A Study of Vulnerability Identifiers in Code Comments: Source, Purpose, and Severity

Yusuf Sulistyo Nugroho*, *Member, IEEE,* Dedi Gunawan, *Member, IEEE,* Devi Afriyantari Puspa Putri, Syful Islam*, and Abdulaziz Alhefdhi

Original scientific article

***Abstract*—Software vulnerability is one of the weaknesses in computer security that challenges developers to rectify. Software maintainers rely on code comments to maintain their source code, including fixing vulnerability issues. To facilitate understanding the security issues in the related code, vulnerability identifiers are commonly included in code comments. However, not all vulnerability-related code comments describe clearly the purposes of the inclusion of the identifiers. Based on this evidence, we investigate the importance of vulnerability identifiers contained in source code comments, which is the novelty of this paper. We performed a study of 1,491 code comments that refer to vulnerability identifiers to define their categories. We then applied a mixed-method approach to classifying the types of the related repository and code, the rationale of identifier references, and the severity level of vulnerabilities in the code. The results indicate that vulnerability identifiers in code comments are useful to notify security issues for the related source code, and our study widens up chances for future work to further investigate these problems.**

***Index Terms*—code comments, identifier, vulnerability.**

I. Introduction

Software systems and their vulnerabilities have become a widespread concern for developers during software development [1] - [4]. As one of the weaknesses in computer security, vulnerability is used by a threat actor to exploit the system to perform unauthorized actions [5], [6]. The high-skilled attacker may take benefit from the vulnerability to steal the users' personal information, run the systems remotely, terminate the systems, or even damage the systems.

Investigating and fixing software vulnerabilities are important tasks in the software development process. Several studies have been conducted to identify and prevent vulnerabilities. A study on the impact of security issues in the npm package shows the results of vulnerability discovery, fixing procedure of vulnerabilities in the package and vulnerability effects to the other packages in the ecosystem [7]. Based on the findings, the authors built a guideline for package maintainers and tool developers to improve the process of dealing with security issues. Another study [8] proposed a framework to automatically describe and predict vulnerabilities with high accuracy. The results also demonstrated that they explored the development of vulnerabilities in specific products.

In most cases, software maintainers rely on source code comments when updating the source code [9]. Since source code comments store valuable information of software engineering tasks, many works have made use of code comments in their studies. For example, source code comments have been found to document personal and team tasks [10], indicate technical debt [11] - [13], code linkage [14], and publication citations [15]. Several qualitative studies on code comments have also been undertaken. A survey on prior studies that have made use of source code comments reports that code comments can assist programmers to comprehend the code and have a particular impact on software development and maintenance [16]. Another study on code comments in different Java projects [17] reveals that the common purpose of developers writing comments in the source code is to provide communication between software developers.

Code comments are also used to indicate vulnerabilities in the related code by referencing vulnerability identifiers. For example, in the following code comment, the `CVE-2013-4238` is referred to indicate vulnerability in the project source code.[1]

```
# GENERAL_NAME_print() doesn't handle NULL bytes in
# ASN1_string correctly, CVE-2013-4238
```

CVE is Common Vulnerabilities and Exposures identification number that describes cybersecurity product and service. Although many works have shown the benefits and general reasons for writing comments, to the best of our knowledge, a comprehensive analysis of the role of vulnerability identifiers in source code comments has never been undertaken.

In this paper, we investigate the importance of vulnerability identifiers inclusion in source code comments, which is the novelty of this work. To achieve this goal, we studied 1,491 distinct vulnerability-related code comments from 32,007 GitHub repositories across 7 popular programming languages. Initially, we performed a quantitative analysis of the extracted vulnerability-related code comments. We then qualitatively and empirically studied the characteristics of the identifiers, code, and the severity of code.

Manuscript received October 19, 2021; revised February 18, 2022. Date of publication May 24, 2022. Date of current version May 24, 2022. The associate editor prof. Tihana Galinac Grbac has been coordinating the review of this manuscript and approved it for publication.

Y. S. Nugroho, D. Gunawan, and D. A. P. Putri are with the Universitas Muhammadiyah Surakarta, Indonesia. S. Islam is with the Noakhali Science and Technology University, Bangladesh. A. Alhefdhi is with the Prince Sattam Bin Abdulaziz University, Saudi Arabia.

E-mails: {yusuf.nugroho, dedi.gunawan, deviapputri}@ums.ac.id, syfulcste@nstu.edu.bd, a.alhefdhi@psau.edu.sa.

Digital Object Identifier (DOI): 10.24138/jcomss-2021-0124

* Corresponding authors

[1]https://github.com/mozillazg/pypy/blob/981c7a7748b15b7a68a714426d8454d428520109/pypy/module/_ssl/interp_ssl.py#L891

The results of our study show that the CVE (Common Vulnerabilities and Exposures) are the most frequent vulnerability identifiers cited in code comments. They are mentioned in a little less than 50% of the analyzed GitHub repositories that are categorized as software projects. The inclusions of vulnerability identifiers are mainly for notifying workaround and warning reasons of the vulnerabilities that are contained in multiple statements of the main source code. Furthermore, the referenced CVE identifiers in code comments commonly have critical and high severity where they have been included in the comments for more than 2 years.

In sum, the contributions of this paper are as follows:

- A comprehensive study of vulnerability-related source code comments from 32,007 GitHub repositories across 7 programming languages.
- A mixed-method approach to qualitatively analyze the inclusion of vulnerability identifiers in source code comments.
- A set of implication and recommendation for both software developers and researchers.

The rest of the paper is organized as follows. Section II describes the related work. Section III presents our study settings to conduct the research. In detail, we explain the research questions, data collection process, and online appendix. The results of the study and their interpretations are described in Section IV. Section V and VI present the implication of our study and threats to validity, respectively. Finally, we conclude this paper in Section VII.

II. Related Work

In this section, we present the works related to the analysis of vulnerability identifiers in source code comments.

A. Software Vulnerability

A number of studies on software vulnerability have been conducted in different systems, such as Android-based apps [18], Windows operating system [19], and open source systems [20]. Linares-Vásquez et.al [18] presents a large-scale study on Android-related vulnerabilities. In the study, the authors proposed a detailed taxonomy of the types of Android-related vulnerabilities, and investigated the affect and survivability of the vulnerabilities to assist developers in verifying and validating mobile apps. Guo et al. [19] proposed a vulnerability detection technique in Windows systems by comparing the security patches. In the other work, Jimenez et al. [20] investigated the effectiveness of three prior vulnerability prediction techniques in three open source systems (Linux Kernel, OpenSSL, and Wireshark). To complement prior studies, we investigated vulnerability-contained code comments from 32,007 open source projects across 7 programming languages extracted from GitHub repositories.

Several studies on software vulnerabilities have used CVE-related databases for their reference. A study by Han et al. [21] extracted a large amount of vulnerability data from the CVE Database to build a deep learning-based technique to predict the severity level of vulnerability. In comparison with previous works, besides referencing to CVE and NVD databases, we enhanced our work by also referencing the CERT/CC database.

B. Source Code Comments

Many researchers have analyzed source code comments in their works. Pascarella et al. [22] built an automated classification of code comments to the defined taxonomy. Fluri et al. [23] investigated the evolution of source code comments to understand whether developers comment their code and to what extent they add comments or adapt them when the code is changed. Source code comments are also used in prior works to develop an automated classifier on self-admitted technical debt [24], [25], and a new approach to detect fragile code comments [26].

Haouari et al. [17] explored the habits of developers in writing code comments and analyzed the existing comments in different open source Java projects. They found that the most important purposes of code comments are to communicate between code authors or to note future changes.

The linkage between source code and external sources for software development is another related topic. Hata et al. [14] investigated the role of links included in source code comments. Their findings report that links in source code comments suffer from decay, insufficient versioning when the link targets evolve, and lack of bidirectional traceability.

In comparison with previous works, we have also made use of the source code comments as our main dataset. However, none of the related work provides a comprehensive analysis of the importance of vulnerability identifiers in source code comments, which is the objective of this paper.

III. Research Method

The purpose of this research is to investigate the importance of vulnerability identifiers contained in source code comments. In the following, we describe our methodology to conduct the research. In details, we define the research questions, data collection procedure, and present an online appendix.

A. Research Questions

To guide the study, we formulate the following research questions with their motivations.

RQ_1: What is the characteristic of vulnerability-related identifier in code comments?

- *$RQ_{1.1}$: Is the repository a software development project?*
  Motivation: The motivation of $RQ_{1.1}$ is to exclude the non-software development project repositories from our study to reduce bias in our analysis. With this exclusion process, we only focus on the code comments from software development projects.
- *$RQ_{1.2}$: Does the source code comment relate to vulnerability?*
  Motivation: Similar to $RQ_{1.1}$, the motivation of $RQ_{1.2}$ is to reduce bias of the code comments investigation. In this research question, we aim to classify whether the vulnerability identifiers inclusion in code comments relate to vulnerability. We found that some source code comments reference a vulnerability identifier only for motivating example.
- *$RQ_{1.3}$:What are the purposes of vulnerability identifier inclusion in source code comments?*

Motivation: The key motivation of this research question is to manually investigate the reasons why developers put vulnerability identifiers in the source code comments.

RQ_2: What kind of code is described in the vulnerability-related code comments?

- *$RQ_{2.1}$: Is the vulnerability-related code source code or test code?*
  Motivation: Since GitHub software projects may separate their main code and the code for testing, this research question aims to analyze whether the vulnerability identifiers are included in the source code or test code.
- *$RQ_{2.2}$: Where is the actual vulnerability with relation to the comment?*
  Motivation: This research question aims to identify whether the vulnerability-related code consists of single or multi statements. This is important to show developers the most frequent actual location of the vulnerability to address the issues accurately.

RQ_3: How vulnerable is the code in GitHub repositories?

- *$RQ_{3.1}$: How severe is the vulnerability impact on the code?*
  Motivation: The severity level of vulnerability indicates the measurement of the vulnerability impact in their code. Thus, in this research question, we investigate the severity levels based on the most frequent identifiers included in the code comments to understand the severity of the source code.
- *$RQ_{3.2}$: How long does the vulnerability reside in the code?*
  Motivation: The key motivation of *$RQ_{3.2}$* is to analyze how long the vulnerability identifiers have been referenced in the code comments.

B. Data Collection

We now describe our methods for identifier definition and the extraction of source code comments.

1) Identifiers Definition: To define the vulnerability identifiers, we initially extracted all identifiers from 2 popular vulnerability databases, that is, National Vulnerability Database (NVD)[2] and Coordination Center of the Computer Emergency Response Team (CERT/CC) Vulnerability Notes Database.[3] From the extraction, we found 4 unique vulnerability identifiers plus 1 additional keyword to indicate the vulnerability scoring system identified in both databases, that is:

- *CVE id (Common Vulnerabilities and Exposures identification number)* describes cybersecurity product and service.
- *CWE id (Common Weakness Enumeration specification)* represents a vulnerability type.
- *CPE Name (Common Platform Enumeration)* is a structured naming scheme for information technology systems, software, and packages.
- *VU Notes* describes a vulnerability issue that include summaries, technical details, remediation information, and lists of affected vendors.
- *CVSS (Common Vulnerability Scoring System)* is an open framework for communicating the characteristics and severity of software vulnerabilities.

[2]https://nvd.nist.gov/

[3]https://www.kb.cert.org/vuls/

2) Source Code Comments Extraction: In our study, the source code comments were extracted using the same procedure as prior work [14]. We extracted the code comments from 32,007 GitHub repositories across 7 languages, that are, C, C++, Java, JavaScript, PHP, Python, and Ruby (on August 10, 2020). We selected these languages since they were ranked consistently in the top 10 languages on GitHub between 2014 and 2019 (based on the number of pull requests, pushes, stars, and issues) [27]. To extract the related code comments, we applied the following regular expression of our 4 defined vulnerability identifiers and 1 additional keyword,

```
(?:CVE-[0-9]{4}-[0-9]{4,}|cpe:2.[23]:\w:\w+:|
CWE-[0-9]+|(?<![\w\d])CVSS(?![\w\d])|VU#[0-9]+)
```

From our extraction, we obtained 6,751 code comments that contain at least one identifier. After we removed the duplication, we ended at 1,491 distinct comments. We then categorized the comments specifically based on the types of the referenced vulnerability identifiers.

Frequency of vulnerability-related comments. In our collected dataset, we found that a code comment may contain more than one type of vulnerability identifiers. Thus, we categorized the comments into more than one categories. For example, if a comment cites both CVE id and CWE id, we classified the comment as both CVE id-related comment and CWE id-related comment. As shown in Table I, the CVE are the most common vulnerability identifiers that are explicitly referenced in the source code comments, as many as 1,366 (90.58%) comments. This indicates that cybersecurity product and the associated services are common issues faced by software developers. Although the number of code comments is not as many as the CVE identifiers-related comments, CVSS and CWE identifiers are also referenced by software developers to indicate the vulnerabilities in the source code, accounting for 63 (4.18%) and 60 (3.98%) comments, respectively. Finally, there are few source code comments that reference to the VU Notes and the CPE Names, as many as 16 (1.06%) and 3 (0.2%) code comments, respectively.

Identifiers references. We found that a source code comment may contain more than one vulnerability identifier of the same type but different identity numbers. This is indicated by the number of appearances of the 5 defined vulnerability identifiers that is higher than the number of the related code comments. Table I describes that the most dominant identifiers, the CVE identifiers, appear 1,921 times in 1,366 source code comments. The second and third highest identifiers that are referenced in source code comments, CVSS and CWE id are cited as many as 127 and 92 times, respectively. While the VU Notes and CPE Names have been explicitly referenced 22 and 8 times in the code comments.

To get insights about the most programming languages of projects that contain vulnerable code, we counted the number of projects classified by different programming languages. Table II describes that C++ is the most widely used language in the studied projects that include the specified vulnerability

TABLE I
NUMBER OF VULNERABILITY-RELATED CODE COMMENTS AND IDENTIFIERS REFERENCES

identifiers	# related comments		# references
CVE id	1,366	(90.58%)	1,921
CVSS	63	(4.18%)	127
CWE id	60	(3.98%)	92
VU Notes	16	(1.06%)	22
CPE Names	3	(0.20%)	8

TABLE II
FREQUENCY OF PROJECTS BASED ON THE PROGRAMMING LANGUAGE

language	# projects	
C++	144	(40.91%)
Python	65	(18.47%)
Java	55	(15.63%)
PHP	38	(10.80%)
Ruby	34	(9.66%)
JavaScript	15	(4.26%)
C	1	(0.28%)
Total	352	(100%)

identifiers (40.91%), followed by Python and Java, as many as 18.47% and 15.63%, respectively. On the other hand, the least programming language applied by developers that contain vulnerability identifiers is C language, that is 0.28%.

Figure 1 shows that the most common identifiers found in our analysis, the CVE identifiers, are dominantly referenced by software developers who wrote their code in all 7 programming languages, accounting for more than 60% references in the source code comments. Additionally, we found that 100% of the projects developed in C language use CVE identifiers to notify the vulnerabilities. CVSS is the second most vulnerability keyword referenced in projects written in Java, JavaScript, Python, and Ruby languages. On the other side, the second most vulnerability identifier cited in C++ projects is CWE. VU Note and CPE Name are the vulnerability identifiers that are only referenced by C++, Java, and Python projects.

For the statistical evaluation, we find that there is a relationship between programming languages and the vulnerability identifiers. Our null hypothesis on "the programming languages and the vulnerability identifiers are independent" is rejected (i.e., p-value is < 0.001).

C. Appendix

Our online appendix contains a dataset of 1,491 manually analyzed vulnerability-related code comments. The appendix is available at https://github.com/yusufsn/VulnerabilityKeywordsAnalysis.

IV. RESULTS

In this study, we aim to comprehensively understand the vulnerability identifiers in terms of their sources, purposes, and the severity level of the vulnerability that are referenced by software developers in the source code comments. Since our data collection in Section III-B2 shows that CVE identifier is the most vulnerability identifier included in the source code comments, we analyze the severity level of the cited CVE identifiers.

A. RQ_1: What is the characteristic of vulnerability-related identifier in code comments?

Our approach to answer RQ_1 is through manual analysis on 1,491 collected code comments to investigate the category of repositories ($\text{RQ}_{1.1}$), the nature of source code comments ($\text{RQ}_{1.2}$), and to extract the purposes of software developers referencing vulnerability identifiers in the source code comments ($\text{RQ}_{1.3}$).

To answer the three sub-questions in RQ_1, we performed an interactive process of manual coding. In this process, all authors of this paper discussed the initial coding guide for annotating the code comments. Then, the first three authors labeled the first 30 code comments in the sample independently using the specified labels. The kappa agreement is calculated to measure the agreement between three annotators. [4] If the kappa score is more than or equals to 0.61 [28], the annotation task for the remaining data is conducted only by a single annotator.

- *$RQ_{1.1}$: Is the repository a software development project?*

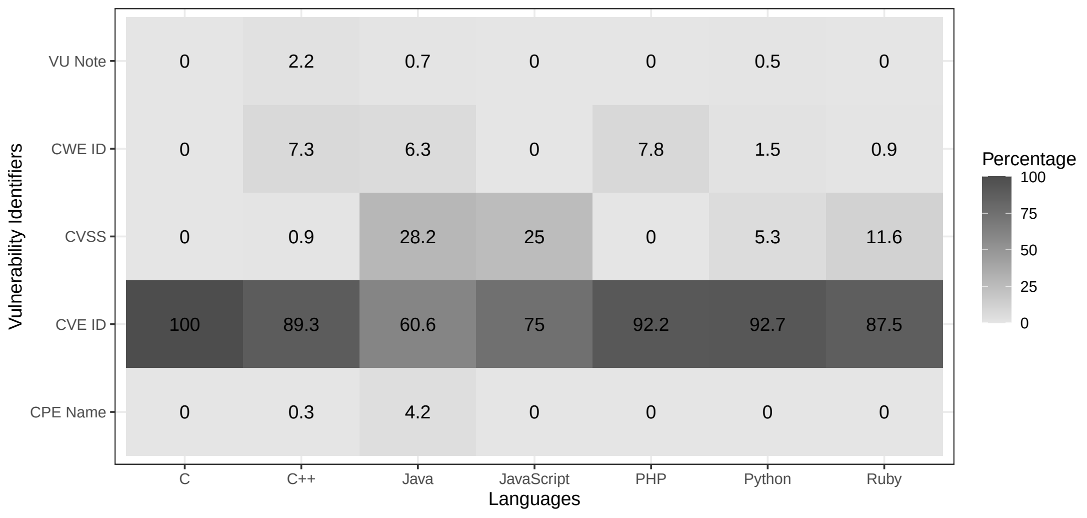
In our collected data, we found that the repositories that include the vulnerability identifiers in the code comments do not always represent software development projects. To reduce bias in our analyses, this research question is implemented to identify the repositories whether they are categorized as software development projects, as classified in a prior study [29]. As described previously in the approach, to categorize the repositories in our data, we identified them manually and calculated the kappa score between three annotators. From the calculation, we obtain 86.67% which describes "almost perfect" [28]. Based on this agreement, the manual labeling for the remaining samples was then undertaken only by the first author.

Results. As illustrated in Figure 2, the result of our manual analysis shows that vulnerability identifiers are mostly referenced in the non-software development projects, as many as 51%. The vulnerability identifiers are also referenced by software developers in their software projects, accounting for 46% of total data. However, only 3% of GitHub repositories in our dataset are inaccessible.

- *$RQ_{1.2}$: Does the source code comment relate to vulnerability?*

To facilitate our investigation to become more valid, we applied a binary classification to identify whether the code comments relate to vulnerability issues. This is because although we applied the vulnerability identifiers to extract code comments from the target repositories, we could not guarantee that the collected comments that contain the identifiers in our dataset always relate to vulnerability. The code comments might describe other things instead of vulnerability-related

[4]http://justusrandolph.net/kappa/



	C	C++	Java	JavaScript	PHP	Python	Ruby
VU Note	0	2.2	0.7	0	0	0.5	0
CWE ID	0	7.3	6.3	0	7.8	1.5	0.9
CVSS	0	0.9	28.2	25	0	5.3	11.6
CVE ID	100	89.3	60.6	75	92.2	92.7	87.5
CPE Name	0	0.3	4.2	0	0	0	0

Fig. 1. Frequency of vulnerability identifier references based on programming languages

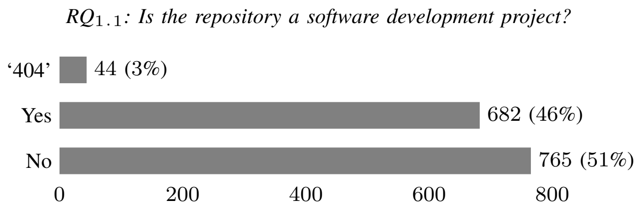


Fig. 2. Distribution of answers to “Is the repository a software development project?”. Initial agreement among the annotators before consolidating the coding schema: 86.67% across the first 30 code comments.

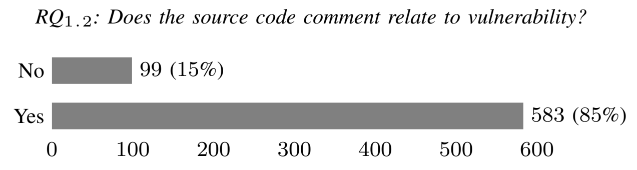


Fig. 3. Distribution of answers to “Does the source code comment relate to vulnerability?” across 682 code comments.

issues, for instance, the CVE id is used as a motivating example.[5]

After removing the non-software development project repositories identified in $\mathrm{RQ}_{1.1}$, we only investigated the code comments from software development projects, as many as 682 repositories. In this analysis, the first three authors of this paper independently performed manual annotation to the first 30 sample code comments and computed the kappa score. The calculation yields 73.33% which describes “substantial” agreement [28]. With this agreement, thus, the remaining sample data were then manually coded by the first author.

Results. Figure 3 describes that in the software project repositories, most code comments that include the vulnerability identifiers relate to vulnerability issues, accounting for 85%. While 15% of code comments do not relate to vulnerability issues. It indicates that the inclusion of the identifiers in code comments is mostly used for developers to notify that the related code has vulnerability issues.

- *$RQ_{1.3}$:What are the purposes of vulnerability identifier inclusion in source code comments?*

Based on the finding in $\mathrm{RQ}_{1.2}$ that shows most code comments relate to vulnerability issues, we then further investigate the identifiers to comprehensively understand the rationale of developers putting the vulnerability identifiers explicitly in source code comments. To achieve this goal, we manually analyzed 583 vulnerability-related code comments classified in $\mathrm{RQ}_{1.2}$. In this analysis, all authors of this paper first discussed the coding guide to identify the purposes of vulnerability identifiers inclusion in code comments.

The following lists describe all 6 codes that emerged from our analysis with their descriptions available in the coding guide:

- *origin*: code is reused from other sources,
- *workaround*: code is needed to address security issues,
- *warning*: notify security issues, recommendation of not use or careful use of the code,
- *see/see-also*: the comment indicates that the identifier points to additional reading material (usually accompanied by a phrase such as “see”, “see also”),
- *keyword-only*: the comment only contains the identifier or link, without further explanation.
- *source code context*: the identifier adds additional information to the source code (use this code for things that do not obviously fit into any of the previous).

The approach to answer $\mathrm{RQ}_{1.3}$ is similar to the two prior RQs. The first three authors of this paper manually investigated the first 30 code comments independently to annotate the purposes of vulnerability identifiers inclusion in code comments based on the defined codes. The agreement score between three annotators is 72.22% which indicates “substantially” agreed [28]. This somewhat lower agreement can be explained by the need to extrapolate the purpose of a vulnerability

[5]Example of non-vulnerability-related comment: https://github.com/ronin-ruby/ronin-exploits/blob/dd25c368ade8d95cdeb5ba2844d4d8d1c4c70a2b/lib/ronin/advisory.rb#L77

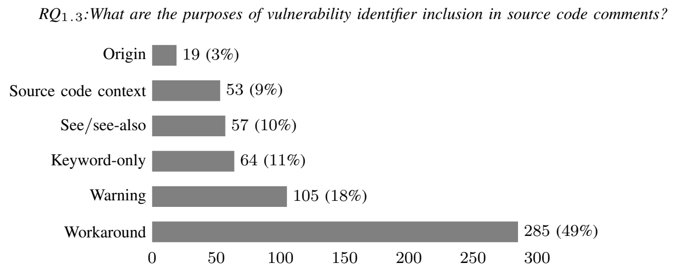


Fig. 4. Distribution of answers to "What are the purposes of vulnerability identifier inclusion in source code comments?" across 583 vulnerability-related code comments.

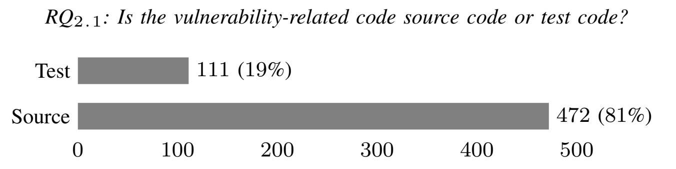


Fig. 5. Distribution of answers to "Is the vulnerability-related code source code or test code?" across 583 code comments.

identifier without being able to interview the code author who added the identifier.

Results. As depicted in Figure 4, the reference of vulnerability identifiers is majorly used as a workaround, accounting for 49%. The inclusion of the identifiers is to indicate that the code is needed to address vulnerability issues. The second most common purpose of developers citing the vulnerability identifiers is used as a warning to notify security issues in the source code, as many as 18%. The developers recommend not to use the code or be careful of using the code. As many as 11% of code comments only contain the vulnerability identifiers, that are used to notify the vulnerability without any further explanation. The other purpose of vulnerability identifiers inclusion in code comments is for additional reading materials to the source code, that is 10% of identifiers present see/see-also and 9% of identifiers represent source code context. Lastly, although the frequency is not as many as the other purposes, code reuse is also described in vulnerability-related code comments, accounting for 3%.

Summary of RQ_1. We found that a little less than 50% of GitHub repositories that refer to vulnerability identifiers in their code comments are software project repositories. There are different purposes for the inclusion of vulnerability identifiers in source code comments, where they are commonly used by the developers as a workaround to indicate that the code is needed to address security issues.

B. RQ_2: What kind of code is described in the vulnerability-related code comments?

Similar to the procedure to answer RQ_1, we conducted a manual analysis to investigate the code that is described by the vulnerability-related code comments. Using 583 vulnerability-related code comments in the data identified from $RQ_{1.2}$, the first three authors of this paper independently performed a manual annotation to the first 30 code comments to categorize the source code ($RQ_{2.1}$) and the location of vulnerability in the code ($RQ_{2.2}$). The kappa score is then calculated to check the level of agreement between the three raters.

- *$RQ_{2.1}$: Is the vulnerability-related code source code or test code?*

In software development projects, developers usually build the code in two types, that is, main source code and test code for software testing. To understand the types of code that commonly relate to the vulnerability issues, in this analysis, we manually categorized the code using the defined labels. The two types of code used in this analysis are described as follows:

- *test*: the code is used for testing, usually the code is placed in the "test" directory in a repository,
- *source*: the primary code used in a project (use this label for any types of code other than test code).

The manual categorization of the first 30 code types between three annotators reaches the kappa score 95.56% or "almost perfect" [28]. Based on this encouraging result, the remaining data was then coded by a single author.

Results. Figure 5 shows the results of the qualitative analysis. The vulnerability identifiers are majorly cited in code comments where the related code are source code in the software project repositories, accounting for 81%. While the other 19% of code in our dataset are classified as test code. Combined with the result of $RQ_{1.3}$, this can be interpreted that the vulnerability identifiers are used to notify and address the security issues in the main code of a software, instead of just for software testing.

- *$RQ_{2.2}$: Where is the actual vulnerability with relation to the comment?*

To comprehensively understand the actual code that might contain the vulnerability, we investigated the location of the code that relates to the code comments. In this qualitative analysis, all authors of this paper initially discussed the common coding guide to define the location of the vulnerable code. Based on the consensus, we specified two codes with a short description, as follows:

- *single*: the vulnerability-related code consists of single statement, although some code written in multiple lines.
- *multiple*: the vulnerability-related code consists of multiple statements.

Using the defined coding guide, the first three authors independently classified the first 30 samples manually to define the location of the code. The calculation of kappa score between the three raters in this iteration yields 77.78% or "substantial" agreement [28]. Based on this score, the first author then manually analyzed the remaining samples.

Results. Figure 6 summarizes the findings of this analysis. Out of 583 code comments which relate to vulnerabilities, 453 (78%) comments describe that the vulnerability issues emerge in the code that consists of multiple statements. While 22% of code comments show that the vulnerability issues are located in a single statement.

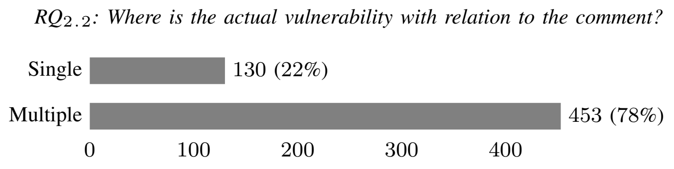


Fig. 6. Distribution of answers to "Where is the actual vulnerability with relation to the comment?" across 583 code comments.

Summary of RQ$_2$. We identified that most vulnerability-related comments describe the vulnerability issues located in the main source code that consists of multiple statements.

C. *RQ*$_3$*: How vulnerable is the code in GitHub repositories?*

Since only CVE identifiers provide the severity level of vulnerability, thus, in this analysis we only focus on code comments that include CVE. To answer the RQ$_3$, we distributed the analyses in 2 sub-research questions, that is, (i) investigating the severity levels of CVE identifiers (RQ$_{3.1}$), and (ii) identifying the duration of CVE identifiers inclusion in the code comments (RQ$_{3.2}$).

- *RQ*$_{3.1}$*: How severe is the vulnerability impact on the code?*

To answer this research question, we classified the severity levels of all 1,921 CVE identifiers that appear in the source code comments, as shown in Table I. According to CVSS score v3.0 in the NVD database, the severity levels of vulnerability are defined into 4 levels: *Low* to represent the least severe vulnerability with the score between 0.1 and 3.9, *Medium* (4.0-6.9), *High* (7.0-8.9), and *Critical*, the most severe vulnerability level in the database with the score ranging from 9.0 to 10.0. [6] For several vulnerabilities, all required information to specify the CVSS score sometimes are not available. This typically happens due to the unwillingness of the software vendors in providing the detailed information relates to the announced vulnerabilities.

Results. By putting aside the "none" severity level, Table III demonstrates that developers commonly included CVE identifiers in their code comments with high severity levels and critical levels, as many as 21,81% and 15,25%, respectively. This indicates that plenty of code in software project repositories needs more attention to handle the vulnerability issues. The number of cited CVE identifiers where their severity level are categorized medium is found as many as 9.47% in the collected code comments. Although it is not as many as the other levels, the low-severity CVE identifiers are also mentioned in code comments during the software development process.

In this RQ, we also identified the most frequent CVE identifiers that are mentioned in source code comments. The affected products and the severity levels of the most common cited CVE ids were extracted from the NVD Database. As described in Table IV, CVE-2016-10033 and CVE-2016-10045 are the most CVE identifiers that are cited by software developers in their code comments, accounting for 1.25% and 1.20%, respectively. The severity level of both CVE identifiers is critical which affects the PHPMailer products. Although the severity levels of CVE-2009-3555, CVE-2012-2459, and CVE-2014-6278 are unavailable according to the NVD Database, these three CVE identifiers are also common to refer to in the source code comments which affect OpenSSL, Bitcoin Core, and Bash, respectively. The other CVE identifiers that are frequently cited in source code comments are CVE-2014-0160 and CVE-2016-7420 with high and medium severity levels, as many as 0.88% and 0.62%, respectively.

TABLE III
FREQUENCY OF CVE IDENTIFIERS REFERENCED IN CODE COMMENTS BASED ON THE SEVERITY LEVELS

Severity levels	# CVE id references	%
High	419	21.81%
Critical	293	15.25%
Medium	182	9.47%
Low	20	1.04%
None	1,007	52.42%
sum	1,921	100%

- *RQ*$_{3.2}$*: How long does the vulnerability reside in the code?*

In this research question, we analyzed the date that the first time the developers explicitly mentioned the CVE identifiers in the code comments. To achieve this goal, we analyzed all 1,366 source code comments that refer to the CVE identifiers, as described in Table I. To extract the date, we implemented a tool [30], [31] that uses `git blame` (also as known as "annotate") to the selected code comments. With this tool, we are able to extract the actual date the first time the code author wrote the CVE-related comments.

To obtain the time duration of the CVE identifiers that have been referred to in the code comments, we calculated the time differences between the initial creation date of the code comments yielded from the annotation and December 31, 2019. We chose the end of 2019 to calculate the time differences since the dataset of source code comments was extracted in August 2020. The resulting time differences are described in days.

Results. As illustrated in Figure 7, the duration of CVE identifiers referenced in the source code comments is varied for all vulnerability severity levels. The CVE identifiers that are identified as critical severity levels have been cited by the developers for mostly between 600 and 1,100 days (with the first quartile (q1): 612, median: 889, and the third quartile (q3): 1,095). In most cases, the high and medium severity levels of the CVE identifiers are included in the source code comments for between 400 and 1,100 days (with q1: 490, median: 739, q3: 1,042 for high severity levels, and q1: 482, median: 751, q3: 936 for medium severity levels). The time duration of the low severity CVE identifiers cited in code comments is more varied compared to the other three prominent severity levels (q1: 376, median: 721, q3: 1,219). On the other hand, the CVE identifiers that unknown severity levels, have mostly been cited much longer in the code comments, that is more than 1,500

[6]https://nvd.nist.gov/vuln-metrics/cvss

TABLE IV
TOP 10 CVE IDENTIFIERS THAT ARE REFERENCED IN SOURCE CODE COMMENTS

CVE id	Severity level	Product affected by the CVE	# Code comments	%
CVE-2016-10033	Critical	The isMail transport in PHPMailer before 5.2.18	24	1.25%
CVE-2016-10045	Critical	The isMail transport in PHPMailer before 5.2.20	23	1.20%
CVE-2009-3555	None	TLS protocol and SSL protocol 3.0 and possibly earlier	20	1.04%
CVE-2016-10074	Critical	Mail transport in Swift Mailer before 5.4.5	17	0.88%
CVE-2014-0160	High	TLS and DTLS implementations in OpenSSL 1.0.1	17	0.88%
CVE-2012-2459	None	Denial of service in Bitcoin network	13	0.68%
CVE-2016-7420	Medium	Crypto++ through 5.6.4	12	0.62%
CVE-2016-10034	Critical	zend-mail component before 2.4.11, 2.5.x, 2.6.x, and 2.7.x before 2.7.2, and Zend Framework before 2.4.11	10	0.52%
CVE-2014-6278	None	GNU Bash through 4.3 bash43-026	10	0.52%
CVE-2014-6271	Critical	GNU Bash through 4.3	10	0.52%

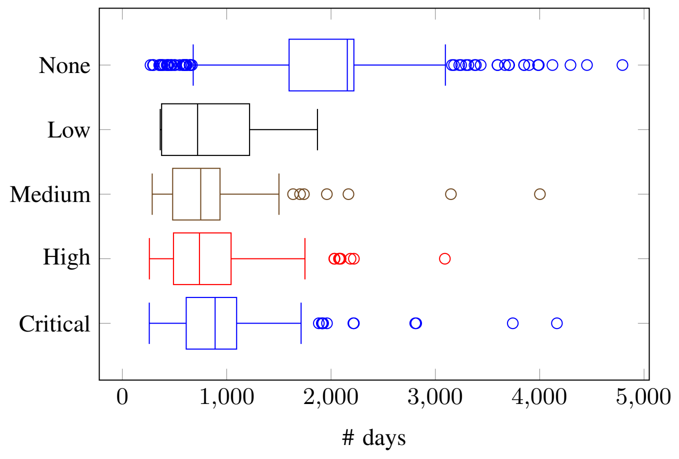


Fig. 7. Number of days the CVE identifiers reside in source code comments until December 31, 2019. Although there are number of CVE identifiers that are included more than 5,000 days for the “None” severity level, we limit up to 5,000 in the figure.

days. The result hints that the vulnerabilities have majorly notified developers for years. Thus, software developers should be aware of this phenomenon.

Summary of RQ$_3$. Our findings show that most CVE identifiers referenced in source code comments are critically and highly severe, where they indicate the vulnerability issues in the related code for more than 2 years.

V. IMPLICATION AND RECOMMENDATION

In this section, we present a description of the impact of our study results, as follows:

- In our study, we found that the utilization of vulnerability identifiers in code comments as a mean of indicating workaround is needed to address security issues. Researchers have demonstrated that open source software is prone to security vulnerabilities [32]. Furthermore, Figure 7 shows the evidence that vulnerability can exist in source code for several years. Therefore, we suggest open source software developers to check vulnerability-related code on a regular basis.
- We noticed that there are many code comments only include vulnerability identifiers without any further explanation. Since code comment is crucial for communication between software maintainers, thus, writing code comments with clear description is required, as suggested in prior study [9].
- We also observed that most related comments that describe vulnerabilities are located in the main source code that consists of multiple statements. It indicates the feasibility of automatic tools development to predict open source software security vulnerabilities based on new features, such as vulnerability identifiers, statement types, etc. Several previous research works show the path to develop feature-based automatic software vulnerability detection, such as dynamic behavior features [33] and flaw function heuristic [34].

We can also recommend researchers to consider future work, as follows:

- *Further study on vulnerability-related code evolution* to understand the changes of source code to address the vulnerability issues,
- *Analyze vulnerability-related commit messages* to understand the more general purposes of vulnerability identifiers inclusion in the commit message,
- *Investigate the CVE ID-related discussion on Stack Overflow (SO)*, as the SO threads might be useful to overcome the vulnerability in the code.
- *Analyze the code length and the complexity of the method* to help practitioners in getting new insight about software vulnerability.

VI. THREATS TO VALIDITY

Several potential threats to the *construct validity* emerge in our study. Regarding the code comments extraction, it is possible that not all vulnerability-related source code comments could be extracted using the defined regular expression, since the vulnerability identifiers might be written in different formats in code comments. In addition, we limited the data extraction until the end of 2019, so that the data from the beginning of 2020 to the date of data collection in this work will not be obtained. However, the number of these issues is small, thus, we consider that the impact of the missing code comments is not significant. The other threat to the construct validity relates to our manual labeling. The labels might be affected by annotator’s misunderstanding or mislabeling. To mitigate this issue, the annotators resolve the disagreements through discussion, so that we can minimize the mislabeling in our manual analyses.

The threats to the *external validity* appear in the data preparation. Despite we investigate a large number of GitHub repositories across 7 popular programming languages, the results could not be generalized to other software development projects, and other programming languages.

We mitigate the threats to *reliability* by preparing an online appendix of our analyzed dataset. The appendix is described in Section III-C.

VII. Conclusion

This paper presents an empirical analysis of vulnerability identifiers that are explicitly written in source code comments. The source code comments were extracted from 32,007 GitHub repositories across 7 popular programming languages, that are, C, C++, Java, JavaScript, PHP, Python, and Ruby. To understand the characteristics and the role of the vulnerability identifiers in code comments, we conducted 2 analyses, (i) quantitative and qualitative analysis on 1,491 vulnerability-related code comments to get insights the rationale of software developers adding the vulnerability identifiers in source code comments, and (ii) severity analysis on the most dominant vulnerability identifiers.

Our work has shown that CVE, the most common vulnerability identifiers found in source code comments, has critical and high severity levels to indicate that the related code is used to address the vulnerability issues and alert developers on using the code. Based on this work which has identified the common vulnerability identifiers in code comments and their general purposes, there are many opportunities for future work: understanding the evolution of vulnerability-related comments and source code, and further investigations of source code comments, to name a few.

Acknowledgment

This research is fully funded by Hibah Integrasi Tridharma (HIT) Universitas Muhammadiyah Surakarta under the grant number 007/A.3-III/FKI/I/2021.

References

[1] Y. Shin, A. Meneely, L. Williams, and J. A. Osborne, "Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities," *IEEE transactions on software engineering*, vol. 37, no. 6, pp. 772–787, 2010.

[2] X. Wang, R. Ma, B. Li, D. Tian, and X. Wang, "E-wbm: an effort-based vulnerability discovery model," *IEEE Access*, vol. 7, pp. 44 276–44 292, 2019.

[3] S. S. Murtaza, W. Khreich, A. Hamou-Lhadj, and A. B. Bener, "Mining trends and patterns of software vulnerabilities," *Journal of Systems and Software*, vol. 117, pp. 218–228, 2016.

[4] J. Ruohonen, S. Rauti, S. Hyrynsalmi, and V. Leppänen, "A case study on software vulnerability coordination," *Information and Software Technology*, vol. 103, pp. 239–257, 2018.

[5] N. Munaiah, A. Rahman, J. Pelletier, L. Williams, and A. Meneely, "Characterizing attacker behavior in a cybersecurity penetration testing competition," in *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2019, pp. 1–6.

[6] F. Piessens and I. Verbauwhede, "Software security: Vulnerabilities and countermeasures for two attacker models," in *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2016, pp. 990–999.

[7] A. Decan, T. Mens, and E. Constantinou, "On the impact of security vulnerabilities in the npm package dependency network," in *Proceedings of the 15th International Conference on Mining Software Repositories*, ser. MSR '18. Association for Computing Machinery, 2018, p. 181–191.

[8] M. A. Williams, R. C. Barranco, S. M. Naim, S. Dey, M. S. Hossain, and M. Akbar, "A vulnerability analysis and prediction framework," *Computers & Security*, vol. 92, p. 101751, 2020.

[9] D. Steidl, B. Hummel, and E. Juergens, "Quality analysis of source code comments," in *2013 21st international conference on program comprehension (icpc)*. IEEE, 2013, pp. 83–92.

[10] M.-A. Storey, J. Ryall, R. I. Bull, D. Myers, and J. Singer, "Todo or to bug: exploring how task annotations play a role in the work practices of software developers," in *Proceedings of the 30th international conference on Software engineering*, 2008, pp. 251–260.

[11] A. Potdar and E. Shihab, "An exploratory study on self-admitted technical debt," in *2014 IEEE International Conference on Software Maintenance and Evolution*. IEEE, 2014, pp. 91–100.

[12] E. d. S. Maldonado and E. Shihab, "Detecting and quantifying different types of self-admitted technical debt," in *2015 IEEE 7Th international workshop on managing technical debt (MTD)*. IEEE, 2015, pp. 9–15.

[13] E. da Silva Maldonado, E. Shihab, and N. Tsantalis, "Using natural language processing to automatically detect self-admitted technical debt," *IEEE Transactions on Software Engineering*, vol. 43, no. 11, pp. 1044–1062, 2017.

[14] H. Hata, C. Treude, R. G. Kula, and T. Ishio, "9.6 million links in source code comments: Purpose, evolution, and decay," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, 2019, pp. 1211–1221.

[15] A. Inokuchi, Y. S. Nugroho, S. Wattanakriengkrai, F. Konishi, H. Hata, C. Treude, A. Monden, and K. Matsumoto, "From academia to software development: Publication citations in source code comments," *arXiv preprint arXiv:1910.06932*, 2019.

[16] B. Yang, Z. Liping, and Z. Fengrong, "A survey on research of code comment," in *Proceedings of the 2019 3rd International Conference on Management Engineering, Software Engineering and Service Sciences*. Association for Computing Machinery, 2019, p. 45–51.

[17] D. Haouari, H. Sahraoui, and P. Langlais, "How good is your comment? a study of comments in java programs," in *2011 International Symposium on Empirical Software Engineering and Measurement*. IEEE, 2011, pp. 137–146.

[18] M. Linares-Vásquez, G. Bavota, and C. Escobar-Velásquez, "An empirical study on android-related vulnerabilities," in *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, 2017, pp. 2–13.

[19] H. Guo, Y.-Y. Wang, Z.-L. Pan, and S.-W. Liu, "Research on detecting windows vulnerabilities based on security patch comparison," in *2016 Sixth International Conference on Instrumentation Measurement, Computer, Communication and Control (IMCCC)*, 2016, pp. 366–369.

[20] M. Jimenez, R. Rwemalika, M. Papadakis, F. Sarro, Y. Le Traon, and M. Harman, "The importance of accounting for real-world labelling when predicting software vulnerabilities," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2019. Association for Computing Machinery, 2019, p. 695–705.

[21] Z. Han, X. Li, Z. Xing, H. Liu, and Z. Feng, "Learning to predict severity of software vulnerability using only vulnerability description," in *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2017, pp. 125–136.

[22] L. Pascarella, M. Bruntink, and A. Bacchelli, "Classifying code comments in java software systems," *Empirical Software Engineering*, vol. 24, no. 3, pp. 1499–1537, 2019.

[23] B. Fluri, M. Würsch, E. Giger, and H. C. Gall, "Analyzing the co-evolution of comments and source code," *Software Quality Journal*, vol. 17, no. 4, pp. 367–394, 2009.

[24] R. Maipradit, C. Treude, H. Hata, and K. Matsumoto, "Wait for it: identifying "on-hold" self-admitted technical debt," *Empirical Software Engineering*, vol. 25, no. 5, pp. 3770–3798, 2020.

[25] A. Alhefdhi, H. K. Dam, Y. S. Nugroho, H. Hata, T. Ishio, and A. Ghose, "A framework for self-admitted technical debt identification and description," *arXiv preprint arXiv:2012.12466*, 2020.

[26] I. K. Ratol and M. P. Robillard, "Detecting fragile comments," in *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2017, pp. 112–122.

[27] C. Zapponi, "Githut 2.0: A small place to discover languages in github," https://madnight.github.io/githut/, accessed: 2020-07-28.

[28] A. Viera and J. Garrett, "Understanding interobserver agreement: The kappa statistic," *Family Medicine*, vol. 37, no. 5, pp. 360–363, 5 2005.

[29] I. Rehman, D. Wang, R. G. Kula, T. Ishio, and K. Matsumoto, "Newcomer candidate: Characterizing contributions of a novice developer to github," in *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2020, pp. 855–855.

[30] J. Eyolfson, L. Tan, and P. Lam, "Do time of day and developer experience affect commit bugginess?" in *Proceedings of the 8th Working Conference on Mining Software Repositories*, 2011, pp. 153–162.

[31] F. Rahman and P. Devanbu, "Ownership, experience and defects: a fine-grained study of authorship," in *Proceedings of the 33rd International Conference on Software Engineering*, 2011, pp. 491–500.

[32] X. Qing, "Initial analysis of open source software in network and information security system," *Computer Application and Software*, vol. 30, pp. 325–327, 2013.

[33] Y. Li, L. Ma, L. Shen, J. Lv, and P. Zhang, "Open source software security vulnerability detection based on dynamic behavior features," *Plos one*, vol. 14, no. 8, p. e0221530, 2019.

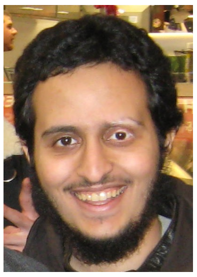
[34] Q. Liu, H. Chen, Y. Wen, and X. Li, "Towards a flaw function heuristic vulnerability static analysis framework for executable file," in *2011 Seventh International Conference on Mobile Ad-hoc and Sensor Networks*. IEEE, 2011, pp. 436–440.

Yusuf Sulistyo Nugroho is a lecturer at the Department of Informatics, Universitas Muhammadiyah Surakarta, Indonesia. He received his Ph.D degree from Nara Institute of Science and Technology in 2020. His research interests include Empirical Software Engineering, Software Documentation, and Mining Software Repositories. Further info on his homepage: https://yusufsn.github.io/.

Dedi Gunawan is currently being a lecturer in Informatics Department, Universitas Muhammadiyah Surakarta. He graduated from electrical engineering from Universitas Muhammadiyah Surakarta. In 2014 he obtained Master degree from National Dong Hwa University, Taiwan while in 2019 he achieved Doc-toral degree from Kanazawa University, Japan. His current research interest including privacy enhancing technology, data anonymity, privacy preserving data mining and privacy preserving data publishing.

Devi Afriyantari Puspa Putri is a lecturer at the Department of Informatics, Universitas Muhammadiyah Surakarta, Indonesia. She received her Master degree of Advanced Computer Science from University of Manchester in 2017. Her research interests include: Outlier detection and Mobile programming.

Syful Islam is working as an assistant professor of Noakhali Science and Technology University, Bangladesh. He received the M.E. and Ph.D. degree from Nara Institute of Science and Technology, Japan. His research interests include software ecosystem, mining Stack Overflow, mining software repositories etc. Contact him at syfulcste@nstu.edu.bd



Abdulaziz Alhefdhi is a PhD candidate at the School of Computing and Information Technology, University of Wollongong (UOW), Australia. He is also a lecturer at the department of Computer Science at Prince Sattam bin Abdulaziz University (PSAU), Saudi Arabia. Alhefdhi received his Bachelor's and Master's degrees in Computer Science form Imam Mohammad Ibn Saud Islamic University (IMSIU), Saudi Arabia and the University of Queensland (UQ), Australia, respectively. He is a member of the Decision Systems Lab (DSL) in UOW. His research interests include Software Analytics and AI-empowered Software Engineering.