Engineering mobile apps for disaster management — the case of COVID-19 apps in the Google Play Store

Authors

- **Ivano Malavolta** (Vrije Universiteit Amsterdam, the Netherlands)
- **Taher A. Ghaleb** (School of Electrical Engineering and Computer Science, University of Ottawa, Canada)
- **Istvan David** (Université de Montréal, Canada; Vrije Universiteit Amsterdam, the Netherlands)
- **Jasper van Rooijen** (University of Twente, the Netherlands)
- **Marielle Stoelinga** (University of Twente and Radboud University, the Netherlands)

ABSTRACT

Several mobile apps have been released to the public in response to the COVID-19 pandemic. The majority of these apps share a similar socio-technological context: they are developed under a tight schedule, with immense social and political pressure, e.g., concerning privacy and security. This pressure can lead to malfunctions with serious consequences, considering the mission-critical nature of these apps.
In this paper, we assess the severity of these factors by comparing 61 COVID-19 apps with 61 traditional (non-COVID-19) health and medical apps on the Android platform. Our analysis reveals several noteworthy differences, such as restrictions on operating system versions, and significantly more software bugs and code smells in COVID apps, directly threatening the utility of the app, e.g. in terms of reach, reliability, and performance.

1. Introduction

The COVID-19 pandemic is being fought with a wide variety of measures [13], including health measures (e.g., disinfection and vaccination) and social measures (e.g., lockdowns and quarantines), in addition to technological measures. In the latter category, several mobile applications have been released to the public in response to the COVID-19 pandemic. Apps belonging to this category (referred to as COVID-19 apps) include: contact tracing apps, apps to inform people about facts, treatments, and procedures related to the pandemic, apps supporting the COVID-19 response, containment, or research, efforts, and apps providing additional services to respond to COVID-19 [1].

The development of COVID-19 apps is hindered by various challenges. Key concerns are the trust and the wide adoption of these apps. For example, contact tracing apps are only useful if they are widely adopted: to get a relevant overview of the spreading of COVID, enough people must install tracing apps on their mobile phones. In particular, the correct functioning of the apps is critical: identifying too few COVID-19 contacts, or erroneous contacts would defeat the whole purpose of the app. Furthermore, security and privacy are major challenges in COVID-19 apps, as they record very privacy-sensitive medical information, as well as location data [2]. These challenges, in turn, threaten the adoption of COVID-19 apps, and thereby their effectiveness. Finally, the urgency of the pandemic requires immediate solutions, which lead to time pressure that can affect the overall technical quality.

In this article, we provide (i) **an overview of the differences between COVID-19 apps and non-COVID-19 apps** on the Android platform in terms of their platform compatibility, requested privacy-related permissions, used software components, and presence of bugs/code smells; and (ii) **two main concerns and six useful suggestions for app developers working on projects facing similar challenges to those of COVID-19 apps**, i.e., a tight schedule, immense social and political pressure, and with potentially severe consequences in case of malfunctions (e.g., apps responding to natural disasters).

In this study, we focus on Android apps since, as of today, Android covers 72.83% of the market share. In addition, the availability of open-source tools for analyzing Android apps and the ease in which apps can be mined from the Google Play store make the whole study replicable and independently verifiable.

Our analysis reveals two main concerns that app developers should address when developing disaster-management apps and six suggestions for app development:

- **Concern A: Facilitate user onboarding in the context of disaster situations.**
  - **Suggestion A1**: keep the required minimum SDK version low, to include as many users as possible, even those with older devices.
  - **Suggestion A2**: keep the set of requested permissions as tight as possible to avoid any suspicion of privacy concerns and lower the barrier for users to install the app.
  - **Suggestion A3**: clearly declare the permissions required and data collected by the app to be fully transparent with both prospective and already-onboarded users.
- **Concern B: Retain users in the context of disaster situations.**
  - **Suggestion B1**: allow disabling/enabling of privacy-related permissions from the app itself, rather than at system level, to ease privacy management for users.
  - **Suggestion B2**: comply with guidelines and norms about component structure to (i) avoid user disengagement due to the lack of trust and (ii) minimize the entry barriers for newly joining developers.
  - **Suggestion B3**: increase the frequency of releases to deliver bug fixes with high priority and urgency to maintain the trust of users.

These results are gained by a thorough comparison of 61 publicly available COVID-19 apps, and 61 traditional (non-COVID-19) apps from the health and medical domains. We analyze each app with respect to four key characteristics:

- **Reach:** The *platform compatibility* of the apps.
- **Privacy:** the number, type, and protection level of the requested user *permissions.*
- **Software components:** the number and type of software *components*.
- **Software quality:** *bugs*, code *smells*, and *code duplication*.

Paper organization. In Section 2, we explain the experimental setup. Sections 3 through 6 discuss our findings and the suggestions for developers. Section 7 concludes the paper.

2. Experimental setup

We built on an open-source tool (https://github.com/iivanoo/covid-apps-observer) that automatically collected Android mobile apps that are made available to the public of 19 countries. For each country, the tool queried the Google Play Store to search for all mobile apps that had the keyword '*covid*' and collected all mobile apps appearing in the search results of each country. We note that the Google Play Store enforces strict requirements on the apps claiming to be responding to COVID-19 [1] and filters its search results accordingly. The search strategy of our tool made sure that only officially-verified COVID-19 apps are analyzed in this study, i.e., apps that are (i) either published, commissioned, or directly endorsed by an official government entity or (ii) successfully passing a thorough review process performed by the maintainers of the Google Play store (https://support.google.com/googleplay/android-developer/answer/9889712?hl=en).

Our search strategy resulted in a total of *61 COVID-19 mobile apps across 19 countries*. After that, we collected non-COVID-19 health/medical mobile apps to perform a comparative analysis with the COVID-19 apps. We chose medical/health apps to carry out a reasonably fair comparison. Indeed, in the Google Play store, medical/health apps belong to those categories that are the closest to COVID-19, reasonably isolating the effect that COVID-19 could have on their development process. Specifically, for each COVID-19 app, we collected an app that is (a) under the *'Medical'* or *'Health & Fitness'* app category and (b) available in the same country(ies). As a result, we obtained 61 non-COVID-19 mobile apps. An analysis for each of the COVID-19 and non-COVID-19 apps was carried out in the following steps.

1. We extracted the metadata of the apps (e.g., title, description, size, and date of release) using a Google Play scraper.
2. We downloaded the Android Package (APK) of each app.
3. We extracted information on four key characteristics with the following tools:
    a. *Reach:* We extracted the AndroidManifest.xml file of the apps to identify the Android version(s) the apps support.
    b. *Privacy:* We extracted the user permissions requested by the apps using `Androguard` (https://github.com/androguard).
    c. *Software components:* We extracted the main components of the apps, i.e., activities, services, broadcast receivers, and content providers using `Androwarn` (https://github.com/maaaaz/androwarn).
    d. *Software quality:* We extracted quality metrics of the software within the apps using `SonarQube` (https://github.com/SonarSource/sonarqube).
4. We explored and analyzed the results via a combination of descriptive statistics, bar plots, and boxplots.

For independent replication and verification, the raw data collected for this study and the source code used to analyze it are publicly available on a dedicated GitHub repository (https://github.com/S2-group/covid-apps-analysis).

SIDEBAR - Other research studies on COVID-19 apps

The prominence of contact tracing apps (CTAs) gave rise to a novel class of challenges. A substantial amount of work has been dedicated to mapping these challenges from technological, societal and political standpoints.

Rahman and Farhana [1] identify the user interfaces and the data management layers of apps as the main hotspots of technical issues. Bug reports related to these components accounted for more than 60% of the bugs in their assessment of 129 COVID-19 CTAs. In general, the main non-functional concerns for CTAs have been security and privacy [2, 3, 4, 5]. Additionally, more general characteristics have been discussed by Samhi et al. [6].

Much research has been done on the econo-political context of CTAs. Bano et al. [7] demonstrate that, despite the sound technological underpinnings and the employment of the best practices of software engineering, political factors and individual behavior patterns often prevent the success of CTAs. Wang et al. point out that technical and societal issues equally prevail in government-backed applications as well [8]. Blasimme and Vayena [9] suggest that adaptive governance models enabling social learning can alleviate these issues and foster the usage of CTAs.

[1] Rahman et al. "An Exploratory Characterization of Bugs in COVID-19 Software Projects." arXiv preprint arXiv:2006.00586 (2020).
[2] Hatamian et al. “A privacy and security analysis of early-deployed COVID-19 contact tracing Android apps.” Empir Software Eng 26, 36 (2021).
[3] Sun et al. “Vetting security and privacy of global covid-19 contact tracing applications.” arXiv:2006.10933
[4] Raskar et al. ”Apps gone rogue: Maintaining personal privacy in an epidemic.” arXiv:2003.08567 (2020).
[5] Cho et al. "Contact tracing mobile apps for COVID-19: Privacy considerations and related trade-offs." arXiv preprint arXiv:2003.11511 (2020).
[6] Samhi et al. ”A First Look at Android Applications in Google Play related to Covid-19.” Empirical Software Engineering 26: 57 (2021).
[7] Bano et al. "Requirements, Politics, or Individualism: What Drives the Success of COVID-19 Contact-Tracing Apps?." IEEE Software 38.1 (2020): 7-12.
[8] Wang et al. “Market-level Analysis of Government-backed COVID-19 Contact Tracing Apps.” Int. Conf. on Automated Software Engineering - Workshops. (2020)
[9] Blasimme and Vayena. "What's next for COVID apps? Governance and oversight." Science 370.6518 (2020): 760-762.

3. Characteristic 1: Reach

Motivation. Android developers have to explicitly declare the specific versions of the Android platform used by their apps so that (i) the Google Play Store can show to users only the apps that are compatible with their devices and (ii) only compatible apps can be installed on the device of the user. Doing so prevents runtime issues due to the mismatch between the system calls made by the app and the running operating system, depending on the features an app provides.
As of November 2020, there is still a non-negligible portion of the Android user base who is running old versions of Android, such as Android Lollipop (6.1%, released in 2014) and Android KitKat (2.5%, released in 2013). Given their social and safety function, it is fundamental that disaster-management apps are compatible with as many as Android smartphones.

Analysis method. We targeted the Android Manifest file and extracted the contents of the Android SDK versions used by each app. Specifically, developers specify the Android version by means of an API level integer that maps one-to-one to a specific version of the Android OS and ranges from

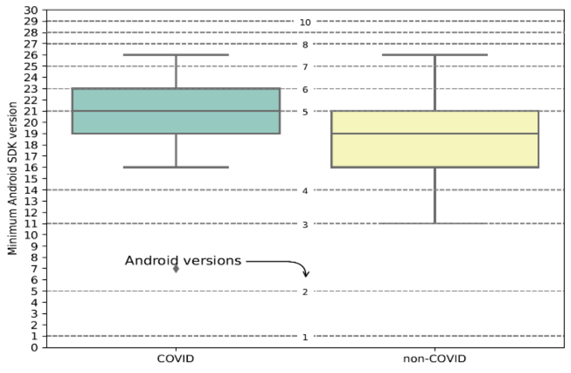
1 (corresponding to the first release of Android) to 30 (corresponding to Android 11). We extracted the following three relevant SDK versions declared in each app:

- *Minimum Android SDK version*: the minimum API Level required for the app to run. A smartphone prevents the installation of apps with a minimum SDK version higher than the Android version running onboard.
- *Maximum Android SDK version*: the opposite behavior of the minimum Android SDK version.
- *Target Android SDK version*: the API level that the developer used to test the app. If the smartphone of the user is running a different version of Android, then such differences are compensated at runtime by the OS.

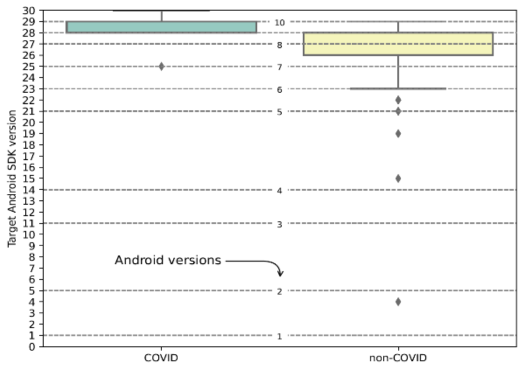
The larger the difference between the minimum and maximum Android SDK versions of an app, the wider the audience the app can reach. For example, old phones support only lower Android versions (e.g., SDK version 15). Hence, an app with a higher minimum supported Android version cannot be used by users with such old phones, which leads to accessibility problems. Android guidelines suggest to have the target Android SDK version as high as possible to ensure that apps behave and look as good as possible on the most recent Android devices.

Results of the analysis. First of all, it is interesting to observe that none of the COVID-19 apps define the *maximum* Android SDK version. This finding is promising, since those apps will be compatible with all future releases of the Android platform.
However, the data looks different when considering the *minimum* Android SDK version. As shown in Figure 1(a), the median value for non-COVID-19 apps is 19 (Android 4.4, released in 2013), whereas the median value for COVID-19 apps is 21 (Android 5.0, released in 2014).
Although COVID-19 apps are compatible with more Android devices, more than half of the apps are compatible only with Android versions released from 2014. In general, for Android developers, a higher minimum Android SDK version means having fewer issues at run-time and fewer corner cases to manage at development time.
The situation about the target Android SDK version looks much better (see Figure 1(b)), since COVID-19 apps have been tested with the latest Android releases (SDK version 28 and 29). This result is not surprising, since the need for developing COVID-19 apps only arose at the beginning of the pandemic (February-March 2020), where Android 10 was already released (SDK version 29).

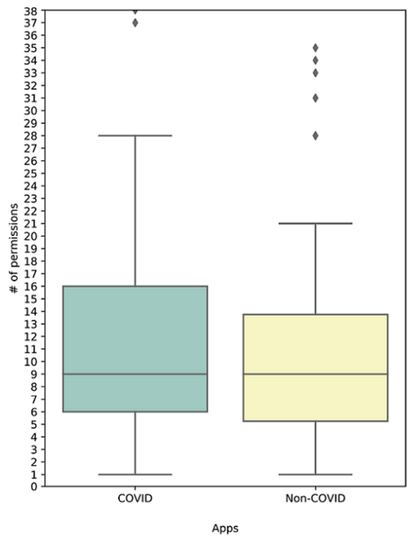
Suggestions for developers. The results about the maximum and target SDK versions are positive and do not raise any warning. However, the data about the minimum SDK version is, in our opinion, worrisome. Given that half of COVID-19 apps have a minimum target SDK version equal to Android 5.0 and that the number of users having an Android version earlier than Android 5.0 is 3.2% of the total number of Android users (about 2.8 billion), there are 89.6 million users who cannot even install a COVID-19 app today. Besides, there are several apps whose minimum Android SDK version is higher than the API level of 21, whereas some apps that require an API level of 26, which corresponds to Android 8.0, released only in 2017.
Based on these results, we suggest Android developers strive towards expanding the target user base of their disaster-management apps by lowering their required minimum SDK version to include those users having older devices (Suggestion A1). Android versions that are running on older or second-hand devices are more likely to be used by people with low-income and the elders, which are sadly the population segments having higher rates of COVID-19 infection and mortality [6].



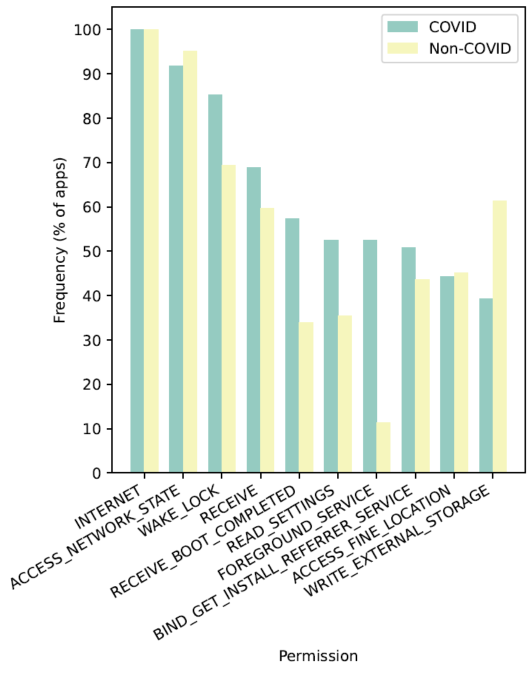
(a) Minimum SDK versions



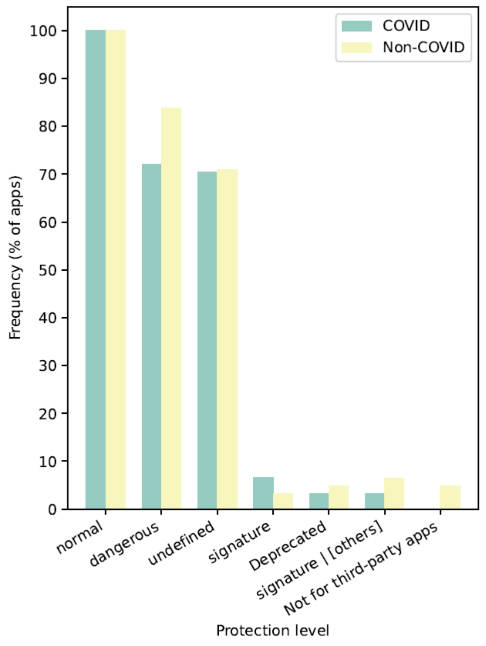
(b) Target SDK versions

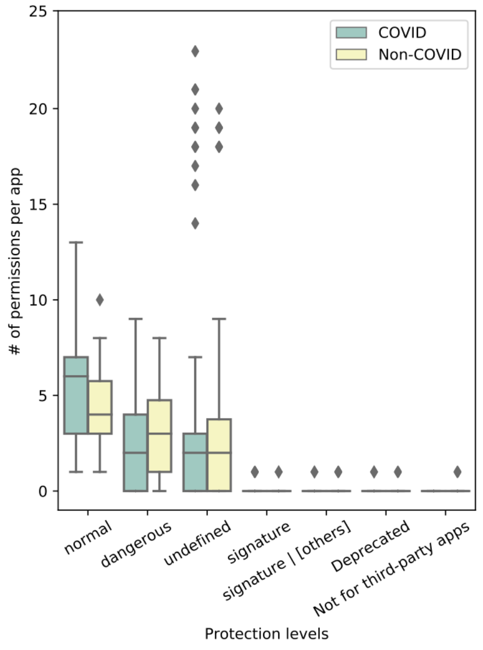


(c) Permissions requested per app



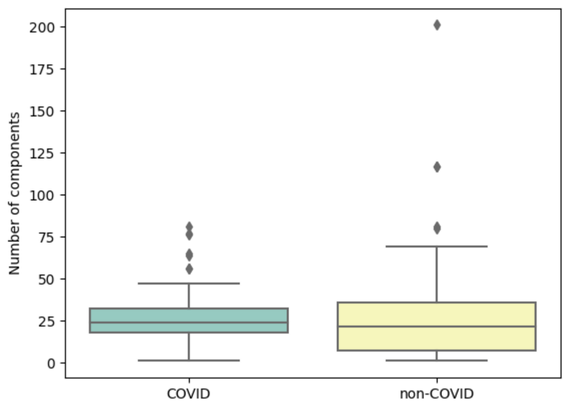
(d) Top ten requested permissions

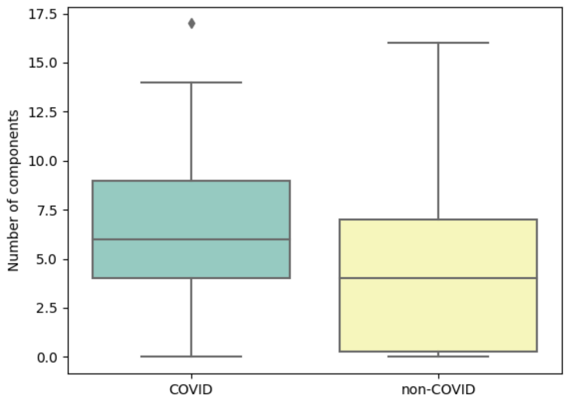




(e) Apps per protection levels

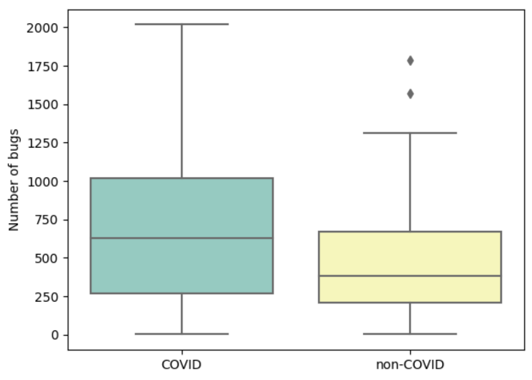
(f) Permissions per app of each protection level

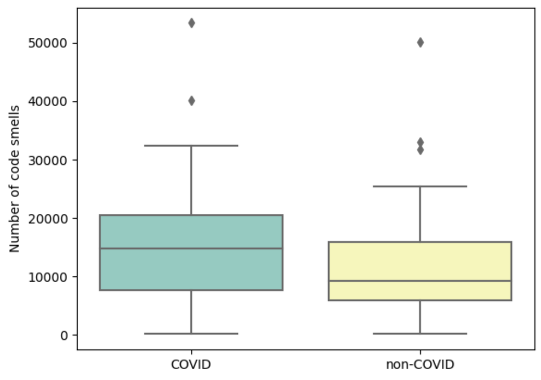




(g) Number of components per app

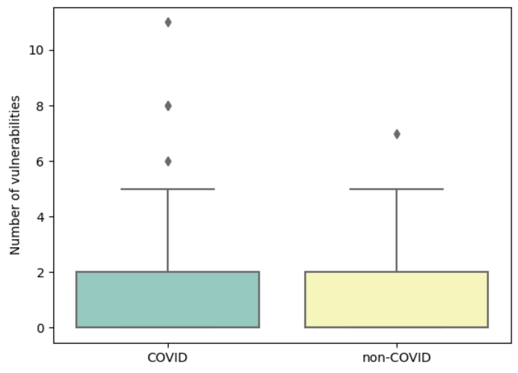
(h) Number of broadcast receivers

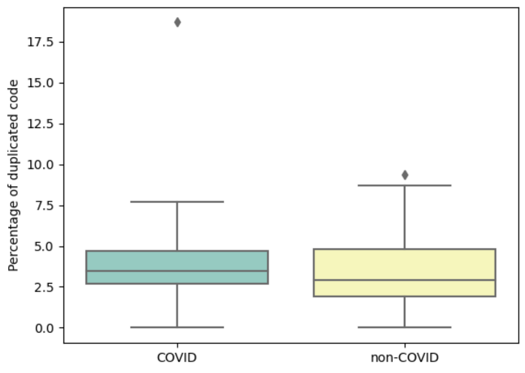




(i) Number of bugs per app

(j) Number of code smells per app





(k) Number of vulnerabilities per app

(l) Percentage of duplicated code per app

Figure 1. Relevant characteristics of COVID-19 and non-COVID-19 apps

4. Characteristic 2: Privacy

Motivation. A disaster-management app may request users to permit the app to access a certain functionality (e.g., Bluetooth) or personal data records (e.g., call history) on their smartphones. While certain app functions may not work properly if no permission is granted, prior research [2] has reported that COVID-19 apps (e.g., contact tracing apps) tend to be overprivileged [3], i.e., apps requesting permissions that are unrelated to the app functionality. Yet, it is unclear whether this phenomenon is common on officially-verified COVID-19 apps and ordinary health and medical apps.

Analysis method. We used the user permissions extracted from each app. We performed a comparative analysis between COVID-19 and non-COVID-19 apps to investigate how different are the (a) number of permissions requested by the apps, (b) the most frequently requested permissions, and (c) the permissions that COVID-19 and non-COVID-19 apps do not have in common. We performed a Mann-Whitney U test to measure the difference in app permissions between COVID-19 and non-COVID-19 apps.

Results of the analysis. *App-level***:** As shown in Figure 1(c), the number of permissions requested by COVID-19 apps do not significantly differ from the permissions requested by non-COVID-19 apps (p-value=0.22 and a median of 9 permissions per app).
*Permission-level***:** Our permission-wise analysis reveals that the frequency of COVID-19 apps that request a certain permission is not significantly different from the frequency of non-COVID-19 apps that request that permission (p-value=0.43). We identified a total 87 unique requested permissions. COVID-19 and non-COVID-19 apps request 54 (62%) permissions in common. Figure 1(d) shows a bar chart showing the top ten permissions requested by COVID-19 apps along with the percentage of apps that request those permissions. There, *INTERNET*, *ACCESS_NETWORK_STATE*, *WAKE_LOCK*, and *RECEIVE* are the top three commonly requested permissions by both COVID-19 and non-COVID-19 apps.
*Protection-level***:** Figure 1(e) shows the most commonly used protection levels. We observe that the majority of COVID-19 and non-COVID-19 apps request both *normal* (non-risky) and *dangerous* permissions, in addition to permissions with *undefined* protection level (according to the Android documentation). Moreover, we observe that COVID-19 apps tend to request fewer *dangerous* permissions than non-COVID-19 apps (see Figure 1(f)).

We identified 13 permissions that are only requested by six COVID-19 apps. We also identified 20 permissions that are requested by non-COVID-19 apps, but not COVID-19 apps. We analyzed the permissions requested by COVID-19 apps only to investigate their protection levels. We found that the majority of those permissions do not require an explicit approval by users. Here is a breakdown of the permissions that only exist in COVID-19 apps:

- seven permissions are of a *normal* protection level, i.e., minimal risk to other apps, the system, or the user;
- three permissions are of a *signature* or *system/signature* protection level, i.e., granted to certificate apps;
- two permissions are likely to be written incorrectly;
- one permission with *undefined* protection level.

Suggestions for developers. Users may not install an app if they suspect any privacy concerns [5]. Hence, developers should keep the set of requested permissions as tight as possible (Suggestion A2), specifically by avoiding to request (a) an extensive number of app permissions, (b) permissions that are unrelated to the app functionality, (c) permissions of high privacy risks, or (d) permissions that are uncommon in health/medical apps.
Moreover, users expect that disaster-management apps collect nothing but physical proximity data [4], in addition to the basic services, such as accessing the Internet. Thus, an app is expected to access the Bluetooth/WiFi/GPS devices to be able to identify the user's location data. In addition, users might not be aware of the 'normal-level' permissions that an app requires, since such permissions are granted by the system without user acknowledgment. Therefore, to facilitate users

onboarding, developers of disaster-management apps are encouraged to clearly declare in the app description what permissions are required by the app to function properly (Suggestion A3). In this way, developers are more transparent with users about the permissions required by their disaster-management apps and which data is to be collected. Developers can do so by explicitly highlighting such information in the app description, i.e., available prior to app installation, or at the welcome screen of the app.
Finally, to retain already-onboarded users, developers should make it easy for users to disable/enable a certain permission from the app itself or provide in-app links to allow users to navigate directly to the permissions of the app in the system settings (Suggestion B1).

5. Characteristic 3: Software components

Motivation. The time pressure in developing disaster-management apps could affect the structure of such apps. Android apps are composed of four types of components, namely activities, services, broadcast receivers, and content providers. This information can be extracted from the manifest of the apps, and may shed light on differences or the lack of thereof between disaster-management apps and traditional apps.

Analysis method. We analyzed the manifest data previously extracted using Androwarn. We investigated the number of such components in COVID-19 apps, and in a selection of general, non-COVID-19 apps, as a reference; and compared these numbers to investigate any relevant differences. Specifically, we performed Mann-Whitney U tests between the number of components of COVID-19 and non-COVID-19 apps to identify any statistically significant differences.

Results of the analysis. The Mann-Whitney U test between the sum number of components in COVID-19 and non-COVID-19 apps yields a p-value of 0.15, which suggests that the null hypothesis of the test cannot be rejected (at α=0.05), i.e., there is *no significant difference* between COVID-19 and non-COVID-19 apps in their respective numbers of components. We have found an average of 28.33 components in COVID-19 apps, and 30.65 components in non-COVID-19 apps. This difference is indeed not significant considering the variances in the two groups. The box plot in Figure 1(g) provides visual evidence to the indifference between COVID-19 and non-COVID-19 apps.

The Mann-Whitney U test between the specific four types of components, however, revealed a significant difference in the case of the broadcast receivers. As shown in Figure 1(h), we observe that the median number of broadcast receivers of COVID-19 apps is higher than that of non-COVID-19 apps: 6.7 and 4.6, respectively. The Mann-Whitney U test yields a p-value of 0.0016, strong evidence against the null hypothesis, i.e., there is a significant difference between COVID-19 and non-COVID-19 apps.

Suggestions for developers. While broadcast receivers serve the purpose of integrating an app into the larger ecosystem of apps, an abundance of broadcast receivers can be a pattern of malicious software [7]. Extreme situations, such as a global pandemic, may motivate users to remain flexible and introduce a trade-off between the development time of the app and its security and privacy preserving mechanisms [9]. Hence, we suggest developers of disaster-management apps to comply with guidelines and norms about component structure to (i) avoid user disengagement due to the lack of trust and (ii) keep low the entry barriers for newly joining developers, thus managing the fluctuations of available development effort (Suggestion B2).

6. Characteristic 4: Software quality

Motivation. It is of paramount importance for Android developers to promptly identify and fix possible bugs, code smells, and vulnerabilities from the source code to avoid crashes, data inconsistencies, or even data loss [9]. While achieving high quality software requires several iterations and a deep reasoning on various technical design decisions, developers of disaster-management apps will likely work under high pressure by both policy makers and society in general. For example, in July 2020, Japan's health authority had to temporarily remove its official COVID-19 app from app stores due to a bug preventing infected users from entering critical information to notify other users who had close contact with them. In the following, we investigate if working under such high social pressure could impact the software quality of COVID-19 apps.

Analysis method. We measured the software quality using SonarQube, a commonly used static code analyzer. To run SonarQube, we decompiled all apps by using dex2jar and JD-Core, two widely-used off-the-shelf tools [9]. Finally, we analyzed each app via the standard Java quality profile of SonarQube, which contains 625 rules, organized into four groups: bugs, vulnerabilities, code smells, and code duplication.

Results of the analysis. *Bugs:* SonarQube supports 152 different types of bugs, ranging from unclosed I/O resources, classes compared by name, etc. With a median of 628 bugs per app, COVID-19 apps have a higher number of bugs than non-COVID-19 apps, which have a median of 382 bugs per app (see Figure 1(i)). Specifically, the most recurrent bugs in COVID-19 apps are: (1) ignoring the initial values of method parameters, caught exceptions, and `foreach` variables (8,155 occurrences), (2) dereferencing potentially `null` pointers (7,286 occurrences), and (3) re-assigning variables to themselves (6,540 occurrences).
Code smells: Although code smells do not prevent apps from functioning, they could negatively impact software maintenance. As shown in Figure 1(j), the median number of code smells in COVID-19 apps (14,735) is higher than that of non-COVID-19 apps (9,222). The most recurrent code smells in COVID-19 apps are: (a) not complying to naming conventions (72,353 occurrences), (b) having unused `private` fields (40,496 occurrences), and (c) declaring methods or field with the same name or different only by capitalization (38,494 occurrences). While higher numbers of code smells are expected, since code smells are not bugs, they would likely make Android apps suffer from maintainability issues in the future, which tends to grow over time [8].
Vulnerabilities: Both COVID-19 and non-COVID-19 apps have very few vulnerabilities (Figure 1(k)). This result indicates that Android developers tend to pay attention to security-related issues of their apps in the e-health domain (both COVID-19 and non-COVID-19 apps).
Code duplication: Finally, we analyze the percentage of duplicated code within each app (Figure 1(l)). Code duplication is slightly higher in COVID-19 apps (median=3.96%) than in non-COVID-19 apps (median=3.34%). Code duplication is a frequent phenomenon in Android apps, mostly because of the activity-intent-based Android programming model [8]. Nevertheless, a higher percentage of duplicated code (as it is happening in COVID-19 apps) might lead to introducing more bugs and overlooking inconsistencies [8], which may negatively impact app maintainability in the future.

Overall, we can observe that *the quality of COVID-19 apps tend to be consistently lower than the quality of non-COVID-19 apps*, especially for bugs and code smells. This phenomenon can be explained by the time pressure associated with the development of COVID-19 apps. Indeed, prior research has shown that time pressure has a detrimental effect on code quality and that it leads to workarounds or compromises and minimal quality assurance [10].

Suggestions for developers. We suggest developers of disaster-management apps allocate sufficient time for the next releases of their apps and to pay careful attention to fixing existing bugs, since they can undermine the trust that end users place in the apps and, in turn, on their provided services (Suggestion B3). Losing the trust of end users may lead to a drop in the adoption of disaster-management apps, thus potentially jeopardizing the control of the disaster. Possible solutions for improving the overall quality of the apps include allocating more time for analyzing the requirements specification and documentation, routinely adopting code reviews, unit testing, and inspecting while taking into consideration the feedback provided by user app reviews.

7. Conclusions

This study is the first investigation on professionals developing software under the combination of the peculiar conditions due to the COVID-19 pandemic, such as working under a strong social and political pressure.
We formulate the emerging concerns and suggestions to make them generically applicable to disaster-management apps, hoping to help professionals working under similar conditions, such as those for responding to natural disasters. It is important to note that the set of concerns and suggestions emerging from this study is not meant to be exhaustive, but rather as a complement to already existing generic guidelines for Android development, such as those by Hatamian on privacy [11] and the ones by Verdecchia et al. on architecting Android apps [12].
We hope that our results will help professionals in (i) developing disaster-management apps with a higher level of quality and (ii) making better informed decisions with respect to the ones made during the COVID-19 pandemic.

References

[1] Google Play Store Support, “Requirements for coronavirus disease 2019 (COVID-19) apps” https://support.google.com/googleplay/android-developer/answer/9889712 (accessed July 22, 2021).
[2] Azad, Muhammad Ajmal, et al. "A First Look at Privacy Analysis of COVID-19 Contact Tracing Mobile Applications." IEEE Internet of Things Journal (2020).
[3] Felt, Adrienne Porter, et al. "Android permissions demystified." Proceedings of the 18th ACM conference on Computer and communications security. 2011.
[4] Salathé M, Cattuto C. (2020) COVID-19 Response: what data is necessary for digital proximity tracing? https://github.com/DP-3T/documents (accessed July 22, 2021).
[5] Gu J, Xu YC, Xu H, Zhang C, Ling H. Privacy concerns for mobile app download: an elaboration likelihood model perspective. Decis Support Syst. 2017;94:19–28.
[6] Oronce, C.I.A., Scannell, C.A., Kawachi, I. et al. Association Between State-Level Income Inequality and COVID-19 Cases and Mortality in the USA. J GEN INTERN MED 35, 2791–2793, 2020.
[7] F. Mohsen, H. Bisgin, Z. Scott and K. Strait, "Detecting Android Malwares by Mining Statically Registered Broadcast Receivers," *2017 IEEE 3rd International Conference on Collaboration and Internet Computing (CIC)*, pp. 67-76, 2017.

[8] Ivano Malavolta, Roberto Verdecchia, Bojan Filipovic, Magiel Bruntink, Patricia Lago. How Maintainability Issues of Android Apps Evolve. In IEEE International Conference on Software Maintenance and Evolution (ICSME), Madrid, Spain, pp. 334-344, 2018.
[9] William Enck, Damien Octeau, Patrick McDaniel, and Swarat Chaudhuri. A study of android application security. In Proceedings of the 20th USENIX Conference on Security (SEC'11), 2011.
[10] Kuutila, M., Mäntylä, M., Farooq, U., & Claes, M. Time pressure in software engineering: A systematic review. Information and Software Technology, 121, 2020.
[11] Hatamian, M. Engineering privacy in smartphone apps: A technical guideline catalog for app developers. IEEE Access 8 (2020): 35429-35445, 2020.
[12] R. Verdecchia, I. Malavolta and P. Lago. Guidelines for Architecting Android Apps: A Mixed-Method Empirical Study. IEEE International Conference on Software Architecture (ICSA), pp. 141-150, doi: 10.1109/ICSA.2019.00023, 2019.
[13] World Health Organization. Considerations for implementing and adjusting public health and social measures in the context of COVID-19: interim guidance, 14 June 2021. No. WHO/2019-nCoV/Adjusting_PH_measures/2021.1. World Health Organization, 2021.

About the Authors

Ivano Malavolta is an assistant professor at Vrije Universiteit Amsterdam, The Netherlands, and the Director of the Network Institute. His research focuses on data-driven software engineering, with a special emphasis on mobile software development, software architecture, model-driven engineering, robotics software, and empirical methods.

Taher A. Ghaleb (tghaleb@uottawa.ca**)** is a Postdoctoral Research Fellow at the School of EECS at the University of Ottawa, Canada. Taher holds a Ph.D. in Computing from Queen's University, Canada. His research interests include continuous integration, software testing, mining software repositories, applied machine learning, program analysis, and empirical software engineering.

Istvan David is a postdoctoral researcher at the University of Montréal, Canada. He received his PhD in Computer Science from the University of Antwerp, Belgium. His research focuses on the foundations and applications of collaborative modeling, inconsistency management, and multi-view modeling. He is also active outside of academia, mainly in innovation consulting.

Jasper van Rooijen, MSc holds a Bachelor and Master degree in Computer Science from the University of Twente, the Netherlands. He is currently employed at Thales.

Prof.dr. Marielle Stoelinga is a full professor of risk analysis for high-tech systems, both at the University of Twente and Radboud University, the Netherlands. She holds a Master's degree in Mathematics \& Computer Science, and a PhD in Computer Science, from
After her PhD, she has been a postdoctoral researcher at the University of California at Santa Cruz, USA. Prof. Stoelinga leads the executive Master on Risk Management at the University of Twente, a part time programme for risk professionals. She also leads various research projects, including a large national consortium on Predictive Maintenance and an ERC consolidator grant on safety and security interactions.