JKSCI

한국컴퓨터정보학회논문지
Journal of The Korea Society of Computer and Information
Vol. 26 No. 6, pp. 89-96, June 2021
https://doi.org/10.9708/jksci.2021.26.06.089

An Empirical Study on the Impact of Permission Smell in Android Applications

Zhiqiang Wu*, Hakjin Lee*, Scott Uk-Jin Lee*

*Student, Dept. of Computer Science & Engineering, Hanyang University, Ansan, Korea
*Student, Dept. of Computer Science & Engineering, Hanyang University, Ansan, Korea
*Associate Professor, Dept. of Computer Science & Engineering, Hanyang University, Ansan, Korea

[Abstract]

In this paper, we proposed a sniffer to detect permission smells from developer and third-party libraries' code. Moreover, we conducted an empirical study to investigate unnecessary permissions on large real-world Android apps. Our analysis indicates that permission smell extensively exists in Android apps. According to the results, permission smells exist in most Android apps. In particular, third-party libraries request permission for functionalities that are not used by developers, which cause more smells. Moreover, most developers do not properly disable unnecessary permissions that are declared for third-party libraries. We discussed the impacts of permission smells on user experiences. As a result, the existence of permission smell does not impact the number of downloads. However, apps that have more unnecessary permissions have received lower ratings from users.

▸ **Key words**: Android, Code Smell, Permission, Empirical Study, Security Risk

[요 약]

본 논문에서는 개발자가 작성한 코드와 써드파티 라이브러리로 인해 발생하는 Permission Smell을 탐지하여 그 영향에 대해 다각적으로 분석했다. 이를 위해서 실제 구글 플레이 스토어에 존재하는 Android 앱로 구성된 대규모 데이터셋을 활용하여 존재하는 Permission Smell의 영향을 조사 및 분석하는 실증적 연구를 수행하였다. 연구 결과에 따르면 대다수의 안드로이드 앱에 Permission Smell이 존재하며 특히 써드파티 라이브러리는 개발자가 사용하지 않는 기능에 대해서도 권한을 요구하므로 이러한 Smell 들을 더 많이 발생시킨다. 또한, 대다수의 개발자는 써드파티 라이브러리로 인해 선언된 불필요한 권한을 올바르게 비활성화하지 않는다는 것을 파악하였다. 이러한 결과를 바탕으로 본 논문에서는 Permission Smell이 사용자 경험에 미치는 영향에 대해 논의한다. 결과적으로 불필요한 권한을 요구하는 앱이더라도 다운로드 횟수에 영향을 주지는 않았다. 그러나 불필요한 권한을 요구하는 앱들은 사용자들로부터 더 낮은 평가를 받았다.

▸ **주제어**: 안드로이드, 코드 스멜, 권한, 실증적 연구, 보안 위험

• First Author: Zhiqiang Wu, Corresponding Author: Scott Uk-Jin Lee
*Zhiqiang Wu (wzq0515@hanyang.ac.kr), Dept. of Computer Science & Engineering, Hanyang University
*Hakjin Lee (gsdjini91@hanyang.ac.kr), Dept. of Computer Science & Engineering, Hanyang University
*Scott Uk-Jin Lee (scottlee@hanyang.ac.kr), Dept. of Computer Science & Engineering, Hanyang University
• Received: 2021. 05. 11, Revised: 2021. 05. 26, Accepted: 2021. 05. 26.

I. Introduction

Android is the most popular mobile operating system in April 2021, with 83.8% worldwide market share [1]. With such an enormous user base, developers eagerly publish more applications (apps for short). As the official app market for Android apps, Play Store has more than 2.98 million apps [2].

Despite the continuous increase of Android apps, many apps still have quality issues due to bad practices [3]. A code smell is a recurring code pattern that causes software quality deterioration such as maintainability, readability, and changeability [4]. In particular, Android apps contain traditional Object-Oriented code smells but also mobile-specific smells due to their framework [5]. The most common smell in Android is permission smell that indicates the app's manifest file contains permissions that are not used [6]. More than 80% of Android apps over-claimed at least one unnecessary permission in the manifest file, which may expose users to additional security risks [7]. To eliminate such smell, developers need to remove corresponding permissions in the manifest file whose associated APIs are not invoked in the source code. Developers could not distinctly seek out the invocation relationship between APIs and permissions since APIs are not always documented very well in the official documentation [8]. Although the existing results are able to analyze the unnecessary permission in the customized code from developers, they ignored a critical issue: the unnecessary permissions from Third-Party Libraries (TPLs). In general, developers leverage the existing TPLs to achieve some trivial functionalities, which invoked permissions to access sensitive information. In order to enable such libraries, the corresponding permissions should be declared in the manifest. However, the permissions for TPLs could not be handled properly in practice, which causes security risks and degrades the user experience.

Therefore, we conduct a static analysis to identify the unnecessary permission smell for enabling TPLs. Based on that, we discuss the occurrence frequency of such permissions on large real-world Android apps from the Play Store. In addition, we perform an empirical study to investigate the impact of unnecessary permissions in terms of user experience and security aspects.

Table 1. Statistic of Permissions in Android

Protection level	# Permissions
Dangerous	30
Normal	51
Signature	97
Total	178

II. Preliminaries

1. Background

Permission mechanism. Android system provides a permission mechanism to protect users' privacy. Based on the privacy level of accessed data, permissions are categorized into three classes: normal, signature, and dangerous permissions. In order to access the sensitive information and hardware with built-in APIs, Android apps are required to declare all needed permissions in the AndroidManifest.xml file. For instance, if a given app needs to access the privacy information by APIs, APIs have become callable when the corresponding permission is declared in the manifest file. From Android 6, although users may grant/deny dangerous permissions to disable associated features, developers always claim more permissions than the actual demand of implemented features to accelerate development [9]. Such over-claimed permissions may cause additional security risks [10, 20]. Table 1 shows the statistic of permissions in the Android 9. Normal and signature permissions support the basic features to app like internet connection, which may not cause security issues. These can be automatically granted when user installs it. However, dangerous permissions allow app to access sensitive information in order to provide various features. From Android 6, user are required to approve/deny dangerous permissions when the corresponding functionality is triggered at the first time. According to previous study [9], 26 out of 30 dangerous permissions impacts users' privacy, which are listed in Table 2.

Third-party library. A TPL is an external package that contains a set of functionalities, which can be reused by developers handily. In general, a TPL depends on other external libraries, which requires developers to install corresponding libraries as well. In particular, TPL in the Android platform may invoke APIs of permissions [11]. To reuse the functionalities in the TPLs, developers have to declare all permissions that are used in TPLs. Otherwise, the undeclared permissions will cause the runtime error even the corresponding functionalities are not called in the apps [12]. Zhang et al. [21] indicate that over half of potential malicious TPLs request excessive unnecessary permissions as shown in Fig. 1. Over half of apps request Phone, Location, and Storage permissions, which can cause the leakage of privacy information.

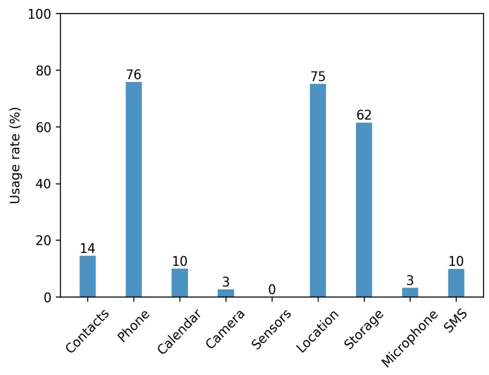


Fig. 1. Dangerous Permissions Used by TPLs

2. Related work

Most studies related to our work, focus on detecting whether the associated API/URIs of declared permissions in the manifest file are invoked in the source code. Dennis et al. introduced one bad permission practice [13]. When executing code that requires dangerous permission, the API *checkSelfPermission()* are required to check whether the user has granted the corresponding permission for the app. Developers do not always leverage this API to check it. If a user denied the permissions, there is a high chance of the app crashing without checking its grant. Wu et al. [6] proposed a method to detect unnecessary permissions using a mapping between permissions and APIs. In addition, Xiao et al. [7] leveraged collaborative filtering to recommend the minimum permission set for each topic based on app descriptions. Liu et al. [14] inferred the functionalities from app descriptions to recommend a set of permissions. Although these previous works can either detect unnecessary permissions or overcome this smell by analyzing relevant metadata, they are only available to detect the smell in the customized code without considering TPLs.

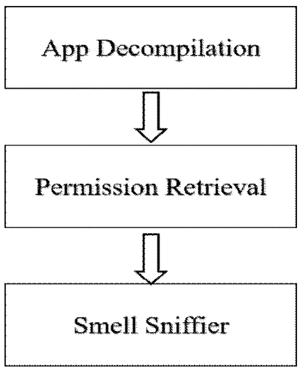


Fig. 2. Procedure of Smell Sniffier

III. Permission Smell Detection

This section presents an approach to detect the permission smells in both customized code and TPLs. Fig. 2 shows the procedure of our approach. The detailed process is described below.

1. Permission Retrieval

To detect the permission smells in apps, Android Package Kit (apk) files are decompiled by AndroGuard [15] to obtain *.dex* and *AndroidManifest.xml* file. The former file is used to hold a set of class definitions and associated methods. The latter one is a mandatory file in the Android app, which describes essential information about the app such as permissions, activities, package name, and so on. In general, the permissions are declared in tag 'use-permission.' We observed that some apps declare the dangerous permissions with tag 'use-permission-sdk23' in the latest Android versions. Therefore, our approach leverages regular expression to extract all declared permissions based on the above-mentioned tags. In addition, declared permission could be disabled with attribute *node="remove"*, which is usually used to prevent the permissions in TPLs. Such disabled permissions satisfy the best permission practices without introducing any security risk. In this case, our approach does not consider them as permission smells. Finally, a set of declared permissions are extracted from each app.

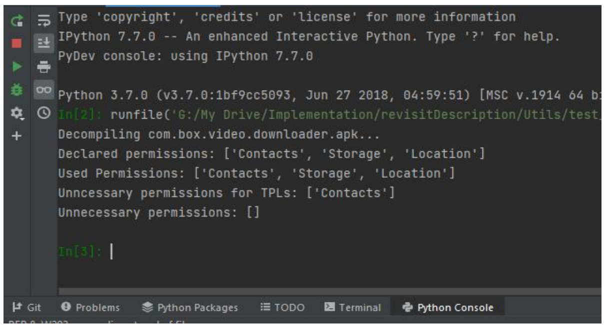
2. Permission Smell Sniffer

To detect the unnecessary permission smell, we need to identify whether the sensitive APIs used in the source code. However, the official documentation has not published such mapping between APIs and required permissions. In this study, we leverage an existing API mapping from PScout [16], which contains 2,118 sensitive APIs in this work.

For a given Android apk file, the *.dex* file is converted into *smali* code. To reduce searching complexity, we only check a subset of API mappings whose permissions are declared in the manifest. Our approach leverages AndroGuard to search the usages of APIs based on their packages, methods, and parameters. In particular, some permission only governs one field used in a general API to achieve the corresponding features. In this case, our approach further analyzes whether the input argument is associated field. For instance, *Sensors* class manages various sensors by type codes in the Android system [17]. A sensor is called when its corresponding field is passed to API *getDefaultSensor*. Therefore, we only consider that permission is used in the app if its field has been passed to API.

Once the permission is flagged as invoked, we further track its sources to discover whether the corresponding APIs are only invoked by TPLs. The permission is determined as TPL smell if the APIs are only invoked in the TPLs and the associated APIs in TPLs are not called in the source code. Otherwise, the permission is not considered in our study. In addition, some permissions have been used in neither app nor TPLs. In this case, our approach classifies them to Unnecessary Permission (UP) smell.

Our sniffer works on top of several state-of-the-art tools (i.e., AndroGuard, PScout) with Python script to check whether the declared permissions are used in the source code. Fig. 3 is a prototype screen for *Sharego Browser* app. As a result, the developers of this app correctly declared permissions for their own code. However, the result indicates that developers did not disable the excessive permissions from TPLs (i.e., Contact permission). The experiments were conducted on a desktop with an Intel Core i7-7700 processor and 32 GB RAM.


```
Type 'copyright', 'credits' or 'license' for more information
IPython 7.7.0 -- An enhanced Interactive Python. Type '?' for help.
PyDev console: using IPython 7.7.0

Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:59:51) [MSC v.1914 64 b
In[2]: runfile('G:/My Drive/Implementation/revisitDescription/Utils/test_
Decompiling com.box.video.downloader.apk...
Declared permissions: ['Contacts', 'Storage', 'Location']
Used Permissions: ['Contacts', 'Storage', 'Location']
Unncessary permissions for TPLs: ['Contacts']
Unnecessary permissions: []

In[3]: |
```

Fig. 3. Prototype Screen

IV. Empirical Study

In this section, we empirically analyze unnecessary permission smell on large real-world Android apps to answer the following research questions.

RQ1: How is unnecessary permission smell prevalent in real-world apps, especially for DP?

RQ2: Does unnecessary permission smell impact the user experience? We leverage a statistical approach to investigate the correlation between smell and meta-data from users.

1. Dataset

This work randomly collected 12,169 Android apps from AndroZoo [18], a weekly update Android repository from various markets. In AndroZoo, an app may occur multiple times with different version codes. Thus, only the latest versions of apps are collected in order to ensure the consistency between apps and users' metadata. In addition, the corresponding metadata is crawled from the Play Store to investigate the impact of permission smell on user experience.

There are 178 permissions in the Android system. However, only 30 dangerous permissions out of them provide associated APIs to threaten the security risks for Android devices or users. We cherry-picked 26 out of them that access user privacy in our study. The discussed permissions are categorized into 10 groups based on the permission grant mechanism [19], as shown in Table 2.

Table 2. Discussed Permissions

Permission Groups	Permission
Calendar	READ_CALENDAR
	WRITE_CALENDAR
Contacts	READ_CONTACTS
	WRITE_CONTACTS
	GET_ACCOUNTS
Location	ACCESS_FINE_LOCATION
	ACCESS_COARSE_LOCATION
Storage	WRITE_EXTERNAL_STORAGE
	READ_EXTERNAL_STORAGE
Call Log	READ_CALL_LOG
	WRITE_CALL_LOG
Microphone	RECORD_AUDIO
Camera	CAMERA
Phone	READ_PHONE_STATE
	READ_PHONE_NUMBERS
	CALL_PHONE
	ANSWER_PHONE_CALLS
	ADD_VOICEMAIL
	USE_SIP
	ACCEPT_HANGOVER
SMS	READ_SMS
	RECEIVE_SMS
	RECEIVE_WAP_PUSH
	SEND_SMS
	RECEIVE_SMS
Sensors	BODY_SENSORS

2. RQ1: The Prevalence of Permission Smell

To answer this research question, we analyzed 12,169 Android apps. As shown in Fig. 4, over half of apps request at least one permission smell. More specifically, 10.47% of apps claim excessive permissions that have never been used in the source code with associated APIs. 6,438 apps (52.9%) applied at least one TPL in the source code. However, only 824 apps completely invoked all permissions that are used in TPLs. As we can see, 46.13% and 25.34% of apps contain DP smells and UP smells due to the bad practices of permission usages, respectively.

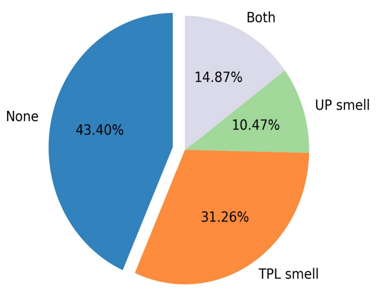


Fig. 4. Distribution of Permission Usages

As a result, the occurrences of unnecessary permissions in the customized code less than TPLs since developers may declare the permissions based on their demands. For permission smells in TPLs, we have checked all apps in our dataset to confirm whether developers disabled the over-claimed permissions that only support TPLs. Table 3 presents disable permissions in our experiment. Overall, 6,438 apps in our dataset totally declared 10,453 unnecessary permissions for TPLs. Unfortunately, none of these apps has disabled such permissions in the manifest file using a code *node="remove"* from our collected dataset. Such permission smells from TPLs may cause more security vulnerabilities since developers could not realize all functionalities of TPLs.

Table 3. Unnecessary Permissions from TPLs

Type	# Permissions
Disabled	0
Remained	10,453
Total	10,453

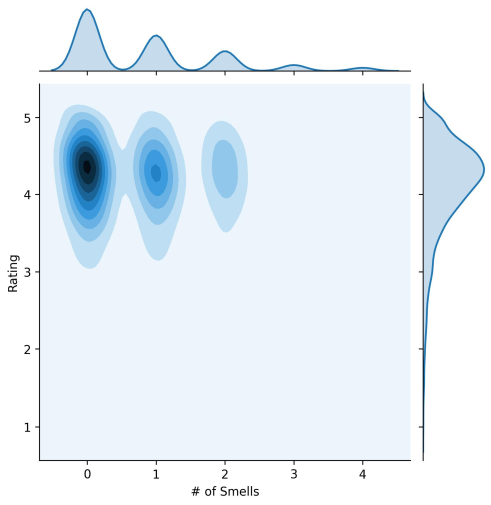


Fig. 5. Correlation between Permission Smells and Rating

3. RQ2: The Impact of Permission Smell

To answer this question, we empirically analyze the correlation between the number of smells and user feedback in terms of rating and the number of downloads from the Play Store. To avoid bias of data distribution, we removed some outliers based on the following conditions: (1) apps that are top-ranking in the market have tremendous downloads, which causes a severe bias in correlation analysis; (2) apps are rated by few users, which may cause a cognitive bias; (3) the meta-data is non-existence. After filtration, a total of 8,396 apps and their metadata are applied to explore this research question.

Downloads. Play Store provides an approximate number of downloads rather than exact amounts for each app, such as '10+' and '100,000+', which is a typical discrete variable. Based on that, we leveraged the Chi-squared test to verify whether the number of permission smells in the apps impacts user downloads. We set the confidence *p-value* <0.05 to verify the correlation between downloads and smells. However, the number of downloads is not related to how many unnecessary permission smells in the app.

Rating. Fig. 5 shows the relationship between the number of smells and ratings. As we can see, the apps have higher ratings if they do not contain permission smell. With increasing the number of permission smells in the apps, the apps that are still rated with higher scores have become less, which indicates that the users are aware of the security threats while using apps.

V. Conclusions

In this study, we detected the unnecessary permission smells in both customized code and TPLs to reveal the security risks from third-party libraries. In addition, we have conducted an empirical analysis to discover the distribution of permission smells on large real-world Android apps. Our experimental analysis indicates that the permission smells exist in over half of the apps in our dataset. TPLs involve excessive permission usages that developers may not use. Moreover, we observed that developers only declared over-claimed permissions in the manifest file without correctly disabling them. Such permission smell can make the app in the additional security vulnerabilities. In addition, we also investigate the impact of permission smells on user experience. As a result, the permission smell in the app does not impact whether users download it since users cannot obtain enough knowledge about apps based on app descriptions. Due to resource limitations, we only applied a small portion of Android apps in our study. Therefore, we could not investigate whether apps with lower ratings and unnecessary permissions are removed from markets. However, we found that the apps with higher scores and more permission smell become less since users are aware of the possible security threats after using apps

In the future study, we plan to build a tool for developers to detect unnecessary permissions and reveal their usages in the source code. It will assist developers in refining the permissions usages in the apps in order to avoid potential security risks.

ACKNOWLEDGEMENT

This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (NRF-2020R1F1A1076208).

REFERENCES

[1] Android Market Share, https://www.idc.com/promo/smartphone-market-share/os

[2] AppBrain, https://www.appbrain.com/stats/number-of-android-apps

[3] H. Wang, H. Li, L. Li, Y. Gao and G. Xu, "Why are Android Apps Removed From Google Play? A Large-scale Empirical Study," Proceedings of International Conference on Mining Software Repository, pp. 231-242, 2018.

[4] H. Mumtaz, M. Alshayeb, S. Mahmood and M. Niazi, "An Empirical Study to Improve Software Security Through the Application of Code Refactoring," Information and Software Technology, vol. 96, pp. 112-125, Apr. 2018.

[5] K. Rahkema and D. Pfahl, "Empirical Study on Code Smells in iOS Applications," Proceedings of International Conference on Mobile Software Engineering and Systems, pp. 61-65, 2020.

[6] Z. Wu, X. Chen and S. U.-J. Lee, "Permission Smells Detection for IoT Applications on Android Platform," Proceeding of Korean Computer Congress, pp. 293-295, 2019.

[7] J. Xiao, S. Chen, Q. He, Z. Feng and X. Xue, "An Android Application Risk Evaluation Framework Based on Minimum Permission Set Identification," Journal of Systems and Software, vol. 163, pp. 110533, May 2020.

[8] C. Lyvas, C. Lambrinoudakis and D. Geneiatakis, "Dyperm- in: Dynamic permission mining framework for android platform," Computer & Security, vol. 77, pp. 472-487, Aug. 2018.

[9] Z. Wu, X. Chen, S. U.-J. Lee, "FCDP: Fidelity Calculation for Description-to-Permissions in Android Apps," IEEE Access, vol. 9, pp. 1062-1075, Jan. 2021.

[10] L. Yu, X. Luo, C. Qian, S. Wang and H. K. N. Leung, "Enhancing the Description-to-Behavior Fidelity in Android Apps with Privacy Policy," IEEE Transactions on Software Engineering, vol. 44, no. 9, pp. 834-854, Jul. 2017.

[11] T. Watanabe, M. Akiyama, T. Sakai, H. Washizaki and T. Mori, "Understanding the Inconsistency between Behaviors and Descriptions of Mobile Apps," IEICE Transactions on Information and Systems, vol. 101, no. 11, pp. 2584-2599, Nov. 2018.

[12] C. Zhang, H. Wang, R. Wang, Y. Guo and G. Xu, "Re-checking App Behavior against App Description in the Context of Third-party Libraries," Proceeding of International Conference on Software Engineering and Knowledge Engineering , 2018. DOI: 10.18293/SEKE2018-1 80

[13] C. Dennis and D. E. Krutz, "P-Lint: A Permission Smell Detector for Android Applications," Proceeding of IEEE/ACM International Conference on Mobile Software Engineering and Systems, pp. 219-220, 2017.

[14] X. Liu, Y. Leng, Y. Yang and C. Zhai, "Mining Android App Descriptions for Permission Requirements Recommendation," Proceeding of 2018 IEEE 26th International Requirements Engineering Conference (RE), pp. 147-158, 2018.

[15] AndroGuard, https://github.com/androguard/androguard

[16] K. W. Y. Au, Y. Zhou, Z. Huang and D. Lie, "PScout: Analyzing the Android Permission Specification," Proceedings of the 2012 ACM conference on Computer and Communications Security, pp. 217-228, 2012.

[17] Android Sensors, https://developer.android.com/reference/android/hardware/SensorManager

[18] K. Allix, T. Bissyande, J. Klein and Y. L. Traon, "AndroZoo: Collecting Millions of Android Apps for the Research Community," Proceedings of International Conference on Mining Software Repositories, pp. 468-471, 2016.

[19] P. Calciati, K. Kuznetsov, A. Gorla and A. Zeller, "Automatically Granted Permissions in Android Apps," Proceedings of the 17th International Conference on Mining Software Repositories, pp. 114-124, 2020.

[20] M. Ghafari and P. Gadient, "Security Smells in Android," Proceeding of 2017 IEEE 17th International Conference on Source Code Analysis and Manipulation, pp. 121-130, 2017.

[21] Z. Zhang, W. Diao, C. Hu, S. Guo, C. Zuo and L. Li, "An Empirical Study of Potentially Malicious Third-Party Libraries in Android Apps," Proceeding of ACM Conference on Security and Privacy in Wireless and Mobile Networks, pp. 144-154, 2020.

Authors

Zhiqiang Wu received the B.S. in computer science from Shanghai Polytechnic Univ. in 2015. He also received M.S. degree from Hanyang University in 2017. Currently, he is pursuing the Ph.D. degree in computer science with the Dept. of Computer Science & Engineering, Hanyang University, Ansan, South Korea. His research interests include Android apps analysis, code smells on security and software refactoring on mobile apps.

Hakjin Lee received the B.S. degree in Computer Science and Engineering from Hanyang University, Korea, in 2017. He is currently a attending Integrated Master's and Doctorate Course in the Department of Computer Science, Hanyang University. He is interested in SW Smell, Quality Assurance.

Scott Uk-Jin Lee received the B.S. degree in software engineering and the Ph.D. degree in computer science from University of Auckland, New Zealand. He was a Post-Doctoral Research Fellow at the Commissariat à l’énergieatomique et aux énergies alternatives, France. He is currently serving as Associate Professor of the Department of Computer Science and Engineering, Major in Bio Artificial Intelligence. His research interests include software engineering, formal methods, and quality assurance. He is also a member of the Korean Institute of Information Scientists and Engineers and the Korean Society of Computer and Information. He has served as an editor, the technical chair, and a committee member for several journals and conferences.