue tracker. However, it can be extended to other issue trackers (*e.g.,Bugzilla*), using the appropriate, available fields. Note that sometimes fields classifying issues as bug fix/new feature/enhancement are not fully reliable [2], [20]. In such cases, the respective elements of the release notes will be inaccurate, as ARENA does not verify the correctness of the classifications.

Once the issues are extracted, they are linked to commits. Different approaches can be used to link issues to commits. One of them is the approach by Fischer *et al.* [15], based on regular expressions matching the issue ID in the commit note. This approach, however, might produce incomplete results in some cases [5], especially when analyzing new features. We complement this approach by using a re-implementation of the *ReLink* approach defined by Wu *et al.* [40], which considers the following constraints when mapping an issue onto a commit:

1) *Committer/author and issue tracking contributor matching*. The committer in the versioning system or the author of the commit (available in *git*) must have participated in the discussion on the issue, *i.e.,* the committer/author must match with an issue tracking contributor. Note that, to match committer/author onto issue tracker contributors, we match email addresses and, when this fails, we use approximate name matching [7], [9], *e.g., Max Di Penta* matches *M. Di Penta*. For the Jira issue tracker, we found that the user ID matches the first part of the email used in *git*, *e.g.,* wherever *dipenta@unisannio.it* was used in *git* the corresponding Jira ID is *dipenta*.
2) *Time interval*. The time interval between the commit and the last comment posted by the same author/contributor on the issue tracker must be less than seven days, as suggested by Wu *et al.* [40]. We compute the time interval as the difference between the commit timestamp and the timestamp of the last comment posted by the same author.
3) *Textual similarity between the commit and the issue*. The textual similarity between the commit note and the last comment referred above must be greater than a threshold. We compute the textual similarity based on Vector Space Model (VSM) [6], after removing stop words, splitting compound identifiers using camel case, and the Porter stemming [32]. We consider as valid links those having a cosine similarity greater than 0.7. This threshold has been set by manually analyzing the linking approach on two projects (*i.e.,* Apache Commons IO and JBoss-AS) not used in our evaluation, and

it can be easily changed if needed.

3.4 Generation of the Release Note

ARENA's *Information Aggregator* is in charge of building the release note as an HTML document.

The changes are presented in a hierarchical structure consisting of the categories from the ARENA requirements defined in Section II and items summarizing each change (see Fig. 3 footnoteNote that the **[...]** in Fig. 3 have been added to fit the image and are not part of ARENA's release note.). Specifically, the release notes generated by ARENA are structured as follows:

Change category

i. *Single change overview* [<u>*more info*</u>]

∘ *Structural change description*

where:

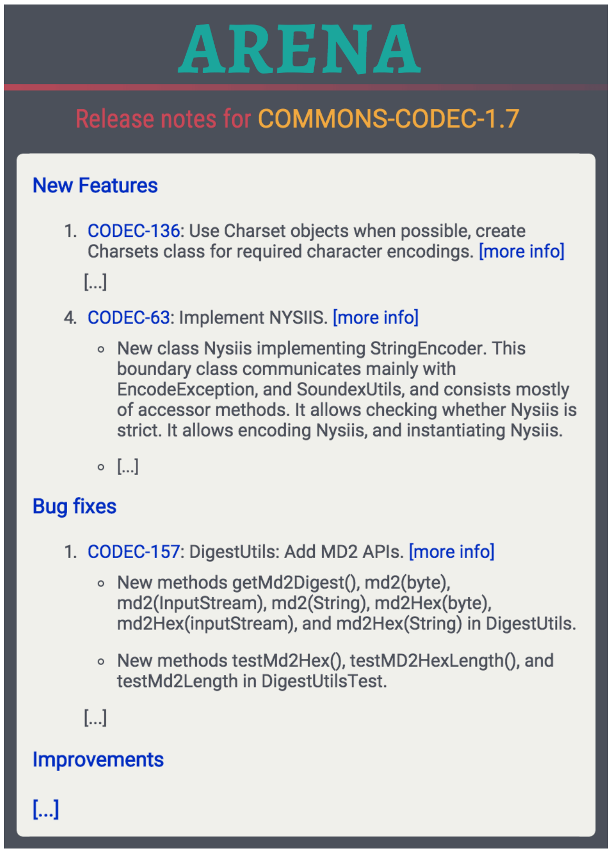
- A *change category* represents a high-level type of information grouping together similar items. The category is generally defined by the type of issue a change is linked to, *i.e., fixed bugs*, *new features*, or *improvements*. Code changes that are not mapped to any of the issues extracted from the issue tracker (*e.g.,* small changes that were not discussed among developers) are grouped based on the type of change into: *added components*, *deleted components*, *modified components*, *deprecated components*, *changes to the visibility of components*, or *refactored components*. Possible change categories are also: *documentation changes*, *license changes*, and *open issues*. Each change category in the generated release note can be expanded to see its list of items (*i.e.,* changes belonging to it). Such changes can be represented as a *single change overview* or as a *structural change description*.
- A *single change overview* is only provided for changes that are linked to an issue. It consists of the issue id linked to the corresponding entry in the issue tracker and a short description of the change as extracted from the issue's summary. The *single change overview* can be further expanded (by clicking on the "more info" link) to access the *structural change details* level, which describes how software artifacts have been impacted by the change (*e.g.,* which code components have been added, deleted, and modified to implement the change).
- A *structural change description* reports the details behind the implementation of a change and is available for all changes, both those linked to or not linked to an issue. It could refer to summarized source code changes, list of documentation artifacts added/deleted/modified, added/removed/upgraded dependencies, and artifacts that underwent changes in licenses.

Complete examples of the generated release notes can be found in our replication package.

3.5 The ARENA web tool

ARENA has been implemented as a Web application and is publicly available[6]. The current implementation of ARENA requires the following input from the user:

6. https://seers.utdallas.edu/ARENA/



ARENA

Release notes for COMMONS-CODEC-1.7

New Features

1. CODEC-136: Use Charset objects when possible, create Charsets class for required character encodings. [more info]

   [...]

4. CODEC-63: Implement NYSIIS. [more info]
   - New class Nysiis implementing StringEncoder. This boundary class communicates mainly with EncodeException, and SoundexUtils, and consists mostly of accessor methods. It allows checking whether Nysiis is strict. It allows encoding Nysiis, and instantiating Nysiis.
   - [...]

Bug fixes

1. CODEC-157: DigestUtils: Add MD2 APIs. [more info]
   - New methods getMd2Digest(), md2(byte), md2(InputStream), md2(String), md2Hex(byte), md2Hex(inputStream), and md2Hex(String) in DigestUtils.
   - New methods testMd2Hex(), testMD2HexLength(), and testMd2Length in DigestUtilsTest.

   [...]

Improvements

[...]

Fig. 3. An excerpt of the ARENA release note for Apache Commons Codec 1.7.

- *Release name*. The name of the release for which note will be generated, *e.g.,* `COMMONS-CODEC-1.7`.
- *Repository URL*. The Internet address of the versioning system containing the code change history of the project, *e.g.,* `https://github.com/apache/commons-codec.git`. Currently, ARENA supports *git*-based code repositories.
- *Issue tracker URL*. The internet address of the issue tracking system storing the list of issues associated to the project, *e.g.,* `https://issues.apache.org/jira/`. Currently, ARENA supports issue repositories managed with the *Jira* issue tracker.
- *System name on the issue tracker*. The unique name that identifies the project in the issue tracker, *e.g.,*`CODEC`.
- *Starting date*. The date when the target release started being implemented in `dd-mm-yyyy` format. This could be the date of the first commit after the previous release, *e.g.,* `20-11-2011`.
- *Ending date*. The date when the target release was finished in `dd-mm-yyyy` format. This could be the date of the last commit performed to the release, *e.g.,* `13-09-2012`.

Notice that all the previous inputs are required by ARENA for the generation of the release note. Once the user fills in this information, only one button click is necessary to generate the release note. Fig. 3 presents part of the release note generated by ARENA for release 1.7 of Apache Commons Codec.

4 Empirical Evaluation

The *goal* of our empirical studies is to evaluate ARENA, with the *purpose* of analyzing its capability to generate release notes. The *quality focus* is the completeness, correctness and usefulness (for developers) of the release notes generated by ARENA. The *perspective* is of researchers, who want to evaluate the effectiveness of automatic approaches for generating release notes, and managers and developers, who could consider using ARENA in their own company.

In the context of our empirical studies we aim at answering the following research questions:

- **RQ$_1$—Completeness:** *How complete are the release notes generated by ARENA, compared with the ones produced manually?* In other words, our first objective is to check whether ARENA is missing information that is contained in manually-generated release notes.
- **RQ$_2$—Importance:** *How important is the content captured by the release notes generated by ARENA, compared with the ones produced manually?* The aim is assessing developers' perception of the various kinds of items contained in manually and automatically-generated release notes. We are interested in the usefulness of the additional details produced by ARENA, which are missing in the notes produced manually.
- **RQ$_3$—Applicability:** *To what extent can developers rely on ARENA to generate the release notes of their ongoing projects?* The aim of this research question is to investigate the applicability of ARENA in the context of an ongoing development project. Specifically, we are interested in understanding (i) to what extent developers need to adapt the ARENA-generated notes in order to meet their needs or to complement them with missing information, and (ii) whether they perceive a benefit in such a tool support.

To answer our research questions, we performed four empirical studies having different settings and involving different kinds of participants. Table 4 summarizes, for each study, the number and kind of participants.

Study I aims at assessing the *completeness* of the ARENA release notes as compared to the original ones available on the systems' websites (**RQ$_1$**). We conducted this study on eight open-source projects to ensure good generalizability of our findings. In addition, since the goal of *Study I* does not require high experience nor deep knowledge of the application domain (as it will be clearer later), we mainly involved students in such evaluation.

Study II aims at evaluating the *importance* of the items present in the ARENA release notes and missing in the original ones and *vice versa* (**RQ$_2$**). In this case, the task assigned to participants is highly demanding, so we conducted the study only on two systems. Also, we involved experts—including original developers of the analyzed projects—since experience and knowledge of the systems were crucial.

Study III is a study conducted with industrial developers and aimed at addressing both **RQ$_1$** and **RQ$_2$**. Specifically, in Study III we asked the original developers of a system, named SMOS, to evaluate a release note generated by ARENA and to compare it with one produced by the development team leader.

Finally, *Study IV* is a six-month in-field study, where a team of developers used ARENA to generate the release notes of a medical software system, named OM, which automates blood tests. This study addresses primarily **RQ$_3$**, but, incidentally, it also provides further evidence to **RQ$_1$** and **RQ$_2$** from the context of a live project.

Table 3 lists the system releases subject of our four studies. In particular, it reports for each of them the name of the system and the specific release considered in the study, its size in terms of KLOC, and the number of commits performed by the developers in the versioning system while working on such a release. For example, in the time period between the issuing of the release 2.1.2 and 2.1.3 of Jackson-Core, 31 commits were performed in the versioning system. These are the changes that ARENA considers while generating the release notes. In addition, Table 3 also reports the size of the release notes generated by ARENA for the subject systems.

4.1 Study I—Completeness

The goal of this study is to assess the completeness of ARENA release notes (**RQ$_1$**) on several system releases, ensuring external validity, both in terms of project diversity and features to be included in the releases. The *context* of *Study I* consists of: *objects*, *i.e.*, automatically generated and original release notes from eight releases of five open-source projects (see Table 3); and *participants* evaluating the release notes, *i.e.*, one B.Sc., five M.Sc., one Ph.D. student, one faculty, and two industrial developers. Before conducting the study, we profiled the evaluators using a pre-study questionnaire, aimed at collecting information about their software development experience.

We manually looked in GitHub for releases of popular open source projects to use in the context of our study. In particular, a release r_k was selected if it satisfied the following criteria:

1) The original r_k release note was available; and
2) The release bundles for r_k and r_{k-1} (*i.e.*, the archive files packaging the software artifacts related to r_k and r_{k-1}) were available;
3) The projects issuing r_k had an issue tracker publicly available.

We stopped the selection process after collecting twenty releases belonging to eleven open source projects. Then, given the limited number of participants (ten) taking part in this study, we selected a subset of these releases such that items from each change type (except for configuration file and architectural changes—see Table 2) were present in at least one of the release notes used in the study. Eight releases from the projects Apache Cayenne, Apache Commons Codec, Apache Lucene, Jackson-Core, and Janino were selected (see Table 3).

4.1.1 Design and Planning

We distributed the original and the ARENA release notes to the evaluators in such a way that each pair of release notes[7] was evaluated by two participants. Having ten evaluators

7. By *pair* we mean the original and the generated release notes.

TABLE 3
System releases used in each study.

Study	Name	Releases	KLOC	# of commits before release	Size of the ARENA release note		
					#Change categories	#Change overviews	#Change descriptions
Study I	Apache Cayenne	3.0.2	248	5,118	10	39	254
	Apache Cayenne	3.1B2	232	2,550	9	45	280
	Apache Commons Codec	1.7	17	267	8	24	110
	Lucene	3.5.0	184	2,869	8	68	249
	Jackson-Core	2.1.0	21	170	8	11	65
	Jackson-Core	2.1.3	22	31	6	6	11
	Janino	2.5.16	26	612	7	22	141
	Janino	2.6.0	31	408	7	10	129
Study II	Apache Commons Collections	4.4.0ALPHA1	104	303	8	42	577
	Lucene	4.0.0	192	758	10	58	677
Study III	SMOS	2.0.0	23	109	4	12	31
Study IV	OM	0.1	5	83	4	18	75
		0.2	9	52	4	13	51
		0.3	12	61	5	19	51
		0.4	19	23	3	11	21

TABLE 4
Participants of the four studies.

Study	Participants	Experience
Study I	10	7 students, 1 faculty, 2 industrial developers
Study II	38	31 industrial and 7 open-source developers
Study III	5	industrial developers
Study IV	3	senior industrial developers

and eight pairs of release notes (evaluated by two participants each), it is clear that not all participants worked on the same number of systems. This is due to the fact that the release notes considered in this study had different sizes as result of the different number of changes performed across the pairs of releases r_{k-1}, r_k described by each of them. We classified the release notes of the eight software systems into three sets:

- *Large*: Apache Cayenne 3.0.2, Apache Cayenne 3.1B2, and Lucene 3.5.0. Each of these releases was the result of over 2,500 changes committed in the versioning system (see Table 3).
- *Medium*: Janino 2.5.16, and Janino 2.6.0. Each of these releases was the result of over 400 changes committed in the versioning system.
- *Small*: Apache Commons Codec 1.7, Jackson-Core 2.1.0, Jackson-Core 2.1.3. Each of these releases was the result of less than 300 changes committed in the versioning system.

We then distributed the ten participants as follows: six participants worked on one *large* release note, each; two worked on two *medium* release notes, each; and two worked on three *small* release notes, each. This was done to balance the workload among the participants while ensuring two evaluators for each pair of release notes.

We provided each participant with:

1) a pre-study questionnaire aimed at gathering information about the participants' background, in particular about their industrial and programming experience (in years) and their habits in exploiting release notes when using a new available system release;
2) the generated release note and the original release note to be compared by the participants, as described later in this section;
3) a post-study questionnaire, where we asked participants to assess the general usefulness of the various categories of information provided by the ARENA release notes (*e.g., new features, bug fixes, refactored code components, etc.*). For each category, the participants provided their level of agreement to the claim *"The category should be included in the release note"* using a 4-point Likert scale [31] (*strongly agree, weakly agree, weakly disagree, strongly disagree*). We opted for a 4-point Likert scale since we wanted to avoid "neutral" answers, thus pushing participants to take a position about the assessed release notes. Neutral answers are less useful in assessing our approach and in getting feedback for its improvement.

Participants were asked to determine and indicate for each item in the original release note whether: (i) the item appears in the generated release note with roughly the same level of detail; (ii) the item appears in the generated release note but with less details; (iii) the item appears in the generated release note and it has more details; or (iv) the item does not appear in the generated release note. In order to avoid bias in the evaluation, we did not refer to the release notes as "original" or "generated". Instead, we labeled them as *"Release note A"* and *"Release note B"*. In addition, the participants were asked to assess the relevancy of all the extra items (as a group) in the generated release note, which were not present in the original one. Ideally, we should have asked them to assess the content of each individual item. Assessing these items as a group introduces a threat to the validity of the results, as individual items may be rated differently than the group as a whole. We accept this threat to the validly of the results, as assessing each item individually would have been too demanding for the evaluators (in terms of time and effort). For these reasons, we asked them to provide an *overall* evaluation of these items, asking them to indicate—using a 4-point Likert scale [31] (*strongly agree, weakly agree, weakly disagree, strongly disagree*)—whether these items should be part of the release note. The rationale for choosing a 4-point Likert scale over a 5-point one is the same as above.

When all the participants completed their evaluation, a Ph.D. student from the University of Sannio[8] analyzed them to verify and arbiter any conflict in the evaluation of the

8. None of the authors.

TABLE 5
Evaluation provided by the study participants to the items in the original release notes.

System	Release	Absent	Less Details	Same Details	More Details
Apache Cayenne	3.0.2	15%	5%	0%	80%
Apache Cayenne	3.1B2	16%	0%	0%	84%
Apache Comm. Codec	1.7	13%	5%	5%	77%
Apache Lucene	3.5.0	37%	39%	8%	16%
Jackson-Core	2.1.0	25%	8%	33%	33%
Jackson-Core	2.1.3	0%	0%	50%	50%
Janino	2.5.16	0%	6%	0%	94%
Janino	2.6.0	12%	38%	0%	50%
Average		**14%**	**13%**	**12%**	**61%**

TABLE 6
Answers to the claim: The extra information in the generated release notes should be included.

System	Release	Strongly agree	Weakly agree	Weakly disagree	Strongly disagree
Apache Cayenne	3.0.2	1	1		
Apache Cayenne	3.1B2	1	1		
Apache Commons Codec	1.7	2			
Apache Lucene	3.5.0		1	1	
Jackson-Core	2.1.0	1	1		
Jackson-Core	2.1.3	2			
Janino	2.5.16	2			
Janino	2.6.0	1		1	
Total evaluations		**10**	**4**	**2**	**0**

items present in the original release notes. For example, if the two evaluators judging the same release note provided opposite responses for the same item (*e.g.*, the item is present in the generated release note *vs.* not present), then the Ph.D. student solved the conflict by verifying the presence/absence of the disputed item. Out of 144 evaluated items, 43 (30%) exhibited some conflict between the two evaluators. Only five of them (3%) showed strong conflict between the evaluators, e.g., "*the item appears*" vs. "*the item is missing*". The other 38 cases had slight deviations in the evaluation, *e.g.*, "*the item appears with roughly the same level of detail*" vs. "*the item appears but with less details*."

4.1.2 Participants' background

Before presenting the results of the first study, we analyze the background of the evaluators. Six out of ten evaluators have experience in industry, ranging from one to five years (median 1.5). They reported four to 20 years (median 5.0) of programming experience, of which two to seven are in Java (median 4.5). Also, seven out of the ten evaluators declared that they routinely check release notes when using a new available system release, where they look for new features (6 evaluators), fixed bugs (4), modified code components (2), and compatibility with previous releases (2).

4.1.3 Analysis of the Results

Table 5 summarizes the answers provided by the evaluators when asking about the presence of items from the original release notes in the release notes generated by ARENA. On average, ARENA correctly captures, at different levels of detail, 86% of the items from the original release, missing only 14%. In particular, ARENA provides more details for 61% of the items present in the original release notes, the same level of details for 12%, and less details for 13% of the items. The following is an exemplar situation where an item in the generated release has more details than in the original release. In the release note of Apache Commons Codec 1.7, the item "*CODEC-157 DigestUtils: Add MD2 APIs*" describes the implementation of new APIs. In the ARENA release note, the same item is reported as shown in Fig. 3. ARENA reports the addition of new APIs to the `DigestUtils` class and it also explicitly includes: (i) which methods are part of the new APIs; and (ii) the test methods added in `DigestUtilsTest` to test the new APIs.

An outlier case is for Lucene 3.5.0, where ARENA missed 14 (37%) of the items present in the original release note. Upon closer inspection, we found that eight of the missed items are bug reports fixed in a previous release, yet (for an unknown reason) reported in the release note of Lucene 3.5.0 (*e.g.*, issue *LUCENE-3390*). If we disregard such issues, the percentage of missed items in this release drops to 20%, almost in line with the other release notes.

We analyzed the items that ARENA missed in the other release notes. We found that all the missed items are due to a slight deviation between the time interval analyzed by ARENA and the one comprising the changes considered in the original release note. As explained in Section 3, we make an assumption about the time period of analysis, going from the r_{k-1} release date t_{k-1} until the r_k release date t_k. This problem would not occur in a real usage scenario, where the developer in charge of creating a release note using ARENA can simply provide the best time interval to analyze via the ARENA GUI (see Section 3.5).

In addition, the evaluators were asked to judge whether the items and details (as a whole) appearing in the ARENA release notes, but missing in the original ones, should be included. Their answers are summarized in Table 6. Remember that each release note was evaluated by two participants, for a total of 16 evaluations. The majority of the participants selected *strongly agree* or *weakly agree* (*i.e.*, 14 out of the 16 evaluations). Also, for three release notes (*i.e.*, Apache Commons Code 1.7, Jackson Core 2.1.3, and Janino 2.5.16), both evaluators *strongly agreed* that the extra information provided by ARENA should be included in the release notes.

Evaluators provided positive feedback about the relevancy of the additional details present in the ARENA release notes. A representative one is: "*Knowing the changes at fine-grained level is very useful for a developer that has to work on the new system release. Also, the information about deprecated code components is very useful.*" Note that in two cases—*i.e.*, Apache Lucene 3.5.0 and Janino 2.6.0—one of the evaluators *weakly disagreed* on the additional details provided by ARENA. The reason behind the assessment was the same in both cases: "*It is useful to list the changes in the code associated with bugs, improvements, or any specific purpose. [...] But the items containing only the [code] changes without any purpose result confusing. It is hard to say why these changes were made.*" Such items are the result of commits that could not be linked to any bug fix, improvement, or new feature requests in the issue tracker system. This highlights the importance of linking issues to changes. Instead, the other two participants (positively) evaluating the same release notes pointed out the usefulness of the extra information and, in particular, of the details about added components.

Finally, Table 7 summarizes the answers of the eval-

TABLE 7
Agreement to the claim: the category should be included in the release note.

Item	Strongly Agree	Weakly Agree	Weakly Disagree	Strongly Disagree
New features	90%	10%	0%	0%
Bug fixes	100%	0%	0%	0%
Improvements	80%	20%	0%	0%
Added code components	50%	50%	0%	0%
Modified code components	70%	30%	0%	0%
Deleted code components	70%	30%	0%	0%
Deprecated code components	80%	20%	0%	0%
Changes to visibility of code components	50%	20%	20%	10%
Refactored code components	10%	60%	30%	0%
Changes to used libraries	70%	30%	0%	0%
Changes to licenses	70%	20%	0%	10%

uators to our post-questionnaire, where we asked them about their perception of the overall usefulness of various categories of information provided by the ARENA release notes. In general, the information considered by ARENA is considered important by participants, with one exception represented by the refactored code components, for which only 10% of participants *strongly agree* about the need for including them in the release notes. As mentioned, we included this kind of information in the release notes based on our initial survey of existing release notes. These questions are not meant to provide definite validation of our choices, but rather a "sanity check".

Summary of Study I (RQ$_1$)—Completeness. The ARENA release notes capture most of the items from the original release notes (86% on average) and most of the missing items can be included by adjusting the considered time interval. Furthermore, 88% of the evaluators agree (strongly or weakly) that the additional information extracted by ARENA should be included in the release notes. Finally, all categories of information included by ARENA in the release notes are considered important by most evaluators, except for refactoring operations, which have less support.

4.2 Study II—Importance

The goal of *Study II* is to evaluate the perceived importance of the captured and missed items in the ARENA release notes from the perspective of external users/integrators, as well as from the perspective of internal developers. In other words, the study aims at determining whether ARENA is missing particularly important information that original release notes contain, or whether it provides unimportant details to developers, resulting in long and tedious to browse/read release notes. The *context* of this study consists of: *objects*, *i.e.*, automatically generated and original release notes from one release of two open-source projects (see Table 3); and *participants* evaluating the release notes, *i.e.*, 38 professional and open-source developers, including three developers of each object project. One release of Apache Lucene and one release of Apache Commons Collections were selected for the study. The conditions to select the release notes were the same as in *Study I*.

4.2.1 Design and Planning

We performed *Study II* by using an online survey. We emailed the survey to several open-source developers registered in the Apache repositories and professional developers from around the world. The questionnaire consisted of two parts on: (i) the participants' background and their experience in using and creating release notes; and (ii) the evaluation of the ARENA release notes and the original release notes for Apache Lucene 4.0.0 and for Apache Commons Collections 4.4.0ALPHA1. Note that we decided to use different systems from the ones used in *Study I* in order to increase the number of systems in the ARENA evaluation and to ensure better external validity. We targeted major releases of medium sized systems (100<KLOC<200).

The evaluation of each release note was divided in two stages. In the first stage, participants were asked to indicate for the types of items (*e.g., Major Changes*) of the original release note, which were missing in the release note generated by ARENA whether each one was: (i) not at all important; (ii) unimportant; (iii) important; or (iv) very important. A similar process was followed in the second stage, but this time for assessing the importance of types of items of the ARENA release note that were missing in the original one. In both stages, we pointed out that some items present in a release note and (apparently) missing in the other one might simply be represented in different ways. In the case of Lucene 4.0.0, for example, the items under the *Improvements* category in the ARENA release note are listed under the *Optimizations* category in the original release note.

4.2.2 Participants' background

Similar to *Study I*, in this study we profiled the participants by collecting information on their experience in software development and their experience in using and creating release notes, and, if they had created such documents, what kind of content do they include in them. Among the 38 evaluators, 31 are professional developers who reported experience in software development ranging from two to 30 years (median 8). The other seven participants are open-source developers, ranging from seven to 25 years of experience in software development (median 13). Three of the participants are developers of Apache Commons Collections, while other three are developers of Apache Lucene. Also, 26 out of the 38 evaluators declared that they use release notes frequently (*i.e.,* more than once a month) or occasionally (*i.e.,* once a month), mainly to check for bug fixes and new features in a software system. In addition, 16 developers declared that they check in the release notes of their project's dependencies for compatibility issues and changes that might arise from the new releases.

Only six evaluators (16%) reported that they have never created release notes. Among the other 32 participants, 22 (58%) reported having created release notes many times (*i.e.,* more than eight times), eight (21%) reported having done it a few times (*i.e.,* three to eight times), and two (5%) declared having created a release note once or twice. Our survey was designed to ask only highly experienced developers in creating release notes (*i.e.,* the 22 cited above) details about this task. Fig. 4 presents the participants' answers on the time and difficulty of creating release notes. Specifically, 64% of the evaluators (*i.e.,* 14 of them) considered this task as difficult or very difficult, while 36% rated it as easy or very easy (median=difficult). The participants also reported a median of *between four and eight hours* to create release notes. One of the participants explained that the time needed to create a release note depends on the release, claiming that

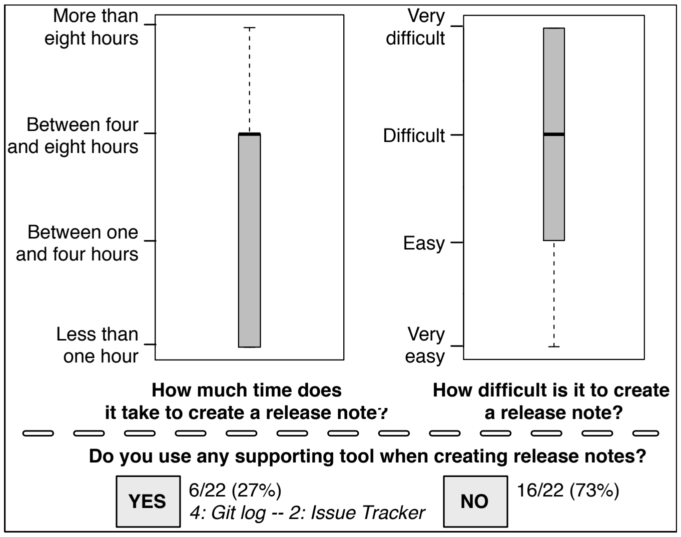


Fig. 4. Difficulty in creating release notes.

he worked in the past on *"a major release of a software company product for which the creation of the release note took three days of work."* Finally, only seven evaluators (31%) declared using a supporting tool, such as, issue trackers or version control systems, when creating release notes. Note that ARENA is the only existing automated tool specially designed to support the generation of release notes. Furthermore, ARENA is able to shorten the time devoted to this task, as less than ten minutes are needed to generate a release note for a release r_i of a medium-sized system ($\sim$50 KLOC) subject to few hundreds of changes ($\sim$500) in the time period between r_{i-1} and r_i.

We also asked the 22 participants with high experience in creating release notes about the kind of content that they usually include in these documents (see Fig. 5). *New Features* and *Bug Fixes* are, by far, the most common items in the release notes: most of the participants reported including them often (21 and 20 evaluators, respectively). *Enhanced Features* are also frequently included in the release notes (often by 11 participants and sometimes by ten). Both results confirm the findings of our survey on 990 existing release notes presented in Section 2. Other frequently included items are *Deleted and Deprecated Code Components*, *Changes to Licenses*, *Library Upgrades*, and *Known Issues* (median=often), while items related to *Refactoring Operations* and *Added*, *Replaced* or *Modified Code Components* are sometimes included. Instead, evaluators rarely include changes to *Configuration Files*, *Documentation*, *Architecture*, and *Test Suites* in the release notes.

4.2.3 Analysis of the Results

The answers provided by the 38 participants on the importance of different kinds of content from the original and the generated release notes are summarized in Fig. 6 and Fig. 7 for Commons Collections and Lucene, respectively. Note that the boxplots depicted for the two systems refer to different sets of information. This is because the two release notes, *i.e.,* the one for Commons Collections (Fig. 6) and the one for Lucene (Fig. 7), included different categories of items in the original release note as well as in the one generated by ARENA. For example, information about the "#Items in each category" (provided in all release notes generated by ARENA) was not provided in the original release note of Commons Collections (Fig. 6) but it was provided in the original release note of Lucene (Fig. 7). Thus, the boxplot for such a category is present in Fig. 6 but not in Fig. 7. Among the items present in the original release notes and missed by ARENA, the ones considered important/very important by developers are: *Major Changes* from Commons Collections, summarizing the most important changes in the new release; and *API Changes*, *Backward Compatibility*, and *Optimizations* from Lucene.

On the one hand, the *Major Changes* section is not present in the ARENA release notes, but future efforts will be oriented to implement an automatic prioritization of changes in the new release, which will allow ARENA to select the most important ones to define such a category. On the other hand, the information present in *API Changes*, *Backward Compatibility*, and *Optimizations* in the original release notes is present in the ARENA release notes, but organized differently. For example, the removal of the `SortedVIntList` class is reported in the original release notes in the *Backward Compatibility* category, while ARENA puts it under *Deleted Code Components*. Thus, ARENA is not missing any important information here. As for the other items present in the original release notes and not in the ones generated by ARENA, they are generally classified as unimportant/not important (see Fig. 6 and Fig. 7).

Regarding the contents included in the ARENA release notes and missing or grouped differently in the original release notes, most of them (nine out of 11 different kinds of content) were predominantly assessed as important or very important (by 28 developers, in average). The *Improvements* category is considered important/very important by 34 developers for Commons Collections and 33 for Lucene, respectively. While the items contained in this category are present in the *Optimizations* section of the Lucene release note, they are absent in the Commons Collections one. Participants also considered as important or very important the categories covering *Known Issues* (by 32 developers) and *Deletion* (29), *Addition* (28), *Deprecation* (29), and *Visibility Changes* (26) *of Code Components*. The evaluators (24 for Commons Collections and 19 for Lucene) also considered important the *fine-grained changes* (*i.e.,* changes at code level) provided by ARENA when describing new features, bug fixes and improvements, and the *links to the change requests* (30) listed under such categories. Note that most of the above categories are absent in the original release notes. For instance, while in the original release note of Lucene one deleted class was listed under the *Backward Compatibility* section, ARENA highlights 76 classes that were deleted in the new release. Surprisingly, *Refactoring Operations* were considered as unimportant in both release notes (by 24 developers for Commons Collections and 25 for Lucene). This might be due to the level of detail that ARENA is currently able to provide in this matter (*i.e.,* the refactored source code files, without explaining what exactly was done to them).

Evaluation made by the original developers. The results presented above are obtained considering the responses of all the 38 surveyed participants. However, among them there are six of the original Lucene and Commons Collections developers (three of each project). Thus, it is worth-

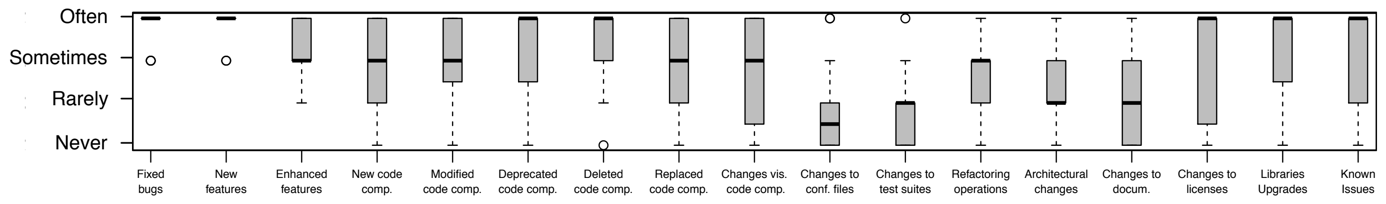


Fig. 5. What kind of content do you include in release notes? (22 developers)

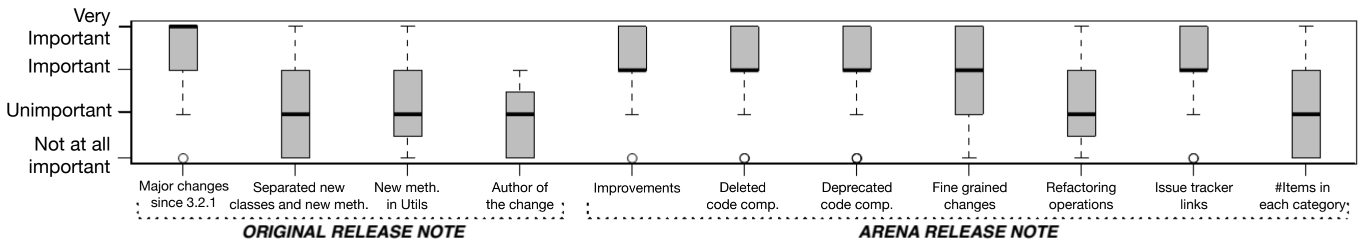


Fig. 6. Importance reported by the evaluators for the content of Commons Collections release notes.

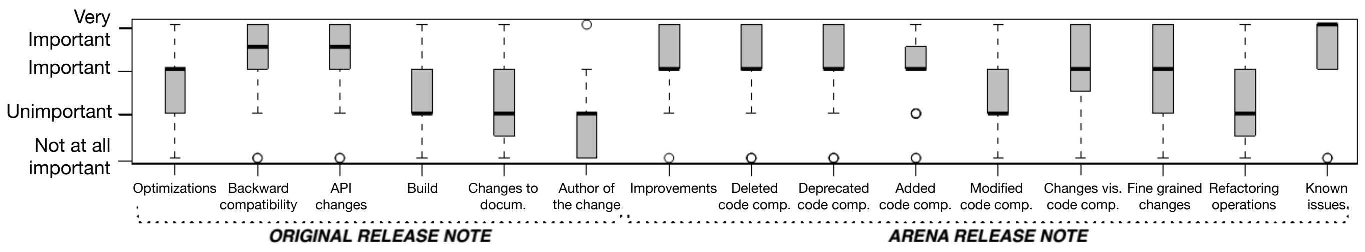


Fig. 7. Importance reported by the evaluators for the content of Lucene release notes.

while to independently analyze the original developers' responses (*i.e.,* of those developers with more knowledge of the object systems) to gain a different perspective on the release notes generated by ARENA.

In the case of the contents explicitly provided by the original release note of Commons Collections, its three developers strongly agreed on the importance of the *Major Changes* category and weakly agreed on the importance of showing the *author of the change* (marked as important by two developers and unimportant by the other one). In contrast, having separate categories describing new classes and new methods (as opposed to ARENA's single *New Code Components* category) was ranked from very unimportant to very important, showing a strong disagreement among participants. The three developers strongly agreed on the importance of all the attributes offered by the ARENA release notes, except for the *number of items in each category* (*e.g.,* indicating the number of added code components near the *New Code Components* category), which was considered unimportant by two developers and very important by the other one.

Turning to the original release note of Lucene, its three developers strongly agreed on the importance of the *API Changes* and *Backward Compatibility* categories, and on the unimportance of the *authors of the changes*. There was little agreement on the importance of the other content of this release note. However, once again, the three developers strongly agreed on the importance of all the content included in the ARENA release note, except for the *Refactoring Operations* category, which was marked as important by two of the developers and unimportant by the other one. Most of the developers' responses are in line with the responses of all the other evaluators, presented above. This indicates that, at least in the case of Commons Collections and Lucene, developers and integrators are interested in the same content within the release notes.

Qualitative feedback. We also allowed participants to comment on the ARENA release notes in a free text box at the end of the survey. The evaluators provided positive feedback about the release notes generated by ARENA. In total, we collected eight positive comments related to the content, potential usefulness and readability of the release notes generated by ARENA, and two negative comments on their presentation and low-level detail of information. Three of the evaluators commented on desired features in the generated release notes, *e.g.,* links to changes in the versioning systems, more details and explanations of particular changes, and browsing options. Representative examples include: *"In general, they are very readable. I think they are aimed at engineers more than at non-engineers. [...] for something* *that is consumed as an API, such as an open-source library or framework, I think these kind of notes are ideal."* Another developer commented *"If it's fully automated (I'm not sure) ARENA is a great tool."* Some of the contents provided by the generated release notes were considered unimportant in some cases. One of the reasons behind such assessments was that *"Every item needs a description to be useful. For example, the Added Components section needs a description of each component, and the Modified Components section needs a description of what is the meaning of the modification."* This comment refers to items in the generated release notes that do not have a textual description motivating the change; this happens for commits that cannot be linked to any bug fix, improvement, or new feature request in the issue tracker system. Since ARENA is meant to support the creation of release notes, they can be augmented by the users with additional information, according to their needs.

Summary of Study II (RQ$_2$)—Importance. Most information included in the ARENA release notes is considered important or very important by developers. Also, most information considered as important in the original release notes is captured by ARENA (although sometimes in a different fashion), with the exception of the *Major changes* category that we plan to include by prioritizing changes.

4.3 Study III—Study with Industrial Developers

The goal of *Study III* is to allow project experts: (i) to evaluate the generated release note on its own, including their perceived usefulness; and (ii) to compare the generated release note with one manually produced by their team leader. The *context* of this study consists of *objects* and *participants*. As objects we use one release of the SMOS system, a software developed to support the communications between schools and the students' parents. Its first release (*i.e.,* SMOS 1.0) was developed in 2009 by M.Sc. students. Its code, composed by almost 23 KLOC, is hosted in a *git* repository and has been subject to several changes over time. This led to the second release (*i.e.,* SMOS 2.0) nearly two years later. The participants of this study are five (out of seven) members of the original development team of this system, all being nowadays industrial developers.

4.3.1 Design and Planning

To have a baseline for comparison, we asked the leader of the original development team to generate a release note of SMOS 2.0. We considered him to be the best qualified person to create a complete and accurate release note for SMOS. During the creation of the release note, the leader had access to: (i) the source code of the two releases; (ii) the list of changes (as extracted from the versioning system) performed between the two releases; and (iii) information from the issue tracker containing the change requests implemented in the time period between SMOS 1.0 and 2.0. Finally, we asked him to report the time he spent on producing the release note. This information, however, is only indicative and cannot be really representative of other projects (different development teams could rely on different tools).

We conducted this study in two stages. In the first stage, the developers (excluding the project leader) evaluated the release note generated by ARENA based only on their knowledge of the system. As in *Study I*, we did not refer to this one as an automatically generated release note (but as *Release note A*). We asked the developers to judge the next four statements with the possible answers on a 4-point Likert[9] scale [31] (*strongly agree*, *weakly agree*, *weakly disagree*, *strongly disagree*):

1) The release note contains all the information needed to understand what changed between the old and the new release;
2) All the information reported in the release note is correct;
3) There is redundancy in the information reported in the release note;
4) The release note is useful to a developer in charge of evolving the software system.

In addition, we asked them to provide a free-text argument for each question. We also asked them what kind of additional information should be included in the release note. In the second stage, we asked the SMOS developers to evaluate—using the same questionnaire—the release note manually produced by the team leader, calling it *Release note B*.

After they completed the questionnaire, we asked the developers to re-evaluate the release note generated by ARENA. To this end, we provided them with their completed questionnaires and asked if they would change any answer and how, in light of the analysis of the second release note. We allowed participants to revise their original evaluation because during the analysis of the first release note they did not have any baseline for comparison. The second release note offered participants this baseline, which, as a result, could (positively or negatively) change the evaluation of the first release note.

The main difference between *Study III* and *Study II* is that in *Study III* developers evaluated the ARENA-generated release note and the release note produced by an original developer as a whole, without focusing on missing information nor without performing a comparison of single items contained in both release notes. It is noteworthy that the evaluation in *Study III* was performed without knowing whether a release note was automatically or manually generated.

4.3.2 Analysis of the Results

As explained before, in the first stage of our evaluation with SMOS developers, we asked the team leader to manually generate the release note for SMOS 2.0. The team leader took 82 minutes to manually summarize the 109 changes that occurred between releases 1.0 and 2.0. This resulted in a release note with 11 items, each one grouping a subset of related changes. For example, one of the items in this release note was:

> *Several changes have been applied in the system to implement the Singleton design pattern in the classes accessing the SMOS database. Among the most important changes, all constructors of classes in the* `storage` *package should now not be used in favor of the new*

9. As already discussed for *Study I*, we opted for a 4-point Likert scale to avoid "neutral" answers.

TABLE 8
Evaluation provided by four original developers to the release note generated by ARENA for SMOS. In parenthesis, the evaluation provided to the manually-generated release note.

Claim	Subject ID			
	1	2	3	4
The release note contains all the information needed to understand what changed between the old and the new release	3 (2)	4 (3)	4 (3)	4 (3)
All the information reported in the release note is correct	4 (3)	4 (4)	4 (4)	4 (4)
There is redundancy in the information reported in the release note	1 (1)	1 (1)	1 (1)	1 (1)
The release note is useful to a developer in charge of evolving the software system	3 (3)	4 (3)	4 (3)	4 (3)

1=strongly disagree, 2=weakly disagree, 3=weakly agree, 4=strongly agree

methods `getInstance()`, returning the single existing instance of the class. This resulted in several changes to all methods in the system using classes in `storage`.

In the meantime, the remaining four developers evaluated the release note generated by ARENA. The results are reported in Table 8 (see the numbers *not* in parenthesis).

Developers *strongly agreed* or *weakly agreed* on the fact that the ARENA release note contains all the information needed to understand the changes performed between the two SMOS releases. In particular, the only developer answering *weakly agree* (ID 1 in Table 8) explained that *"the release note contains all what is needed. However, it would be great to have a further level of granularity showing exactly what changed inside methods."* In other words, this developer would like to see, on demand, the source code lines modified in each changed code entity. While this information is not present in the ARENA release note, it would be rather easy to implement. All other developers answered with a *strongly agree* and one of them explained her score with the following motivation: *"I got a clear idea of what changed in SMOS 2.0. Also, I noticed as I was not aware about some of the reported changes."* Two of the other developers, answering with *strongly agree*, indicated additional information that could further improve the completeness of the release note: *"information about who performed each change would be great to contact her/him for clarification."* Note that this information is quite easy to include in ARENA.

All developers *strongly agreed* on the correctness of the information reported in the release note generated by ARENA (see Table 8). Also, they *strongly disagreed* on the presence of redundancy in the information reported in the release note. In particular, one of them explained that *"[the] information is well organized and the hierarchical view allows visualizing exactly what you want, with no redundancy."*

Finally, all developers *weakly agreed* or *strongly agreed* on the usefulness of the ARENA release note for a developer in charge of evolving the software system, for example, *"the release note is very useful to get a quick idea of what changed in the system and why."* The only developer answering *weakly agree* commented: *"developers are certainly able to get a clear idea about what changed in the system, but they may still need to look in source code for details."* Note that this developer was the one asking for granularity at the line level.

In the second part of this study, we asked the same four developers to evaluate (by using the same questionnaire) the release note manually generated by their team leader. Table 8 reports these results in parenthesis. In this case, three developers *weakly agreed* on the completeness of the release note, while one *weakly disagreed*. As comparison, on the completeness of the release note generated by ARENA

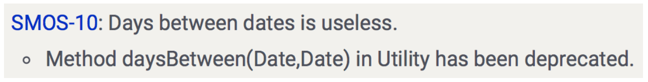

SMOS-10: Days between dates is useless.
- Method daysBetween(Date,Date) in Utility has been deprecated.

Fig. 8. An excerpt of ARENA release notes for SMOS.

three developers *strongly agreed* and one *weakly agreed*.

The developer answering *weakly disagree* motivated her choice explaining that *"the level of granularity in this release note is much higher as compared to the previous one. Thus, it is difficult to get a clear idea of what changed in the system. Also, information about the updated libraries is missing."* When talking about "the granularity" of the release note, the developer refers to the fact that changes are not always reported at method level as it was in the ARENA release note. This is likely due to the fact that changes in the versioning system are stored at file level granularity and, thus, it was simpler for the developer manually writing the release note to list the files changed to implement a new feature, a bug fix, *etc.*, without going at method-level granularity. Also, one of the developers that *weakly agreed* on the completeness of the manually-generated release note referred to the previously evaluated one (*i.e.*, the one generated by ARENA), claiming that: *"it is almost complete, but the previous one was more precise."*

Three developers *strongly agreed* on the correctness of the information reported in the manually-generated release note, while one of them answered *weakly agree*, reporting an error present in the release note: *"the method `daysBetweenDates` has been deprecated, not deleted."* In particular, the manually-generated release note contained the item:

The method `daysBetweenDates` has been deleted.

while the release note generated by ARENA reported the information shown in Fig. 8. This highlights that, even when preparing a release note for a moderately small software system like SMOS, developers can include errors, which are avoidable by using a fully automated approach like ARENA.

Finally, all developers *strongly disagreed* on the presence of redundant information in the manually-generated release note, and *weakly agreed* on its usefulness. For the ARENA release note, instead, three of developers *strongly agreed* on its usefulness. In this regard, some of the explanations provided by developers highlighted the usefulness of the details provided by ARENA: *"this release note is less detailed than the previous one, but it is still useful,"* and *"also in this case a developer needs to look into the source code to get a clear idea of what changed, but she has to spend more time to find the modified pieces of code, since several changes are just reported at* *file level."* At last, nobody decided to change the scores given to the ARENA release note after having seen the manually-generated one.

Summary of Study III. The SMOS developers judged the ARENA release note as more complete and precise than the one created by the team leader ($\mathbf{RQ}_1$). Moreover, the extra information included in the generated release note makes it to be considered more useful than the manual one ($\mathbf{RQ}_2$).

4.4 Study IV—Six-month In-field Study

Study IV is an in-field study conducted over six months with the *goal* of assessing the quality and usefulness of the release notes generated by ARENA in a real, everyday working environment. The *context* of this study consists of *objects*, *i.e.,* four different releases of OM[10] (see Table 3), a medical software system that automates blood analyses by guiding a robot in performing tasks that are typically performed by human operators (*e.g.,* mixing reagents to human blood in order to perform HIV test). OM is developed by an Italian company (from now on simply referred as *software company*) that started its implementation in September 2014. The *participants* of the study are members of the initial OM development team, which consisted of three developers, including two senior developers with over ten years of experience each and the team leader with more than 15 years of experience in software development. All of them had past experience in writing of release notes, with the team leader highlighting this activity as one of his main duties when coordinating a team.

OM has been commissioned by an analysis laboratory and, as established in the contract between the *software company* and the laboratory: (i) it has been incrementally released over the past six months (*i.e.,* from September 2014 to March 2015) in four different releases (*i.e.,* on average, a release each 45 days), and (ii) each of the issued releases has been accompanied by detailed documentation, including a release note. The *software company* agreed to use ARENA for generating the release notes of OM over this six-month implementation activity and to provide us with feedback on the strengths and weaknesses of our tool. OM is hosted in a *git* repository and uses *Mantis*[11] as issue tracker.

4.4.1 Design and Planning

In September 2014, when the OM development started, one of the authors instructed the team on how to use ARENA in order to generate release notes. Then, the following procedure was adopted each time a new OM release was issued:

1) ARENA was used by the team to automatically generate the release note. An adapted version of the ARENA tool supporting the extraction of issues from Mantis was used in this case.
2) The team answered a questionnaire asking their level of agreement to the same four claims from the questionnaire used in *Study III* (see Table 9). The answers were given using the same 4-points Likert scale [31]. Also in this case, we asked the team to provide a free-text argument for each question. Additionally, since the release note was one of the official documents to issue as part of the OM contract, we asked the participants to list the information that they added/deleted/modified in the release note generated by ARENA, before including it as part of the official release documentation. Note that, given the small size of the team (*i.e.,* three developers), we only asked for one filled-in questionnaire as representative for the entire team's perception of the ARENA's release notes.
3) Finally, the team sent to the author: (i) the release note that they generated with ARENA, and (ii) the answers to the questionnaire.

10. OM is a pseudonym of the software system object of our study. The company requested to anonymize the software system in the study.
11. https://www.mantisbt.org

As previously said, four OM releases (from 0.1 to 0.4) were issued in the six months of our study, leading to four generated release notes and corresponding answered questionnaires. In April 2015, at the end of our study, one of the authors conducted a two-hours, semi-structured interview with the OM team, in order to gather further qualitative feedback about ARENA and its release notes. The interview included (but was not limited to) the following questions:

Q1 *Would you like to adopt ARENA in the company where you work?*
Q2 *What are the three major strengths and weaknesses of ARENA?*

Also, the interviewer asked the team members to comment about and/or clarify some of the answers provided in the questionnaires they filled in the previous six months.

4.4.2 Analysis of the Results

Table 9 reports the results of the evaluation provided by the development team to the four OM release notes generated by ARENA. Notice that for the first OM release (*i.e.,* OM 0.1) there is no previously existing release and, indeed, most of its information refers to new features implemented from scratch. The team always *strongly* or *weakly agreed* on the fact that the ARENA release notes contain all the information needed to describe the changes performed between the old and the new release. A reason behind the *"weakly agreed"* assigned to the release notes of OM 0.2 and 0.4 is reflected by the comment left in the free-text form questionnaire of OM 0.4:

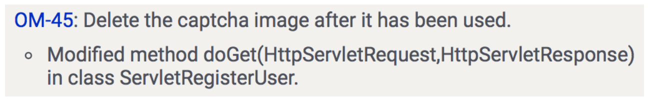
> *"Overall the release note looks very good. However, while for most of the changes the title of the issue is enough to document the rationale behind the performed changes, there are specific changes that would benefit from the inclusion of additional information to document in the release note the reasons behind changes performed in the system. An example in this release is the issue* `OM-45`*."*

During the unstructured interview we asked the team members additional insights about this answer. As explained in Section 3.4, ARENA organizes the changes in the release note based on the category to which they belong (*e.g., Improvements, etc.*). These categories can then be expanded to list the set of items (*i.e.,* issues) belonging to each of them (*e.g.,* the list of improvements implemented in the new release). When structural changes detected in the source code are linked to an issue, its title is presented as a

TABLE 9
Evaluation provided by the involved team the release notes generated by ARENA for the four OM releases.

Claim	OM Release			
	0.1	0.2	0.3	0.4
The release note contains all the information needed to understand what changed between the old and the new release	4	3	4	3
All the information reported in the release note is correct	4	4	4	4
There is redundancy in the information reported in the release note	2	2	2	2
The release note is useful to a developer in charge of evolving the software system	4	4	4	3
1=strongly disagree, 2=weakly disagree, 3=weakly agree, 4=strongly agree				

description of the change (see for instance Fig. 8 from *Study III*). The team pointed out that, although in general this is enough to explain why certain structural changes were performed, there are a few cases where additional details on the rationale of the changes are needed. In particular, the team refers to the issue `OM-45`, which represents an improvement implemented in OM 0.4 and is shown in Fig. 9, as reported by the ARENA release note. The team found important to document in the release note why the randomly generated captcha image used to discriminate between humans and bots during a "new user registration" is deleted right after the user registration is completed. They suggested to integrate into ARENA a mechanism to automatically generate a summary of the issue description and its discussion (*i.e.,* the comments left by the developers). This is part of our future work on ARENA, in which we can make use of approaches that automatically summarize issue reports [23], [24], [35].


OM-45: Delete the captcha image after it has been used.

- Modified method doGet(HttpServletRequest,HttpServletResponse) in class ServletRegisterUser.

Fig. 9. An excerpt of the ARENA release note for OM 0.4.

The participants *strongly agreed* on the correctness of the information reported in all four ARENA release notes. In other words, they did not find code changes assigned to the wrong release. Note that in this study the developers manually provided the time interval for each release. Thus, such result is an important confirmation that, when the time interval to analyze is correctly provided, the information extracted by ARENA is correct.

Consistently across the four releases, participants *weakly disagreed* on the presence of redundancy in the ARENA release notes (see Table 9). The justification for this question was also consistent across the four release notes:

> *"The possibility to expand and retract the details about any of the different items allows the reader of the release note to control in some way the level of redundancy she wants. The only point of redundancy we see is in the description of classes, especially in terms of Java Servlets."*

When asking further insights about this comment the team explained that, especially in the *New Feature* category, the class descriptions provided by ARENA are quite similar. The cause of such a situation is that several of the new classes implemented in the OM releases are Java `Servlets`, all summarized by ARENA with a common pattern, *i.e., New class* `ClassName` *extending* `HttpServlet`*. This boundary class communicates mainly with* `HttpServletRequest`, `HttpServletResponse`, *and* `HttpSession`. Since OM has been built from scratch during the six months of our study, each one of its releases contained several new classes (including `Servlets`), leading to this redundancy observed by the developers. Note that these descriptions are independently generated for every class based on their particular implementation, which suggests a common design and structure in this case. Future work on ARENA will focus on grouping structurally similar classes to provide a unique summary for them.

TABLE 10
Information added/deleted/modified before issuing the OM release note as official documentation.

Change Type	OM Release			
	0.1	0.2	0.3	0.4
Information Added	No	No	No	Yes
Information Deleted	No	No	No	No
Information Modified	Yes	Yes	Yes	Yes

Finally, the developers *strongly agreed* (releases 0.1, 0.2, and 0.3) and *weakly agreed* (0.4) on the usefulness of the ARENA release note to a developer in charge of evolving the software system. As explained by the team in the unstructured interview, the latter evaluation (*i.e.,* the *weakly agreed* to the usefulness of the ARENA's release note for OM 0.4) was due to the missing rationale for the change related to the `OM-45` issue, previously described (and reported in Fig. 9).

Table 10 reports on the ARENA's release notes in which the developers added, deleted, and modified information before issuing them as part of the official documentation of the OM project. In the release 0.4, the team documented the rationale behind the changes implemented for the `OM-45` issue, thus adding new information to the ARENA release note. As previously said, the documentation of the rationale behind the applied changes is something we plan to explore in the future. In the other three release notes, no information was added.

No information generated by ARENA was deleted from any of the release notes. This confirms the results achieved in *Study II* providing further empirical evidence on the correctness and importance of the items reported in the release notes generated by ARENA. In all four release notes developers merged together the description of all the servlets that were added to implement a specific feature, as a single point in the release note. For example, a description like:

- OM-31: Implement the kit management subsystem
  - New class `ServletEditKitsAP` extending `HttpServlet`. This boundary class communicates mainly with `HttpServletRequest`,

`HttpServletResponse`, and `HttpSession`.
  - New class `ServletShowKitsAP` extending `HttpServlet`. This boundary class communicates mainly with `HttpServletRequest`, `HttpServletResponse`, and `HttpSession`.

was converted into:

- OM-31: Implement the kit management subsystem
  - New classes `ServletEditKitsAP`, `ServletShowKitsAP`, both extending `HttpServlet`. These boundary classes communicate mainly with `HttpServletRequest`, `HttpServletResponse`, and `HttpSession`.

The goal of this modification was to reduce the level of redundancy in the official release notes. We are working on a similar mechanism that allows ARENA to group classes having a similar description.

Finally, during the final semi-structured interview, all three developers expressed their support to systematically adopting ARENA in their company. In the team leader's own words:

> "*ARENA helps to save time, especially when you* ***need*** *to create release notes and you cannot really allocate too much time on such a task. While some improvements are still needed, this version of ARENA is already enough to automate most of the work needed in the creation of a release note.*"

When asked about the three major strengths and weaknesses of ARENA, the three developers agreed on indicating the *completeness of the captured information*, *full automation*, and *high level of detail in the changes' description* as the three main strengths of the ARENA tool. On the other hand, they indicated as major weaknesses: the *missing rationale behind some changes*, the *limitations of the code summarization technique* (*i.e.,* the ones leading to the presence of some level of redundancy in the ARENA release notes), and the *impossibility to personalize the ARENA release notes* (*e.g.,* by choosing which information to include before running the release notes generation). These three aspects will guide our work on the next release of ARENA.

Summary of Study IV. The OM developers support the adoption of ARENA in their company, indicating the usefulness of our tool. They added very little information to the generated release notes (only in one note)–(**RQ**$_3$)– and did not delete any information (**RQ**$_2$). The OM developers found the release note generated by ARENA to be complete and precise (**RQ**$_1$), with the few exceptions described above. Finally, they suggested improvements in ARENA, devoted to unify description of (changes to) similar classes, and to incorporate in the release note a summary of developers' discussion on changes (**RQ**$_3$).

4.5 Summary of the Evaluations

The main findings of the four empirical studies can be summarized as follows:

- **Study I—Completeness.** The ARENA release notes capture most of the items from the original release notes (86% on average). Furthermore, 88% of the evaluators agree that the additional information extracted by ARENA should be included in the release notes.
- **Study II—Importance.** Most information (82%) included in the ARENA release notes and missing in the original release notes is considered important or very important by developers. Also, the information considered as important in the original release notes is captured by ARENA, with the only exception of *Major changes* category, which we plan to include in the future by prioritizing changes.
- **Study III—Study with Industrial Developers.** Developers judged the ARENA release note as more complete, useful, and precise than the one manually created by the team leader.
- **Study IV—Six-month In-field Study.** The developers support the adoption of ARENA in their company, indicating the usefulness of our tool. They found the release notes generated by ARENA mostly complete and precise.

5 Threats to Validity

This section describes the threats to validity of the overall ARENA evaluation, highlighting threats that arose in specific studies and the extent to which such threats were mitigated in subsequent studies.

Threats to *construct validity* concern the relationship between theory and observation. In this work such threats mainly concern how the generated release notes were evaluated. For *Study I* and *Study II*, we tried to limit the subjectiveness in the answers by asking respondents to compare the contents of a generated release note with that of the actual one. In *Study I* we assigned each release note to two independent evaluators. For *Study III* and *Study IV*, although our main aim was to collect qualitative insights, we still tried to collect objective information by (i) involving multiple evaluators, and (ii) using an appropriate questionnaire with answers in a Likert scale, complemented with additional comments.

Threats to *internal validity* concern factors that could have influenced our results. In *Study I* and *Study III* we tried to limit the evaluators' bias by not telling them upfront which were the original and automatically generated release notes. Instead, this was not done in *Study II*, which purposely aimed at determining the possible gaps or redundant information in ARENA release notes with respect to the original one. Another possible threat is that the participants in *Study I* and *Study II* have a different level of knowledge of the object projects. We must note that some of the respondents in *Study II* were developers of the object projects, that their answers were in line with the answers of the other participants, and that we also reported a detailed, separate analysis limited to these participants. None of the participants in *Study I* were developers/contributors of the object projects. In *Study III* we asked developers to perform the comparative evaluation only after having provided a first assessment of the automatically generated release note. This allowed us to gain both an absolute and a relative assessment.

Threats to *external validity* concern the generalizability of our findings. In terms of *evaluators*, the paper reports results concerning the evaluation of release notes from the perspective of potential end-users/integrators (*Study I* and *Study II*) and of developers/maintainers (*Study II*, *Study III*, and *Study IV*). In terms of *objects*, across all studies, release notes from 15 different releases of eight different projects were generated and evaluated. Such objects comprised open-source libraries (*i.e.,* the Apache projects for the first two studies), a management system (SMOS) and a proprietary embedded system (OM). In terms of *participants*, the study involved informed outsiders (*Study I*), developers not involved in the projects for which they evaluated release notes (part of of *Study II* participants), as well as original developers of open-source and proprietary projects (some of the *Study II* participants, as well as all *Study III* and *Study IV* participants). Having said that, we recognize the usefulness of a further evaluation of ARENA on other projects and within different kinds of organizations. Also for this aim we have made ARENA available.

6 Related Work

Little research has been done on release notes or their automatic generation. Recently, Abebe *et al.* [1] conducted a study on the content of 85 release notes from 15 software systems. In this study, *title*, *system overview*, *resource requirements*, *installation*, *addressed issues* (*i.e.,* new features, bug fixes, and improvements), and *caveats* were identified as information types contained in release notes. An additional study on eight of the initial release notes revealed that different factors related to an issue, such as, its *type*, *number of comments*, *description size*, *days to be addressed*, *number of modified files*, and *reporter's experience*, potentially influencing the likelihood of an issue to be appear in a release note. Considering these factors, Abebe *et al.* proposed four predictors based on machine learning models to automatically identify issues to be listed in release notes, and found random forest being the one achieving the best performance. Differently from Abebe *et al.*'s work, our exploratory study covers 990 release notes from 55 software systems and analyzes types of information at a finer granularity level, which (in one way or another) overlap with the kind of information studied by Abebe *et al.* (*e.g.,* the *migration instructions* from our study could be regarded as Abebe *et al.*'s *installation-related information*). Moreover, ARENA is not limited to list issues extracted from the issue tracker, but also links them to summarized structural changes extracted from the versioning system. This not only includes the use of source code summarization but, as detailed in Section 3, specific approaches to summarize various kinds of changes, *e.g.,* to documentation, licenses, or libraries. Finally, it is worth noticing that ARENA and the work by Abebe *et al.* [1] are rather complementary than competing techniques. The goal of our tool is to extract as much information as possible concerning the issued release and organize it in order to limit redundancy and information overload. This is implemented via the expand/collapse mechanism, showing information at different granularity level. None of the information extracted from the issue tracker is excluded from the release note, since we tried to increase as much as possible the completeness of the ARENA release notes. The results of the reported evaluations show the appreciation of such a feature by developers. The work by Abebe *et al.* [1] tries instead to discriminate between issues to show/not show in the release note on the basis of specific characteristics (*e.g.,* type, number of comments, description size, *etc.*). Such a technique, properly extended to work not only with information from the issue tracker but with all categories of changes captured by ARENA, could be integrated in future releases of our tool as an option to highlight the major changes of a system in the release note.

Regarding the automatic extraction and description of system-level changes occurring between two subsequent versions of a software system, some research has been conducted to summarize changes occurring at a smaller granularity [8], [13], [34]. Buse and Weimer proposed *DeltaDoc* [8], a technique to generate a human-readable text describing the behavioral changes caused by modifications to the body of a method. Such an approach, however, does not capture why the changes were performed. In this sense, Cortes-Coy *et al.* [13] proposed *ChangeScribe*, a technique that describes a given set of source code changes based on its stereotype, type of changes and the impact set of the changes. Similarly, Rastkar and Murphy [34] proposed a machine learning-based technique to extract the content related to a code change from a set of documents (*e.g.,* bug reports or commit messages). Differently from our approach, these summaries focus on describing fine-grained changes at commit and method level, respectively. ARENA is meant to identify and summarize related sets of structural changes at the system level and, where possible, link them to their motivation.

The automatic generation of release notes relies on extracting change information from software repositories and issue tracking systems. More research work has been done in this area, especially in the context of software evolution [21]. Related to our approach is the work on traceability link recovery between issues reported in issue trackers and changes in versioning systems [5], [12], [15], [39]. While ARENA employs similar techniques to extract change information, none of these approaches attempted to produce a natural language description of the code changes linked to the reports.

Concerning the summarization of other software artifacts, different approaches have been proposed to automatically summarize bug reports [23], [24], [35]. The focus of such approaches is on identifying and extracting the most relevant sentences of bug reports by using supervised learning techniques [35], network analysis [23], [24], and information retrieval [24]. These summaries are meant to reduce the content of bug reports. In a different way, the bug report summaries included in the ARENA release notes are meant to describe the changes occurred in the code of the software systems in response to such reports. At source code level, the automatic summarization research has focused on describing OO artifacts, such as, classes and methods by generating either term-based [14], [18], [26], [27], [37] or text-based [28], [38] summaries. ARENA uses the approach proposed by Moreno *et al.* [28] for generating text-based descriptions of the classes added in the new version of the software.

7 Conclusion and Future Work

This paper described ARENA, a technique that combines in a unique way source code analysis, summarization techniques, and information mined from software repositories to automatically generate complete release notes. ARENA has been evaluated in four different studies, aimed at assessing the completeness, importance, and usefulness of ARENA releases notes, as well as letting developers use ARENA within an ongoing project. The results obtained from the four studies indicated that:

1) the ARENA release notes provide important content that is not explicit or is missing in manual release notes, as considered by professional and open-source developers;
2) the ARENA release notes include more complete and precise information than the manually-produced ones; and
3) the extra information included by ARENA makes its release notes to be considered more useful.

Based on the work done and the results obtained in our studies, the research on release note generation moves into a new arena, where additional research questions can be investigated, such as: *What is the most important information to include in the release notes and how to classify it? How should release notes be presented to users?* These are just two items on our future research agenda.

ARENA is currently implemented to work with Java-based systems, using *Jira* as issue tracker and *git* as versioning system. Adapting it to work with software systems implemented in other OO languages, using other issue trackers and versioning systems is an additional engineering effort, which we plan to undertake as future work. We will also introduce in ARENA features that have been suggested by participants to our studies, *e.g.*, integrating a summary of issue discussions and grouping together changes to related/similar classes. Finally, we will investigate a possible integration with the approach proposed by Abebe *et al.* [1], with the goal of highlighting in the release note the most important changes brought by the new release.

Acknowledgments

The authors would like to thank all the participants of our four studies. Laura Moreno and Andrian Marcus are supported in part by grants from the National Science Foundation (CCF-1017263 and CCF-0845706). Gerardo Canfora and Massimiliano Di Penta are partially supported by the Markos project, funded by the European Commission under Contract Number FP7-317743.

References

[1] S. Abebe, N. Ali, and A. Hassan, "An empirical study of software release notes," *Empirical Software Engineering*, pp. 1–36, 2015.

[2] G. Antoniol, K. Ayari, M. Di Penta, F. Khomh, and Y.-G. Guéhéneuc, "Is it a bug or an enhancement?: a text-based approach to classify change requests," in *Proceedings of the 2008 conference of the Centre for Advanced Studies on Collaborative Research, October 27-30, 2008, Richmond Hill, Ontario, Canada*, 2008, p. 23.

[3] G. Antoniol, M. Di Penta, and E. Merlo, "An automatic approach to identify class evolution discontinuities," in *7th International Workshop on Principles of Software Evolution (IWPSE 2004), 6-7 September 2004, Kyoto, Japan*. IEEE Computer Society, 2004, pp. 31–40.

[4] A. Bacchelli, M. Lanza, and R. Robbes, "Linking e-mails and source code artifacts," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1, ICSE 2010, Cape Town, South Africa, 1-8 May 2010*. ACM, 2010, pp. 375–384.

[5] A. Bachmann, C. Bird, F. Rahman, P. T. Devanbu, and A. Bernstein, "The missing links: bugs and bug-fix commits," in *Proceedings of the 18th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2010, Santa Fe, NM, USA, November 7-11, 2010*. ACM, 2010, pp. 97–106.

[6] R. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval*. Addison-Wesley, 1999.

[7] C. Bird, A. Gourley, P. T. Devanbu, M. Gertz, and A. Swaminathan, "Mining email social networks," in *Proceedings of the 2006 International Workshop on Mining Software Repositories, MSR 2006, Shanghai, China, May 22-23, 2006*. ACM, 2006, pp. 137–143.

[8] R. P. Buse and W. R. Weimer, "Automatically documenting program changes," in *Proceedings of the IEEE/ACM international conference on Automated software engineering*, ser. ASE '10. New York, NY, USA: ACM, 2010, pp. 33–42.

[9] G. Canfora, L. Cerulo, M. Cimitile, and M. Di Penta, "Social interactions around cross-system bug fixings: the case of FreeBSD and OpenBSD," in *Proceedings of the 8th International Working Conference on Mining Software Repositories, MSR 2011, Waikiki, Honolulu, HI, USA, May 21-28, 2011*. IEEE, 2011, pp. 143–152.

[10] ——, "How changes affect software entropy: an empirical study," *Empirical Software Engineering*, 2012.

[11] M. L. Collard, H. H. Kagdi, and J. I. Maletic, "An XML-based lightweight C++ fact extractor," in *11th International Workshop on Program Comprehension (IWPC 2003), May 10-11, 2003, Portland, Oregon, USA*. IEEE Computer Society, 2003, pp. 134–143.

[12] C. S. Corley, N. A. Kraft, L. H. Etzkorn, and S. K. Lukins, "Recovering traceability links between source code and fixed bugs via patch analysis," in *Proceedings of the 6th International Workshop on Traceability in Emerging Forms of Software Engineering*, ser. TEFSE '11. New York, NY, USA: ACM, 2011, pp. 31–37.

[13] L. Cortes-Coy, M. Linares-Vasquez, J. Aponte, and D. Poshyvanyk, "On automatically generating commit messages via summarization of source code changes," in *Source Code Analysis and Manipulation (SCAM), 2014 IEEE 14th International Working Conference on*, Sept 2014, pp. 275–284.

[14] A. De Lucia, M. Di Penta, R. Oliveto, A. Panichella, and S. Panichella, "Using ir methods for labeling source code artifacts: Is it worthwhile?" in *Program Comprehension (ICPC), 2012 IEEE 20th International Conference on*, June 2012, pp. 193–202.

[15] M. Fischer, M. Pinzger, and H. Gall, "Populating a release history database from version control and bug tracking systems," in *19th International Conference on Software Maintenance (ICSM 2003), The Architecture of Existing Systems, 22-26 September 2003, Amsterdam, The Netherlands*. IEEE Computer Society, 2003, pp. 23–.

[16] D. M. Germán, Y. Manabe, and K. Inoue, "A sentence-matching method for automatic license identification of source code files," in *ASE 2010, 25th IEEE/ACM International Conference on Automated Software Engineering, Antwerp, Belgium, September 20-24, 2010*. ACM, 2010, pp. 437–446.

[17] M. W. Godfrey and L. Zou, "Using origin analysis to detect merging and splitting of source code entities," *IEEE Trans. Software Eng.*, vol. 31, no. 2, pp. 166–181, 2005.

[18] S. Haiduc, J. Aponte, L. Moreno, and A. Marcus, "On the use of automated text summarization techniques for summarizing source code," in *Proceedings of 17th IEEE Working Conference on Reverese Engineering*. Beverly, MA: IEEE CS Press, 2010, pp. 35–44.

[19] A. Hassan and R. Holt, "Architecture recovery of web applications," in *Software Engineering, 2002. ICSE 2002. Proceedings of the 24rd International Conference on*, 2002, pp. 349–359.

[20] K. Herzig, S. Just, and A. Zeller, "It's not a bug, it's a feature: how misclassification impacts bug prediction," in *35th International Conference on Software Engineering, ICSE '13, San Francisco, CA, USA, May 18-26, 2013*. IEEE Computer Society, 2013, pp. 392–401.

[21] H. Kagdi, M. L. Collard, and J. I. Maletic, "A survey and taxonomy of approaches for mining software repositories in the context of software evolution," *J. Softw. Maint. Evol.*, vol. 19, no. 2, pp. 77–131, Mar. 2007.

[22] R. Koschke, "Atomic architectural component recovery for program understanding and evolution," in *18th International Conference on Software Maintenance (ICSM 2002), Maintaining Distributed Heterogeneous Systems, 3-6 October 2002, Montreal, Quebec, Canada*. IEEE Computer Society, 2002, pp. 478–481.

[23] R. Lotufo, Z. Malik, and K. Czarnecki, "Modelling the 'hurried' bug report reading process to summarize bug reports," in *28th IEEE International Conference on Software Maintenance, ICSM 2012,*

Riva del Garda, Trento, Italy, September 23-28, 2012. IEEE Computer Society, 2012, pp. 430–439.

[24] S. Mani, R. Catherine, V. S. Sinha, and A. Dubey, "Ausum: approach for unsupervised bug report summarization," in *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, ser. FSE '12. New York, NY, USA: ACM, 2012, pp. 11:1–11:11.

[25] O. Maqbool and H. A. Babri, "Hierarchical clustering for software architecture recovery," *IEEE Transactions on Software Engineering*, vol. 33, no. 11, pp. 759–780, 2007.

[26] P. W. McBurney, C. Liu, C. McMillan, and T. Weninger, "Improving topic model source code summarization," in *Proceedings of the 22Nd International Conference on Program Comprehension*, ser. ICPC 2014. New York, NY, USA: ACM, 2014, pp. 291–294.

[27] P. W. McBurney and C. McMillan, "Automatic documentation generation via source code summarization of method context," in *Proceedings of the 22Nd International Conference on Program Comprehension*, ser. ICPC 2014. New York, NY, USA: ACM, 2014, pp. 279–290.

[28] L. Moreno, J. Aponte, G. Sridhara, A. Marcus, L. Pollock, and K. Vijay-Shanker, "Automatic generation of natural language summaries for Java classes," in *Proceedings of the IEEE International Conference on Program Comprehension*, ser. ICPC '13. IEEE, 2013, pp. 23–32.

[29] L. Moreno, G. Bavota, M. Di Penta, R. Oliveto, A. Marcus, and G. Canfora, "Automatic generation of release notes," in *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE 2014. New York, NY, USA: ACM, 2014, pp. 484–495.

[30] L. Moreno, A. Marcus, L. Pollock, and K. Vijay-Shanker, "Jsummarizer: An automatic generator of natural language summaries for java classes," in *Proceedings of the IEEE International Conference on Program Comprehension, Formal Tool Demonstration*, ser. ICPC '13. IEEE, 2013, pp. 230–232.

[31] A. N. Oppenheim, *Questionnaire Design, Interviewing and Attitude Measurement*. London: Pinter, 1992.

[32] M. F. Porter, "An algorithm for suffix stripping," *Program*, vol. 14, no. 3, pp. 130–137, 1980.

[33] K. Prete, N. Rachatasumrit, N. Sudan, and M. Kim, "Template-based reconstruction of complex refactorings," in *26th IEEE International Conference on Software Maintenance (ICSM 2010), September 12-18, 2010, Timisoara, Romania*. IEEE Computer Society, 2010, pp. 1–10.

[34] S. Rastkar and G. C. Murphy, "Why did this code change?" in *Proceedings of the 2013 International Conference on Software Engineering*, ser. ICSE '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 1193–1196.

[35] S. Rastkar, G. C. Murphy, and G. Murray, "Summarizing software artifacts: a case study of bug reports," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ser. ICSE '10. New York, NY, USA: ACM, 2010, pp. 505–514.

[36] J. Ratzinger, T. Sigmund, and H. Gall, "On the relation of refactorings and software defect prediction," in *Proceedings of the 2008 International Working Conference on Mining Software Repositories, MSR 2008, Leipzig, Germany, May 10-11, 2008*. ACM, 2008, pp. 35–38.

[37] P. Rodeghero, C. McMillan, P. W. McBurney, N. Bosch, and S. D'Mello, "Improving automated source code summarization via an eye-tracking study of programmers," in *Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE 2014. New York, NY, USA: ACM, 2014, pp. 390–401.

[38] G. Sridhara, E. Hill, D. Muppaneni, L. Pollock, and K. Vijay-Shanker, "Towards automatically generating summary comments for Java methods," in *Proceedings of the IEEE/ACM international conference on Automated software engineering*, ser. ASE '10. New York, NY, USA: ACM, 2010, pp. 43–52.

[39] J. Wu, A. E. Hassan, and R. C. Holt, "Comparison of clustering algorithms in the context of software evolution," in *Proceedings of 21st IEEE International Conference on Software Maintenance*. Budapest, Hungary: IEEE CS Press, 2005, pp. 525–535.

[40] R. Wu, H. Zhang, S. Kim, and S.-C. Cheung, "ReLink: recovering links between bugs and changes," in *SIGSOFT/FSE'11 19th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE-19) and ESEC'11: 13rd European Software Engineering Conference (ESEC-13), Szeged, Hungary, September 5-9, 2011*. ACM, 2011, pp. 15–25.