Privacy-Enhancing Development Environment

Tianshi Li

December 2022

Human-Computer Interaction Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Jason I. Hong (Chair), Carnegie Mellon University
Lorrie Faith Cranor, Carnegie Mellon University
Brad Myers, Carnegie Mellon University
Yuvraj Agarwal, Carnegie Mellon University (ISR)
Tadayoshi Kohno, University of Washington

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Keywords: Privacy, Human-Computer Interaction, Software Engineering, Development Environment, Privacy-Enhancing Developer Support

Abstract

Data has driven many technological advancements, while the ubiquitous collection and sharing of data have caused a privacy trust crisis in our society. Developers' nuanced understanding of their app's behavior and ability to adjust the app design put them in a critical role in making apps that respect the norms and users' expectations of data use. However, developers are not privacy experts. Developing a privacy-friendly app is often a challenging task due to their lack of 1) awareness of privacy issues, 2) knowledge of privacy best practices, and 3) time for handling privacy requirements. These problems have become more and more salient with the advent of a flurry of privacy requirements from platform providers (e.g., Google Play and Apple App Store) and laws (e.g., GDPR, CCPA), creating urgent needs for designing effective, opportune, and usable privacy support for developers.

Hence, I propose *Privacy-Enhancing Developer Support* as a new area of interest at the intersection of privacy, HCI, and software engineering research. The first challenge is that although there has been some research on developers' challenges for handling privacy requirements, they tend to be more descriptive than prescriptive. Therefore, our community still lacks a clear direction of how to solve the problems. To fill in this gap, I first synthesize developers' needs for designing privacy-enhancing developer support based on my work and past literature to provide a roadmap for future explorations into this problem.

Informed by the identified needs, I demonstrate my work that pioneers a novel type of developer tooling: *Privacy-Enhancing Development Environment*. I propose *privacy annotation*, a type of structured metadata that embeds privacy information such as data use purposes directly in code. Based on this concept, I designed and implemented three plugins for Android Studio, the official Integrated Development Environment (IDE) for Android development, to increase developers' awareness and knowledge of privacy best practices and to reduce the work required for complying with privacy requirements. With one set of annotations, my tools offer privacy support in multiple aspects, including 1) detection of sensitive API calls and third-party SDKs to support accurate understanding, documentation, and disclosure of data practices, 2) just-in-time reminders and lightweight code repair features (quick-fixes) to help developers conform to best practices, and 3) annotation-based declarative programming to generate in-app privacy notices and privacy nutrition labels required by app stores.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Data privacy has become an essential social problem of this era. A study by the Pew Research Center in 2019[1] showed that about six-in-ten Americans believed it was not possible to go through daily life without having their data collected by companies and governments, and a majority of Americans felt they had limited understanding and little control over the data collected. Meanwhile, a flurry of privacy scandals happened in recent years echoed the privacy concerns of the general public [48].

Consequently, many stakeholders have taken efforts to combat the privacy concerns in different avenues. Privacy laws such as General Data Protection Regulation (GDPR) and California Consumer Privacy Act (CCPA) were enacted to provide EU and California residents with better privacy protection. Platform providers like Google and Apple have rolled out privacy features for consumers and platform-level policies that impose privacy requirements for developers. Researchers investigated end-users' perceptions and concerns about privacy, which yielded design recommendations and novel technologies for helping users manage their privacy in everyday digital activities such as sharing accounts with partners or colleagues [135, 153], posting on social media [43, 169], or talking to a smart speaker [15, 37, 55].

Developers play an important role in complying with the legal and platform requirements for privacy and in following the design recommendations for building privacy-friendly apps from the ground up. However, as compared to the large body of research that aims to provide better privacy support for users, much less attention has been paid to developers' challenges and needs. Developers are often not privacy experts and may face various challenges for implementing a privacy-friendly app even if they sincerely care about user privacy. These problems have become more and more salient with the advent of a flurry of privacy requirements from platform providers (e.g., Google Play and Apple App Store) and laws (e.g., GDPR, CCPA), creating urgent needs for designing effective, opportune, and usable privacy support for developers.

Hence, I propose *Privacy-Enhancing Developer Support* as a new area of interest at the intersection of privacy, HCI, and software engineering research. The first challenge is that although there has been some research on developers' challenges for handling privacy requirements, they tend to be more descriptive than prescriptive. Therefore, our community still lacks

[1] https://www.pewresearch.org/internet/2019/11/15/americans-and-privacy-concerned-confused-and-feeling-lack-of-control-over-their-personal-information/

a clear direction of how to solve the problems. To fill in this gap, I first applied human-centered methods [122] to investigate developers' perceptions about privacy and their challenges when handling different privacy tasks [95, 98, 101, 160, 161]. Then I synthesize developers' needs for designing privacy-enhancing developer support based on my work and past literature to provide a roadmap for future explorations into this problem. The needs consist of three themes: 1) Improve accountability, namely to help developers set actionable goals for privacy and clarify and track their responsibilities; 2) Lessen Burden, namely to either offload or streamline privacy tasks for developers; 3) Increase Engagement, namely to remind developers to actively reflect on and improve their privacy practices and provide compelling incentives for voluntary adoption of privacy-enhancing tools. My developer studies and the three themes of developers' needs are summarized in chapter 3.

Informed by the identified needs, I demonstrate my exploration into a specific type of solution: *Privacy-Enhancing Development Environment*. I propose *privacy annotation*, a type of structured metadata that embeds privacy information such as data use purposes directly in code. Based on this concept, I designed and implemented plugins for Android Studio, the official Integrated Development Environment (IDE) for Android development, to increase developers' awareness and knowledge of privacy best practices and to reduce the work required for complying with privacy requirements.

The overarching design goal of my privacy-enhancing IDE plugins is that with one set of annotations, my tools can offer developers various types of privacy support, such as reminding developers of privacy issues and offering quick-fixes for the issues while programming (see my tool *Coconut* [95], detailed in chapter 4), automatically generating privacy user interfaces to enhance data transparency and control for users (see my tool *Honeysuckle* [100], detailed in chapter 5), and helping Android developers create accurate privacy nutrition labels which are required by the major mobile app stores (see my tool *Matcha*, detailed in chapter 6). The three tools have been built sequentially on top of the previous version, and I have released Matcha, which is the latest work of my thesis on the plugin store and has received move a hundred installs by real-world Android developers. With a unified privacy annotation, developers who install the plugin and annotate their code for creating privacy nutrition labels can also take advantage of the privacy features offered by the other two systems. In the following, I provide a brief overview of the three main pieces of work of my dissertation.

1.1 Thesis Outline

In this thesis, I first provide an overview of background and related work (chapter 2). Then I introduce my empirical studies that aimed to identify the challenges developers encounter while managing privacy requirements. The findings from these studies were synthesized into three types of needs for developer support (chapter 3). Then I describe the design, implementation, and evaluation of the three IDE plugins for addressing these needs (chapter 4, chapter 5, chapter 6). Lastly, I summarize the key contributions of my tool, including the novel concept of *privacy annotation*, and *privacy-enhancing IDE plugins*, the benefits they achieve through the tools that I built, and the potential of to build more tools for supporting the development of privacy-friendly software systems by breaking down privacy responsibilities into lightweight annotating tasks,

making privacy-enhancing considerations and actions part of the normal development process (chapter 7).

1.1.1 Coconut: An IDE Plugin for Developing Privacy-Friendly Android Apps

My research [95, 98] showed developers had passive attitudes towards privacy issues and had partial and vague understandings of privacy. These issues suggest the need for tooling to keep developers better engaged in protecting user privacy, which inspired my first work, Coconut, an IDE plugin to help developers build privacy-friendly Android apps [95]. Coconut detects sensitive API calls and helps developers add auto-filled privacy annotations. It offers real-time privacy suggestions and quick-fixes based on the privacy annotations. The privacy overview panel aggregates privacy information from annotations to help developers manage all data practices in one place. My developer study showed that adding privacy annotations were perceived as easy and helpful. The privacy suggestions effectively reduced excessive data use based on Android best practices. The privacy overview panel helped developers precisely recall their apps' data practices and write more accurate privacy policies.

1.1.2 Honeysuckle: Annotation-Guided Code Generation of In-App Privacy Notices

Developers' lack of privacy knowledge has resulted in low adoption of in-app privacy notice and dark patterns in privacy notice design [95, 100], suggesting the need for tooling to automate privacy tasks to mitigate developer's workload, such as designing and implementing privacy UIs. Hence, I designed and built Honeysuckle [100], a developer tool that leverages privacy annotations to generate code for multiple designs of privacy notice UIs. This approach also promotes the standardization of privacy notices because of the consistency of the generated UIs. Honeysuckle contains an IDE plugin subsystem based on Coconut and directly interacts with developers. In a lab study, developers created in-app privacy notices much faster with a significantly lower cognitive load using Honeysuckle's annotation-based code generation than manually writing API calls. Honeysuckle allows developers without expert privacy knowledge to create privacy notices following the best practices with little extra work. Having developers focus on what rather than how, Honeysuckle demonstrates a viable path to promote standard privacy notice designs, making it easier for users to compare privacy practices across apps.

1.1.3 Matcha: An IDE Plugin for Creating Accurate Privacy Nutrition Labels

The lack of knowledge about privacy requirements among developers makes it difficult for them to fully utilize their expertise in app development to meet those requirements. For example, as my CHI 2022 paper [101] shows, misunderstandings of label terminology and data practices among third-party SDKs can lead to widespread inaccuracies in self-reported privacy nutrition labels required by the Apple and Google app stores.

To help developers create accurate privacy nutrition labels, I designed, built, and evaluated Matcha, an IDE plugin that leverages the synergies between developers' knowledge and source code analysis to create accurate privacy nutrition labels for Android apps. By detecting sensitive API calls that may access and transmit personal data, Matcha guides developers to add privacy annotations to complement important details such as purposes and backend storage practices that cannot be inferred automatically. Matcha then translates the annotations to the label. My lab studies with Android developers working on a real-world app that they developed showed that Matcha improved the accuracy of data safety labels than using the Google Play console, and all developers preferred Matcha to the developer console. I have open-sourced Matcha[2] and released it on the JetBrains official plugin marketplace and have seen the plugin gaining adoption by real-world developers.

[2] https://matcha-ide.github.io

Chapter 2

Background and Related Work

Portions of this chapter were adapted from my published papers:

Li, Tianshi, Yuvraj Agarwal, and Jason I. Hong. "Coconut: An IDE plugin for developing privacy-friendly apps." Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies 2.4 (2018): 1-35.

Li, Tianshi, et al. "Honeysuckle: Annotation-guided code generation of in-app privacy notices." Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies 5.3 (2021): 1-27.

This chapter summarizes the background and related work that motivated and inspired my dissertation research in the following structure. First, I introduce prior literature that identified privacy issues for everyday users. I also mention users' current coping strategies and their limitations, alluding to opportunities that better engagement of developers may generate better solutions. Then, I summarize prior research that proposed design or technological methods to better protect users (Section 2.2) and recent efforts from legislature and platforms (e.g., Google and Apple) for regulating developers' use of sensitive data (Section 2.3). As I discuss their benefits, I also highlight the inherent limitations of the two directions due to the lack of input from developers and the lack of scaffolding for developers. Finally, I summarize prior research on developers' practices and challenges when dealing with security and privacy requirements and discuss existing developer support for security and privacy. Most of them are focused on the security side of the problem, which has very different needs from privacy.

2.1 Privacy Issues for Everyday Users

Privacy is a crucial challenge for our society. According to a recent study conducted by Pew Research Center [1], roughly six-in-ten Americans believe "it is not possible to go through daily life without having their data collected" and over eighty percent of Americans felt they had little/no control over their data collected by companies and the government. To regain users' trust, it is necessary to establish a thorough understanding of the main privacy issues for everyday

users and how users cope with these issues. In this section, I discuss three main issues: violations of users' expectations, tensions between privacy and usability, and issues with privacy notices and controls.

2.1.1 Data Collection Violating Users' Expectations

Users need to have a clear understanding of how their data is collected, used, and shared and have a correct estimate of the corresponding privacy risks. However, this task can be very challenging for users for various reasons.

According to the contextual integrity theory [127], the legitimacy of data flows depends on the contexts characterized by an array of attributes, including the data subject, the data type, the sender, and receivers of the data, and the transmission principles. This means that users need to know a lot of detailed information, including but not limited to what data is collected, for what purposes, how the data is processed, whether it is retained and associated with personally identifiable information (PII), and whether it is shared with any third parties. However, many users do not have sufficient technical literacy to understand all of these data practices. Kang et al. [84] conducted a study about people's mental models of the Internet and found that people without an education background in computer science or related fields had simpler models of the Internet and correspondingly perceived fewer privacy threats.

Using data for purposes different from users' expectations is a common type of violation of users' expectations for privacy. Lin et al. [106] introduced *privacy as expectations* as a model for privacy and observed via a crowdsourcing study that users' perceived necessity of an app accessing a specific type of data had a strong correlation with the users' perceived comfort level of granting the app access to the data. However, research has also shown that while some data collection may have a seemingly legitimate reason, they may also be used for other purposes that surprise users or even creep users out [106, 149].

The lack of awareness for the actual data practices has led to a myriad of high-profile privacy scandals. For example, the famous Facebook–Cambridge Analytica data scandal [7] can be attributed to several types of violation of expectations. First, when the user installed the personality quiz app, they may not realize that a lot of information, including that of their friends, was shared with the third-party app. Second, while they initially installed the app for taking personality tests, the data was later used for a different purpose – political advertising.

Note that the violation of expectations could happen for a seemingly legitimate data use. In 2019, Google [8] and Apple [2] both acknowledged that contractors could listen to audio clips collected by their voice assistants that may include sensitive information about the users. Although the goal was to improve the transcription quality, this type of data sharing practices still felt very surprising to general users, especially for people who are less familiar with the process of the development process of a machine learning model.

Lacking correct expectations of data practices has negative implications on users, including not being able to make informed decisions on whether to disclose sensitive information or grant access to sensitive resources. It may also have a negative impact on developers as users may choose to err on the side of caution and do not share data if they do not understand the reason for collecting it [34]. Developers have the best knowledge about how their apps handle users' data, so they are responsible for providing detailed disclosure about their privacy practices, especially for

the parts that are likely unexpected to users (e.g., the Google and Apple voice assistant example). However, as the complexities of data practices are brought to another level with the proliferation of AI technologies and digital devices equipped with rich sensors, this is not a simple task for developers.

2.1.2 Tensions Between Privacy and Usability

People often express their privacy concerns, but their actions suggest otherwise – this is known as the *privacy paradox* [91]. The tensions between privacy and usability are one reason that account for users' paradoxical behaviors. Users may be willing to trade off their privacy when the data is necessary to achieve goals that are beneficial to them. For example, my recent research about COVID-19 contact-tracing apps showed that the most popular design choice actually needed to collect more data in a centralized way and therefore more privacy-invasive than other solutions [96]. In a follow-up study, I more explicitly demonstrated that perceived benefits were a stronger motive for people to install the app than better privacy [97]. However, even if the users choose to allow data collection, it does not mean they do it comfortably. As Bonné et al. [34] showed in an experience-sampling study of Android users' decisions with the runtime permission requests, about 10% of the decisions of granting access were made reluctantly.

On the other hand, users sometimes have to trade off usability for privacy. Cobb et al. [43] found that users became app-dependent as they had to adjust their behaviors to manage what is displayed by the online status indicators, such as giving up using the phone just to make they look like they are asleep or at work. Sannon et al. [144] studied how, when, and why people lied online to protect their privacy. Similarly, Kang et al. [84] found users may simply limit or change information shared online to protect their privacy. Privacy concerns have also led to chilling effects [117].

To help users successfully navigate through the complicated trade-offs between privacy and usability, the developers need to build in more flexible and fine-grained choices for users. Unfortunately, users often have to make a difficult choice between privacy and usability. As shown in the controversial news about Google assistant [5], if a user did not want a contractor to listen to their audio recordings, the only way at that time was to to give up personalization.

2.1.3 Issues with Privacy Notices and Controls

Privacy policies are required by laws and are currently the essential way for users to learn about the privacy practices of digital services. However, it has long been known that privacy policies are lengthy, vague, full of legalese and hard to understand. In 2008, McDonald and Cranor [116] estimated that it would take an individual 244 hours a year if they wanted to read through the privacy policies of all the sites they visited and 154 hours a year if they skimmed at them. This level of cost means it is virtually impossible for users to gain a complete understanding of how a website or an app uses their data if privacy policies are the only available form of disclosure. In addition to the readability and understandability issue, prior research also uncovered the prevalent errors in privacy policies, including discrepancies between what are stated in the privacy policies and the actual data practices [23, 183] and internal contradictions [22].

For mobile apps, the operating system provides a permission system to protect sensitive resources such as location, microphone, and local storage. Users can see all the permissions requested by the app before installation, which provides them with a straightforward way to learn about what sensitive data may be collected by the app. However, Felt et al. [60] evaluated the Android permission systems and found that only 17% participants paid attention to the permissions during app installation and only 3% correctly answered all the comprehension questions. Because purpose is an important type of information that needs to be provided by developers, Apple requires iOS app developers to specify a purpose description for permissions requested by the app and Google also provides similar recommendations for developers. However, Liu et al. [111] showed only 25% of the sampled Android apps provided rationales. Furthermore, among the rationales provided by developers, a lot of them were either incorrect or did not provide further information than the default permission request message.

For mobile apps, there are many system-level privacy features for improving data transparency and control, such as the runtime permission requests. There are also other features introduced in more recent releases. For example, in iOS, an icon will show up on the notification bar when audio or location recording is ongoing. Another feature is to alert users if an app just copied data from their clipboard. The system also occasionally reminds users of background location access of a certain app [12]. Although these features do improve transparency, the lack of customization of what information is displayed and when and how it is displayed makes them inherently insufficient to accommodate the varying privacy preferences among different people [107] and under different contexts [106].

Some apps and websites provide internal privacy settings, while prior research has also identified numerous usability issues with them. One problem is their poor discoverability. Chen et al. [39] found in a user study that 47.12% of the privacy settings of the sampled apps were difficult to find and 9.64% could not be located by any participant. Their further analysis revealed the poor discoverability was mainly due to the problematic hierarchy of the apps' user interface and the confusing descriptions. Similarly, Habib et al. [69] studied the data deletion and out-opt choices displayed on 150 websites and found that privacy choices were located inconsistently across websites and may be confusing and difficult to use. These studies suggest that although the app-level or website-level privacy settings may provide more customized privacy controls, the usability issues made it difficult for users to take advantage of them. Besides, achieving standardization in the design and placement of these privacy settings may be a promising direction to improve their usability, which is later explored in my work *Honeysuckle*, a tool for generating privacy interfaces based on developer-specified annotations [100].

Another usability issue of the internal privacy settings lies in correctly configuring them. The default values often allowed for more data collection, and Krsek et al. [93] showed that reflective writing made Facebook users more likely to select more private configuration settings than the default settings. This suggests that developers should either choose more privacy-friendly default values or employ techniques to help users make the best choice that balance their need for privacy and utility.

All in all, current privacy notices and controls still fall short providing users with enough support for protecting their privacy. To make more helpful privacy notices and controls, we need to leverage developers' rich knowledge about how and why their apps collect user data and their ability to integrate contextualized, fine-grained privacy notices and controls. However, we also

need to regulate and standardize their implementation to ensure the privacy interfaces that they help generate are accurate, consistent, and usable. My tool Honeysuckle [100] mainly aims to tackle this problem (detailed in Chapter 5).

2.2 Research on Mitigation Approaches for Users

This section provides an overview of past research that proposed design or technological approaches to tackle the above privacy issues. Although most of these proposals have not been deployed on a large scale, they provide insights into the pros and cons of different methods and shed light on how to solve these privacy issues. I also discuss to what extent developers need to be involved to achieve the goal.

2.2.1 Designing Alternative Formats of Privacy Notices

To improve data transparency and control, researchers have proposed other formats of privacy notices beyond privacy policies and the permission systems.

Standardized Privacy Notices

Website privacy policies are well known for being long and difficult to read [80, 116]. To make it easier for users to quickly glean important information from those policies and to compare privacy practices between websites, researchers have explored multiple ways to offer privacy summaries that are standardized, succinct, and both human-readable and machine readable.

Privacy Preferences Platform (P3P) is an early exploration of this idea by the World Wide Web Consortium [45]. The goal was to let website developers specify their privacy practices in a standard machine-readable format, so users can specify their privacy preferences for one time and use their agents to automate their decisions every time they visit a website. However, this attempt was not successful largely because it depends on self-regulation and developers could provide incorrect information or simply do not prepare the policy file. The annotation idea I proposed and designed with in Coconut [95] (Chapter 4) and Honeysuckle [100] (Chapter 5) is to some extent in the same spirit of P3P, but I tried to increase the incentive for adoption by aligning it with existing privacy requirements and improve the accuracy by incorporating some program analysis to check developers' input.

Kelley et al. [87] proposed and evaluated a design for privacy nutrition labels for websites, drawing on lessons from the food nutrition labeling literature such as adopting a standardized and brief format. Kelley et al. [88] evaluated the proposed privacy label design, comparing it with shorter tabular and text variants as well as traditional long privacy policies in a large-scale randomized controlled trial. The researchers found that standardized labels could increase both speed of finding information and accuracy of users' comprehension. They found that privacy labels allowed users to better compare policies, and users found standardized formats more enjoyable to read.

In 2013, Kelley et al. [89] followed up with a privacy nutrition label design for mobile apps, demonstrating that labels presented clearly and at relevant times could affect users' decisions

when choosing between similar apps. Later Emami-Naeini et al. [56] proposed a privacy and security label for Internet of Things devices and showed it could help consumers incorporate privacy and security into their IoT device purchase decisions.

Researchers have also investigated how to maximize the benefits of privacy labels by exploring and evaluating design variants of privacy nutrition labels in multiple dimensions [32, 46, 56, 88, 120, 145, 155]. This line of work has yielded design recommendations for improving the design of privacy nutrition labels.

However, my recent work has shown that developers have numerous challenges in creating privacy nutrition labels [101] accurately and updating them on time [105]. Therefore, my proposed work aims to tackle this problem by designing and evaluating an approach to automating the privacy label generation process.

In-App Privacy Notices

Research in usable privacy has argued that effective privacy notices should be concise, contextualized, and emphasize unexpected data practices [66, 106, 138, 145]. I use the term in-app privacy notices to refer to notices that are well-integrated into the app's interaction flow and customized based on a deep understanding of the app's design and data practices.

However, designing effective in-app privacy notices is not a trivial task. Several common issues need to be taken into account to achieve a usable and useful design. For example, developers need to reduce the notice complexity by using concise description and plain language, mainly for two reasons. First, prior research has found that it takes a great deal of time to read through lengthy privacy policies [116] and understand the data practices and corresponding implications [59]. Second, the limited screen space constrains the amount of information that can be conveyed [145]. Furthermore, developers need to grapple with notice fatigue [145, 146] and habituation [85, 145, 146] by finding a balance between increasing transparency and decreasing disruption. A common approach to addressing these issues is to use multi-layered notices [145] that can first present data practices that are less expected by users initially and further present more information on demand [106]. However, there is no one-size-fits-all definition of unexpected data practices, and developers need to analyze their own app design and conduct user studies to understand the perceptions of different audiences [107, 108].

Prior research has proposed many designs of in-app privacy notices such as just-in-time privacy notices [44] and privacy dashboards that show the access frequency of specific data [20, 29]. My work Honeysuckle [100] supports annotation-guided generation of these features for Android apps (see details in Section 5.2.1).

2.2.2 Automated Assistance for Managing Privacy

Researchers and practitioners have also investigated ways to help users gain better control over their data. One type of these tools focus on avoiding third-party web tracking, including technologies for measuring online tracking behaviours [58] and detecting and preventing the use of cookie-based tracking [6], device fingerprinting [4], and other tracking techniques [141].

For mobile privacy, researchers and practitioners have also explored building privacy managers as third-party apps. Examples include monitoring data collection and alert users about a

data access [19, 42], and monitoring network traffic to alert users about potential leaks of sensitive personal data [140, 152, 154]. Some work has even demonstrated the feasibility to inferring the purposes of data use by analyzing the semantics of the decompiled code [168] or data traffic [81]. Despite the benefits of offering better privacy protection in a plug-and-play manner, these privacy manager apps also have limitations, including 1) performance degradation caused by the overhead of dynamic program analysis or running a VPN to intercept all network traffic, 2) imposing extra privacy risks when allowing a third-party app to monitor data flows and intercept network traffic (though this may have been addressed with native privacy monitoring tools such as the App Privacy Report introduced since iOS 15.2[1]), 3) automated analysis has limited ability in inferring fine-grained data usage information, and 4) potentially breaking or compromising the app's functionality when blocking the data collection or transmission, and anonymizing and obfuscating the data.

Overall, current technologies show great promises in helping protect users from malicious data use and third-party data use, while there is still a large room for improvement regarding helping users manage the first-party data collection practices for benign purposes. A main goal of my dissertation research is to study how to unleash the synergy between developer's knowledge about how their own apps and technologies that can analyze and infer data practices.

2.3 Regulation and Platform Requirements for Developers

In addition to protective techniques for users, we also witnessed a lot of efforts for regulating developers' privacy practices over the past five years. Below, I summarize some requirements of the two most impactful privacy laws (GDPR and CCPA) and privacy requirements from the two major mobile app stores.

2.3.1 GDPR

The EU General Data Protection Regulation (GDPR) went into effect on May 25, 2018. It is among the toughest privacy and security law in the world. Upon violations, the EU data protection authorities can impose fine up to €20 million. Organizations are subject to GDPR regardless of where they are located, as long as the collect or process data of EU residents. GDPR involves requirements that cover a wide range of data protection principles and privacy rights. To be GDPR-compliant, organizations must determine the proper lawful basis for data processing and document the basis and communicate them with the data subject. Some requirements are still fairly abstract, making compliance a difficult goal for small-to-medium-sized companies or independent developers.

While GDPR has placed more strict requirements on data controllers and processors, research has shown its limitations and enforcement challenges. For example, the rights for data portability may be misused as weapons to steal other users' data by issuing fraudulent requests. In addition, although many websites are trying to comply with GDPR requirements by requesting explicit consent for using cookies, many of the consent dialog designs employed dark patterns [33, 129] and caused a lot of hassles for users [50]. Prior research has also found violations of GDPR [126].

[1] https://support.apple.com/en-us/HT212958

2.3.2 CCPA

The California Consumer Privacy Act (CCPA) is a state statute to enhance privacy protection for California residents. CCPA went into effect on January 1, 2020. Similar to GDPR, it secures new privacy rights including the right to know, the right to delete, the right to opt-out, and the right to non-discrimination. Under the right to know, the CCPA requires businesses to give consumers certain information in a "notice at collection", including a specific requirement for a "Do Not Sell" link if the data may be sold. The CCPA also provides an opt-out icon [3] following recommendations based on user research [47, 70], which can be used by consumers to tell the company to stop selling their data.

2.3.3 Privacy Requirements from App Stores

The Apple App Store and the Google Play store have also announced a series of privacy requirements for developers that want to release apps on their platforms. Although they are not laws, they still have a significant impact on developers as failing to comply with them may cause the app to be banned from the app stores. Both app stores require apps to provide a privacy policy. Recently, these app stores announced further requirements for improving transparency. Drawing on the idea of privacy nutrition label originated from the academia [87, 89], the Apple App Store introduced App Privacy Details in Dec. 2020 and the Google Play Store introduced a forthcoming Google Safety Section in Feb. 2022. Both are required to be self-reported by developers.

2.4 Developer-Centered Research for Security & Privacy

Developers have an in-depth understanding of their apps' behaviors and are responsible for correctly handling privacy requirements throughout the app development process. However, developers are humans, too. They may be susceptible to insufficient abilities and cognitive biases when handling privacy requirements. In addition, they may have limited time for privacy due to the constraints of other factors such as app performance, communication with other colleagues, time pressure, and so on.

To design better privacy-enhancing support for developers, we need to treat them as users and employ human-centered research methods to understand their challenges [122]. This section summarizes prior research that investigated developers' practices and challenges for security and privacy and existing developer support for enhancing the security and privacy of their apps. Since prior research has shown a strong bias towards security-related studies and tools while having limited understanding about privacy issues, I conducted multiple developer studies focused on privacy, from which (and prior work) I synthesize needs for designing privacy-enhancing developer support in the next chapter (Chapter 3).

2.4.1 Developers' Practices and Challenges for S&P

To provide proper developer support, we need to understand what accounts for common privacy issues in the Android ecosystem. Research on this topic is growing, but is still at a relatively early stage and shows a strong bias towards security-related issues.

One fundamental issue relates to how much developers value privacy and how much they pay attention to it. Prior work has found that developers may make design decisions that trade off privacy for better usability or for features they would like to have [17, 28, 71, 112], which may be in contrast to users' actual preferences for more privacy guarantees [148]. Developers also feel that security is the responsibility of other parties [83, 142, 176], which suggests that developers who work independently or in an organization that cannot afford a specialized security team may pay less attention or have a harder time dealing with security issues.

Several researchers have investigated the gap between widely acknowledged principles for data privacy and how developers actually understand privacy. Although most of this work looks at developers outside of Android, the findings should be generalizable for our needs since the studies involved few platform-specific details. For example, Hadar et al. found that developers hold a partial understanding of privacy, mostly limited to security concerns, and the organizational privacy climate was an important factor playing into their perceptions of privacy [71]. Sheth et al. found that developers preferred data anonymization to using privacy policies to reduce privacy concerns, and there exists a large disparity between developers' and users' belief on the same issue [148]. This body of work suggests many developers have a vague and incomplete understanding of what may raise users' privacy concerns and what might actually compromise their privacy.

Some papers drill down into more technical details. One frequently reported issue was that developers may lack knowledge of potential privacy invasions and corresponding coping strategies. For example, many developers are not aware that some advertising and analytics third-party libraries automatically share users' personal data with service providers [30]. This is likely due to poor readability of the privacy policies of third-party services [28] and insufficient exposure to privacy guidelines [30]. A number of papers studied the limitation of existing programming models and developer support, such as program analyzers detecting security vulnerabilities [150, 173, 174], security APIs / personal data APIs [17, 18, 78, 170], and official documentation / other information sources for security [16, 17, 61]. However, most of this body of work focuses solely on security.

Understanding how developers comprehend privacy guidelines and how these principles influence implementation can be useful for informing the design of better developer tools. However, as noted earlier, current work mostly focuses on security, and offers little insight about other aspects of privacy like data retention, purpose limitation, and privacy notices.

2.4.2 Developer Support for S&P

Here, we review past work in developer support for privacy/security, focusing mostly on Android. We also discuss some gaps in these tools.

Documentation and Tutorials for Privacy/Security

Developer support for educating developers of essential privacy principles is primarily limited to privacy guidelines [25, 44, 73, 131, 132, 133, 134, 136, 163] or official documentation from the platform [52]. Although they are carefully designed to be comprehensive and written in plain language, Balebako et al. [30] found that developers still have a low awareness of them. Acar et al. [16] found that they were harder to use than informal information sources such as QA sites. These sources are also highly distributed, with advice spread out across many guidelines. Lastly, they tend to only offer high-level advice, such as only using sensitive data when necessary, which may not be easy to apply concretely. These issues indicate the need for a better form to educate developers of these privacy principles in practice.

Static Analyzers for Privacy/Security

Several program analyzers have been built to detect security vulnerabilities for Java and Android [40, 115, 124, 137]. However, past work found a lack of adoption of these analyzers [173, 174], suggesting that perceived importance of security and visibility of peer developers using these tools are important factors influencing adoption. Furthermore, these analyzers are mostly used by security experts rather than normal developers [164], echoing past findings that some developers tend to treat security (and likely privacy as well) as the responsibility of specialized teams [83, 142, 176]. There are also information flow analyzers which can detect malicious or unwanted information leaks from an app [24, 57, 67, 94, 130]. However, it is currently unclear how to map these leaks to specific privacy risks, let alone helping developers mitigate those risks.

IDE-Level Support for Privacy/Security

There are some existing IDE plugins for detecting security vulnerabilities in Android [9, 76, 125, 177], which can offer developers real-time feedback on security issues as they are coding, and in some cases even provide developers with a direct solution to fix the problem. There is also some IDE-level support for other privacy-related issues. For example, Android Lint tries to mitigate over-privileged data tracking behaviors by detecting hardware identifier usage, and suggests alternative and more privacy friendly choices like advertising ID or instance ID [9]. However, the amount of IDE-level support to help developers handle other privacy-related issues is much more limited than security issues.

Support for Enforcing Privacy Policy Compliance

Researchers have investigated how to automate compliance checking of privacy policies, much of which relies on manual review. For example, Bing has an internal system[147] that can automatically check code compliance with privacy policies. Researchers have also looked at new programming models for enforcing privacy policies by factoring out specification of security and privacy concerns from the rest of the program [178, 179]. In this line of work, privacy is handled in a top-down approach, with privacy policies first specified and then enforced in the implementation. In contrast, we tackle this problem in a bottom-up way, asking developers to think about

and annotate essential information about privacy practices when constructing code. My research showed that Coconut [95] may also help developers make more accurate privacy policies.

Support for Creating Privacy Policies

Privacy policies are required by some laws and by some app stores. Developers can use online privacy policy generators (e.g., freeprivacypolicy.com, generateprivacypolicy.com, appprivacy.net) to make privacy policies for their apps. These generators typically step developers through a series of questions, which are often too coarse to capture enough details of the personal data use. Furthermore, in our interviews we found that developers may not have an accurate and up-to-date understanding of their apps' data collection behaviors, suggesting that these tools may not be able to solicit reliable answers. Some programming models [103] are specifically designed to streamline analyzing how personal data is processed, which can be potentially used to help generate privacy policies. My tool Coconut [95] applies a similar idea by reminding the developer to add annotations about how the personal data is used, which could be more accessible to developers and may also promote the adoption of these privacy-friendly programming models.

Chapter 3

Needfinding Studies for Privacy-Enhancing Developer Support

Portions of this chapter were adapted from my published papers:

Li, Tianshi, Yuvraj Agarwal, and Jason I. Hong. "Coconut: An IDE plugin for developing privacy-friendly apps." *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies 2.4 (2018): 1-35.*

Li, Tianshi, et al. "Understanding Challenges for Developers to Create Accurate Privacy Nutrition Labels." *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems. 2020.*

Li, Tianshi, et al. "How Developers Talk About Personal Data and What It Means for User Privacy: A Case Study of a Developer Forum on Reddit." *Proceedings of the ACM on Human-Computer Interaction 4.CSCW3 (2021): 1-28.*

Tahaei, Mohammad, Tianshi Li, and Kami Vaniea. "Understanding Privacy-Related Advice on Stack Overflow." *Proceedings on Privacy Enhancing Technologies 1 (2022): 18.*

Tahaei, Mohammad, et al. "Charting App Developers' Journey Through Privacy Regulation Features in Ad Networks." *Proceedings on Privacy Enhancing Technologies 1: 24.*

Although there has been some work studying privacy issues from the developer's perspective, there is little work on the constructive side that aims to design solutions for these issues. To fill this gap, I took the first step to conduct studies that aimed to understand developers' perceptions, practices, and challenges that are related to privacy [95, 98, 101, 160, 161]. In this chapter, I first briefly overview my studies. Then I introduce design needs derived from the findings of my work and prior literature, synthesized into three themes. The related empirical evidence will be demonstrated when introducing each theme of design need for privacy-enhancing developer support.

3.1 Interviewing Android Developers About Creating Privacy-Friendly Apps

To inform the design of Coconut [95], I first conducted a formative study by interviewing nine Android developers. The goal is to understand the following: how they manage privacy in practice when developing apps, their general understanding of privacy, their mental models for some well-acknowledged data protection principles, and specific steps and practices used for addressing privacy issues. Balebako et al. [30] also conducted interview studies with mobile app developers to study their privacy and security behaviors, which examined developers' perceptions and behaviors regarding privacy using high-level questions. In contrast, I inquired specifically about how apps that our participants had recently developed used personal data, and mapped out these personal data practices to fundamental privacy principles to identify misconceptions and incorrect behaviors for privacy.

3.2 Interviewing iOS Developers About Creating Privacy Labels

Apple announced the introduction of *app privacy details* to their App Store in December 2020, marking the first ever real-world, large-scale deployment of the *privacy nutrition label* concept, which had been introduced by researchers over a decade earlier. The Apple labels are created by app developers, who self-report their app's data practices.

I conducted the first study examining the usability and understandability of Apple's privacy nutrition label creation process from the developer's perspective. By observing and interviewing 12 iOS app developers about how they created the privacy label for a real-world app that they developed, I identified common challenges for correctly and efficiently creating privacy labels.

3.3 Usability Studies About the Privacy Regulation Features for Ad Networks

Mobile apps enable ad networks to collect and track users. App developers are given "configurations" on these platforms to limit data collection and adhere to privacy regulations; however, the prevalence of apps that violate privacy regulations because of third parties, including ad networks, begs the question of how developers work through these configurations and how easy they are to utilize. My colleagues and I study privacy regulations-related interfaces on three widely used ad networks using two empirical studies, an expert review and think-aloud sessions with eleven developers, to shed light on how ad networks present privacy regulations and how usable the provided configurations are for developers.

We find that information about privacy regulations is scattered in several pages, buried under multiple layers, and uses terms and language developers do not understand. While ad networks put the burden of complying with the regulations on developers, our participants, on the other hand, see ad networks responsible for ensuring compliance with regulations.

3.4 Analyzing Discussions About Personal Data on r/androiddev

While online developer forums are major resources of knowledge for application developers, their roles in promoting better privacy practices remain underexplored. I conducted a qualitative analysis of a sample of 207 threads (4772 unique posts) mentioning different forms of personal data from the `/r/androiddev` forum on Reddit. I started with bottom-up open coding on the sampled posts to develop a typology of discussions about personal data use and conducted follow-up analyses to understand what types of posts elicited in-depth discussions on privacy issues or mentioned risky data practices. The results show that Android developers rarely discussed privacy concerns when talking about a specific app design or implementation problem, but often had active discussions around privacy when stimulated by certain external events representing new privacy-enhancing restrictions from the Android operating system, app store policies, or privacy laws. Developers often felt these restrictions could cause considerable cost yet fail to generate any compelling benefit for themselves.

3.5 Analyzing Privacy Advice on Stack Overflow

Privacy tasks can be challenging for developers, resulting in privacy frameworks and guidelines from the research community which are designed to assist developers in considering privacy features and applying privacy enhancing technologies in early stages of software development. However, how developers engage with privacy design strategies is not yet well understood.

My colleague and I look at the types of privacy-related advice developers give each other and how that advice maps to Hoepman's privacy design strategies. We qualitatively analyzed 119 privacy-related accepted *answers* on *Stack Overflow* from the past five years and extracted 148 pieces of advice from these answers. We find that the advice is mostly around compliance with regulations and ensuring confidentiality with a focus on the `inform`, `hide`, `control`, and `minimize` of the Hoepman's privacy design strategies. Other strategies, `abstract`, `separate`, `enforce`, and `demonstrate`, are rarely advised. Answers often include links to official documentation and online articles, highlighting the value of both official documentation and other informal materials such as blog posts.

3.6 Needs Synthesized from My Work and Prior Literature

I envision that privacy-enhancing developer support should ideally address these three themes of needs: 1) Codify Accountability; 2) Lessen Burden; 3) Leverage AgencyIn the following, I explicate each theme using concrete examples.

Note that these themes and analysis are still preliminary. A goal for my final dissertation work is to further consolidate these needs and publish them as a standalone paper.

3.6.1 Codify Accountability

Developers are accountable for privacy. However, they face challenges in operationalizing the abstract concept of privacy, having a clear understanding of their responsibilities, and contributing

Theme	Key Message
Codify Accountability	Set clear scope of responsibilities for developers and hold developers accountable for privacy.
Reduce Burden	Reduce the burden of privacy for developers
Leverage Agency	Guide and incentivize developers to actively find and fix privacy issues

Table 3.1: An overview of the three key themes of needs for designing privacy-enhancing developer support.

their knowledge of app behaviors to privacy audits. There is need for better scoping developers' responsibilities and studying how to hold developers accountable for privacy.

Operationalize Privacy

Privacy is a complex, abstract concept, whose definition has been discussed as a longstanding research questions for legal, social science, and computer science scholars. Therefore, it is not surprising that transferring the high-level goal of improving privacy to actual design and implementation tasks is only going to be more challenging for developers.

Prior research has already shown that developers tended to reduce the concept of privacy to security concerns [71]. Sheth et al. [148] mentioned that developers preferred data anonymization to using privacy policies to reduce privacy concerns. My interviews with Android developers further investigated this problem by probing developers' general perceptions of privacy [95]. I found that many participants only considered partial aspects, such as collecting data only when users have fully consent to it, preventing using identifiable information, minimizing data usage, or encrypting or obfuscating data before sending it out of the phone. Most participants have low awareness of privacy concerns related to data sharing and data retention.

This body of work suggests many developers have a vague and incomplete understanding of what may raise users' privacy concerns and what might actually compromise their privacy, which calls for the following design goal for developer support:

Need #1: Developers need actionable goals to tackle the abstract concept of privacy.

Clarify Division of Labor

Privacy involves various types of stakeholders, and sometimes it may get unclear who is responsible for which part. Mhaidli et al. [119] studied app developers' adoption of ad networks and showed that app developers considered ad networks responsible for addressing the privacy risks posed by ads. Therefore, developers keep the default settings in these ad networks without considering the impact on privacy risks and financial benefits. The same issue also manifested in our studies about the privacy regulation-related interfaces of three widely used ad networks [161]. In addition, my studies have repeatedly shown that some developers said they were unconcerned about privacy because it was not their responsibility [95, 101].

To leverage the unique values that developers can offer and also respect the limit of their abilities, a clear division of labor for privacy is needed. In addition to issues that have been relatively well studied in the past such as the relationship between developers and ad networks, further research is needed to analyze how to divide the labour between developers and other entities such designers, privacy engineers, lawyers, and managers, as well as more high-level parties such as app development platforms.

Need #2: Developers need a clear scope of privacy responsibilities.

Support Auditing

Privacy audits are required by laws like GDPR. Currently this type of task is usually led by lawyers and privacy or security compliance teams, while developers also need to participate in this process by filling out forms about how they implemented the actual data practices. However, my research has shown that developers could also have inaccurate understanding of app behaviors [95, 101] due to 1) insufficient understanding of how some API works, especially ad libraries, 2) having difficulty in tracking changes of data practices across different versions, and 3) lacking documentation of data practices when previous developers have left the team. Therefore, better tools need to be built for helping developers manage data practices and attribute privacy-related changes to specific developers.

Another problem is that current software systems often consists of subsystems written in different languages and using different SDKs. These subsystems may take care of different part of the data flows. For example, a mobile app may have a client app installed on the user device to collect data, a backend server that stores data, and also third-party services that it shares data with and gathers data from. The developers who build and maintain these subsystems may also be different people. To generate an accurate and attributable privacy audit report, it requires great interoperability across these subsystems. The privacy annotations proposed by my work shows a possible direction to solve this problem. These annotations are supported by many programming languages and can be embedded in code so developers can easily add them when coding a new feature. They follow a standard format, so it is also straightforward for each subsystem to generate a machine-readable summary using a standard form for the entire system.

Need #3: Developers need proper attribution of their privacy-related work.

3.6.2 Lessen Burden

Developers are not privacy experts [95, 160] and there are many other factors that falls in the developers' responsibilities, such as app performance, communication with other colleagues, and time pressure [95, 101]. By acknowledging these limits in developers' abilities and bandwidths, we need to seek better developer support that can help achieve more privacy benefits when requiring less work.

Offload Privacy Tasks

Currently, laws and platforms define many privacy tasks that need to be handled by developers, while I argue that developers do not have the ability and bandwidth to handle all of them correctly.

Fortunately, some tasks only require developers to provide a little information, and the rest of the work can be offloaded to an automated system.

The design of my tool Honeysuckle [100] was directly informed by this idea. With Honeysuckle, developers only need to specify the information that can not be accurately inferred using program analysis techniques (e.g., data use purposes, whether certain data will be transmitted off the device), and the tool will automatically generate a variety of in-app privacy notices using the information. The generated notices can partially fulfill the requirements of laws and the Google Play Store. In this way, developers need to write much less boilerplate code, making the privacy features easier to implement and maintain. In addition to saving on the burden for developers, emancipating developers from handling the design and implementation details and adopting a standardized design made by privacy experts improve the quality of the privacy features.

Need #4: Developers need to be emancipated from privacy-related tasks whenever it is possible, especially when they do not provide irreplaceable values and when standardization is needed.

Streamline Privacy Tasks

In contrast to tasks that can be offloaded, there are privacy tasks that rely on knowledge in the developer's head and can not be automated. For these tasks, we can provide a guided process to help developers handle them accurate and efficiently.

One example is the two major mobile app stores' requirements for creating privacy nutrition labels in a self-report manner. My research has shown that iOS developers could not accurately report their apps' data practices due to various problems [101, 105]. Interestingly, there are techniques or resources that can mitigate these problems, while it is difficult for developers to find and and apply them.

For example, although many widely used third-party libraries such as Google Analytics and the Facebook SDK provided documentation for this task, developers did not know these resources exist, partly because they did not use the right search terms. Furthermore, program analysis techniques can easily detect all data accesses of the app and third-party libraries used by the app. Although this is not sufficient to satisfy all the disclosure requirements, it can greatly streamline the task and prevent inaccuracies due to memory challenges as identified in my studies. Another issue is developers may not promptly update the privacy labels upon changes of data practices because they found the creation process complicated and confusing, and hence did not want to frequently go through it.

In my proposed work, I plan to design an IDE plugin that can leverage these opportunities to help developers generate accurate privacy nutrition labels and make prompt updates for them.

Need #5: Developers need assistance for privacy tasks that must be handled by them.

Reduce Task Overload

While dealing with one requirement is already challenging for developers (e.g., creating privacy nutrition labels), what they are facing in real life is a collection of requirements from different sources (e.g., different privacy laws, requirements from different platforms). This requires us to take a holistic view when estimating the cost for developers caused by privacy requirements.

Since these privacy requirements are mostly driven by the same privacy principles, they are likely to correlate and overlap with each other, such as the introduction of the same idea "privacy nutrition label" by Google and Apple. Nevertheless, because the privacy requirements were often designed independently, the subtle inconsistencies between similar requirements may lead to even more confusions and errors. For example, I noticed that Apple and Google used the same word "data collection" in their privacy label design while they selected very different meanings, which could cause trouble for cross-platform apps to comply with the privacy label requirements of both platforms [101].

Need #6: Developers need support for handling privacy requirements from different sources.

3.6.3 Leverage Agency

Although setting up concrete privacy requirements both directly prompt developers to take actions to protect privacy [98, 158] and help with enforcement, existing privacy requirements and the way they are enforced can not solve all privacy issues, especially when it comes to the complex trade-offs between privacy and usability regarding first-party data practices and the apps' main functionality. Therefore, we need to investigate how to take advantage of developers' knowledge about their app and abilities to adjust their app design. Specifically, I argue that there needs to be more research on how to encourage and support developers to learn more about privacy and reflect more on their data practices to detect privacy issues proactively.

Increase Awareness

As I analyzed the subreddit themed around Android development (r/androiddev), I found most privacy-related discussions were triggered by external events such as new platform policies regarding privacy, while privacy-related discussions rarely emerged when discussing the design and development of specific apps [98]. The low visibility of privacy-related discussions on an online developer forum is a specific case of the low visibility of privacy in development-related activities in general, which reflects developers' lack of awareness of privacy issues [95], best privacy practices [95], and design strategies to enhance privacy [160]. Overall, developers often lack the awareness about what are potential privacy issues and how to solve these issues.

Education is one direction to address this problem. We can more systematically study how developers learn privacy knowledge and acquire privacy skills and design better curricula and techniques to facilitate the learning process. Another direction is to make privacy a first-class feature in the software development ecosystem to increase its visibility. My tool Coconut explored this idea by providing developers with contextualized reminders of potential privacy issues and suggestions of solutions [95].

Need #7: Developers need to be aware of privacy issues and the corresponding solutions.

Incentivize Adoption

Developers often treat privacy as a secondary consideration [30, 95]. A fundamental challenge for designing privacy-enhancing developer support is there is little incentive for voluntary adop-

tion of tools that help improve privacy. Prior research has identified personal concerns, client or company requirements, platform feedback, and laws and regulations as drivers of privacy questions on Stack Overflow [159] and identified platform policy updates and privacy-related updates in the operating system and developer API and as triggers of privacy-related discussions on r/androiddev [98]. Therefore, one potential way to incentivize adoption is to leverage these existing drivers. Namely, we can design tools that appeal to these more urgent needs when also provide support that help with other privacy goals.

Another idea is to design support to achieve other primary goals of developers while offering beneficial side effects for privacy. One example is PrivacyStreams [103], which is a programming model that is designed to streamline the use of personal data while simultaneously improve transparency because it is also easier to analyze how the data is processed and what granularity of data is used.

Need #8: Developers need incentives for taking privacy-protective actions that are not directly required and enforced by any parties.

Chapter 4

Coconut: An Android Studio IDE plugin For Privacy

This chapter was adapted from my published paper:

Li, Tianshi, Yuvraj Agarwal, and Jason I. Hong. "Coconut: An IDE plugin for developing privacy-friendly apps." *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies 2.4 (2018): 1-35.*

4.1 Overview

Privacy has become a growing concern with today's smartphone apps, and the need for developers to protect users' privacy has become more urgent with the advent of new privacy regulations, such as the EU General Data Protection Regulation (GDPR). However, developers still face many challenges in handling privacy, which has resulted in many apps using sensitive personal data in a problematic manner [19, 57, 180]. Prior research has identified various causes of this misbehavior. For example, copying-and-pasting can be convenient, but can also introduce security bugs [61]. Unusable documentation and guidelines result in a lack of awareness and understanding of recommended security practices [16]. Poorly designed security and personal data APIs may incur extra overhead to conform to the principles of data privacy [18, 78, 79, 103]. Perhaps most importantly, developers usually treat privacy as a secondary concern [17, 28, 112], and have an incomplete understanding of what needs to be considered regarding data privacy protection [71].

Researchers and practitioners have developed a number of tools to promote secure coding [41, 114, 125, 151, 177]. However, there is currently little support to help developers in creating privacy-friendly apps. Specifically, how to design usable tools to facilitate certain aspects of privacy (e.g. privacy notices, data retention, and data sharing) has received little attention. Prior work suggests that these under-investigated aspects are indeed areas where developers lack the most awareness and understanding [71].

My overall goal is to create tools to help app developers manage the complex requirements for privacy, especially for those who work independently or in a relatively small organization that cannot afford a privacy team. I first conducted semi-structured interviews with nine Android developers to examine their understanding of privacy and how they deal with privacy issues. Several of my findings echo those from prior work, such as the tendency to de-prioritize privacy-related tasks and being unaware of the personal data automatically shared with some third-party advertising libraries [30]. I also identified additional challenges that have not yet been documented in existing literature. For example, developers may have inaccurate understanding of how their apps handle personal data, due to frequent iterations of the app and lack of documentation in collaboration scenarios.

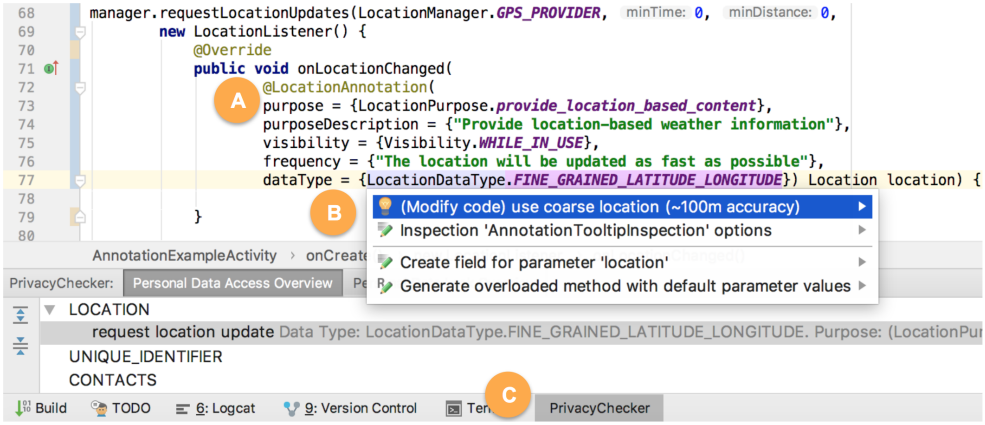


Figure 4.1: The main features of Coconut. A: LocationAnnotation is a customized Java Annotation with certain fields (e.g. "purpose") that help developers describe how and why the object 'location' obtained from the "requestLocationUpdates" API call is used. B: Coconut can give real-time feedback of potential privacy issues (highlighted in purple) and, in some cases, offer a quick fix. Here, the quick fix makes it easy to change the code to only collect coarse-grained location data. C: Coconut also uses these annotations to generate a summary of personal data practices in the app.

The findings from prior work and my interviews indicate the need for new mechanisms and tools to support privacy when developing apps. Towards this end, I designed and implemented Coconut, an IDE plugin for Android Studio, the most popular IDE for Android development. Figure 4.1 shows a screenshot of Coconut. Coconut uses heuristics to detect code that handles personal data, and asks the developer to add an annotation that describes how and why the personal data is used. Each annotation is a customized Java annotation of key-value pairs, with keys predefined for the specific type of personal data used (see Figure 1A). While programming, Coconut offers real-time feedback on potential privacy concerns by highlighting the annotation. In some cases, Coconut can offer quick fixes that make it easy to directly adopt the recommended practice (see Figure 1B). Lastly, all annotations are gathered in the "PrivacyChecker" summary panel, to facilitate review (see Figure 1C).

I conducted a between-subjects lab study with 18 Android developers to evaluate the usability and effectiveness of Coconut. The results suggest that Coconut could help developers deal with

privacy issues from multiple aspects, such as avoiding violations of privacy principles, gaining more knowledge in the apps' behavior, and providing better privacy notices to end users.

This work makes the following research contributions:

- I present the design and implementation of Coconut, an Android Studio plugin for privacy, which nudges developers to think about privacy while programming, improves developers' understanding of the app's personal data use, increases developers' knowledge of alternative options to achieve a better trade-off for privacy, and motivates developers to search and learn about privacy.
- I present the results of my lab studies of Coconut. My results show that developers using Coconut can build more privacy-preserving apps, gain a better understanding of how their app handles personal data, and write better privacy notices for users. I also observed other privacy challenges, such as developers inadvertently introducing privacy issues while tweaking their code for some functionality to work.

4.2 Design of Coconut: An IDE Plugin for Privacy

The goal of Coconut is to get developers to think about privacy and make it a natural part of their app development process. I decided to build my tool as an IDE plugin, thereby enhancing the Android Studio development environment already familiar to, and used by, all developers. I first present a hypothetical use case of my tool, and then describe the design of the plugin along with my design rationale.

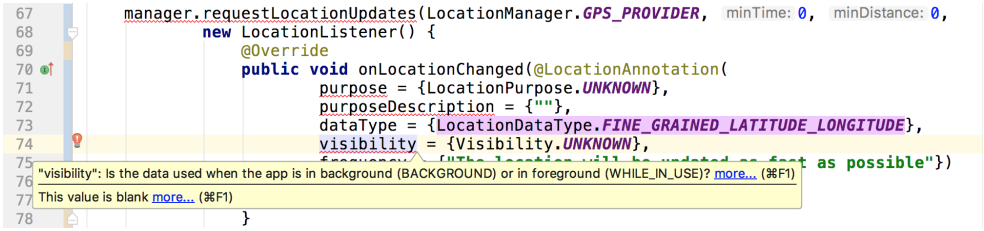
4.2.1 Coconut Use Case

Here, I present a fictional scenario to help illustrate how Coconut can help developers write privacy-preserving apps. Ann is developing a run tracking app, and a core feature of this app is to render the current route on the screen in real time, and then upload the route to the remote server database, where it is backed up.

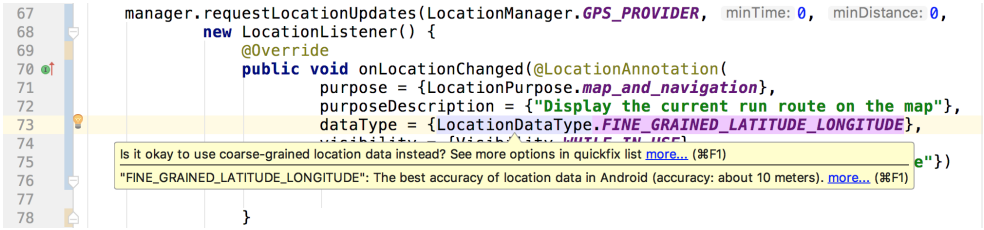
When collecting location data for rendering the route on the map, a '@LocationAnnotation' is needed to describe how and why the location data is used (see Figure 4.2). Ann decides to use the '`requestLocationUpdates`' among the series of native '`LocationManager`' APIs. After writing the API call in the Android Studio IDE, Coconut uses simple heuristics to detect this API, and requests the developer add a '`@LocationAnnotation`' annotation to the returned location object. (Figure 4.2a). Ann uses a quickfix to automatically generate an annotation skeleton with several fields (i.e. '`dataType`', '`frequency`') automatically filled in based on the parameters in the API call (i.e. '`GPS_PROVIDER`', min time and distance interval for location updates), while other fields that can not be inferred from the code need to be filled manually. Ann then inspects each of them and fills them out one by one. When she is not sure about what she is supposed to put in a particular field (e.g. '`visibility`'), she just hovers on the field name to read a further explanation in the tooltips (Figure 4.2b).

Filling out the annotation gets Ann to think through this use of location data. She first examines the collection of predefined purposes and picks '`LocationPurpose.map_and_navigation`' for her case. She then elaborates a bit more about this purpose in the field '`purposeDescription`'.

(a) When the location API is detected, the plugin requires an annotation of the corresponding data type attached to the returned location data.



(b) Coconut offers a quickfix to generate a skeleton annotation with some fields auto-filled based on code analysis. This example shows the result after applying the quickfix. An explanation of the field is displayed when hovering on the field name.

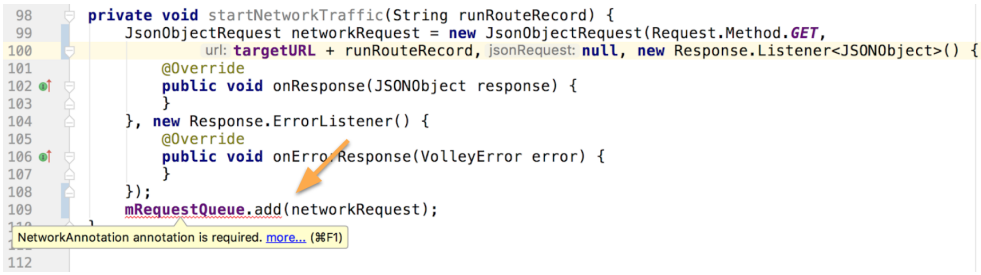


(c) Potential privacy issues are defined as a warning and highlighted in purple. Where possible, quickfixes are provided so that developers can apply another option with one click if they think it is a better fit for their purposes.

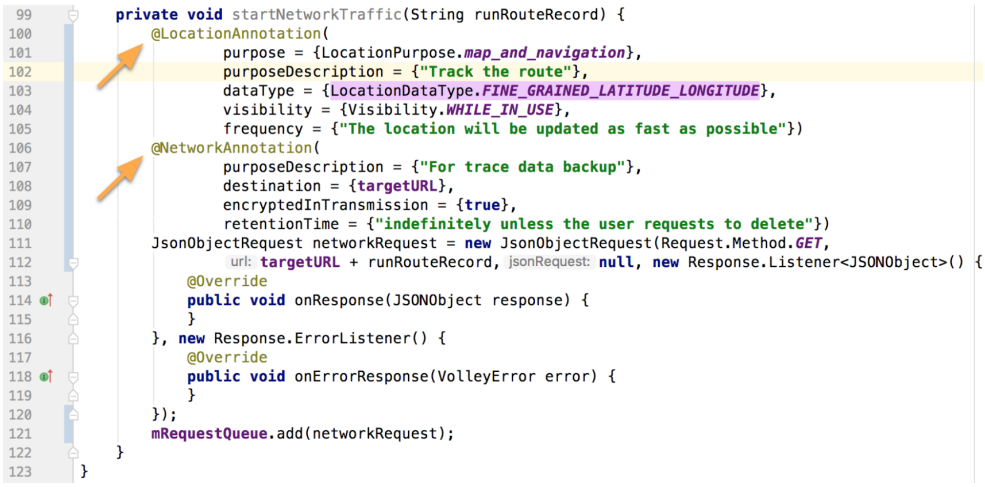
Figure 4.2: Building a run tracking and mapping app with Coconut: When collecting location data for rendering the route on the map, a '@LocationAnnotation' annotation is needed to describe how and why the location data is used.

During this process, she is reminded to check that she has a legitimate purpose to collect location data. When it comes to '`visibility`', she considers for a while and decides that the data is only needed when the user is using the app, so she opts for '`Visibility.WHILE_IN_USE`'. After resolving all "errors" marked with a squiggly red underline, she attends to the warning at the value '`LocationDataType.FINE_GRAINED_LATITUDE_LONGITUDE`' highlighted in purple (Figure 4.2c). Before reading this tooltip, she has a rough idea that she can collect fine-grained location because the '`ACCESS_FINE_LOCATION`' permission is requested, but she does not know the exact accuracy of "fine-grained location", or what other accuracy is available in Android. After exploring the quickfixes for this warning, she learns more about the alternatives and finally decides to stick with the original choice because it best fits her purpose.

When sending the routes to the remote server, a '@NetworkAnnotation' annotation is needed to describe how the data is transmitted from the phone, along with other annotations (the '@LocationAnnotation') that describe what personal data may leave the phone and for what purposes (see Figure 4.3). Ann starts to construct a network request to send the routes aggregated from the location data to the remote server. Similar to the location API in the previous example, Coconut



(a) When network traffic is detected, the plugin requires a 'NetworkAnnotation' attached to the target network request data.



(b) After completing all required annotations. NetworkAnnotation describes how the data is transmitted, other personal data annotation describes what data will leave the phone at this point.

Figure 4.3: Building a run tracking and mapping app with Coconut: When sending the routes to the remote server, a '@NetworkAnnotation' annotation is needed to describe how the data is transmitted out of the phone, along with other annotations (e.g. the '@LocationAnnotation') that describe what data may leave the phone at this point and what the purposes are.

detects the network connection, and requires developers to add a '`@NetworkAnnotation`' that describes how data is transmitted out of the phone (Figure 4.3a). In addition, Coconut also requires Ann to annotate what data may leave the phone at this point. Since this request contains location gathered previously, Ann just reuses the prior annotation. The final result is presented in Figure 4.3b.

When writing the privacy policy for this app, Ann reviews personal data practices using the PrivacyChecker window (see Figure 4.4). One month later, Ann finishes the app and plans to release it on the app store. The app store requires her to provide a privacy policy for her app. She starts looking at other apps' privacy policies and tries to adapt them to her app's practices. During this process, she uses the PrivacyChecker tool window, which is a feature of Coconut that gathers all data practices from the annotations. By referring to the annotations, Ann can precisely describe how personal data is used for the current version of the app, including how the location data is collected for tracking running routes and how data is sent over the network for

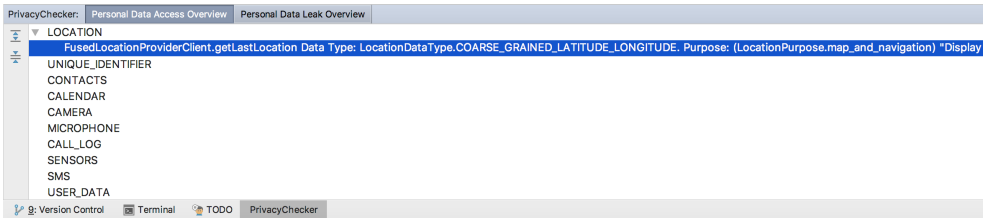


Figure 4.4: The developer can examine the global personal data practices using the Privacy-Checker window. Personal data practices are categorized by data type and whether it leaves the internal runtime environment. All cells that display data use instances are clickable and can help the developer navigate to the corresponding code snippet when they click on it.

backup purposes.

4.2.2 Designing Coconut to Promote Privacy-Preserving Personal Data Use

Above, I described how an Android developer can use Coconut across the entire development process to gain a better understanding of app behavior, learn about potential privacy issues and better choices, and manage data practices. Next, I detail the design of the main features and explain the underlying design rationale.

Coconut Requires the Developer to Annotate Personal Data Practices in a Pre-Defined Format.

Privacy annotation is a key feature of Coconut (e.g. the '`@LocationAnnotation`' and '`@NetworkAnnotation`' in Figure 4.3b). To implement this concept, I use Java Annotations[1], a form of syntactic metadata in Java. Coconut has two major kinds of annotations, namely 'source annotations' like '`@LocationAnnotation`', and 'sink annotations' like '`@NetworkAnnotation`'. Using these two types of annotation, developers can track how personal data flows within an app. More specifically, the source annotation specifies where the personal data is acquired, and the sink annotation specifies where it leaves the app. Since developers are likely to have an incomplete understanding of privacy as discussed in Section **??**, I draw on previous research in data privacy modeling [82, 86] to design the fields for each type of annotation, so that they cover important aspects of data privacy that are supposed to be described in the privacy policy. Coconut runs code inspection continuously in the background. If a relevant API call is detected and the corresponding annotation is missing, it is treated as a "missing-annotation error", and the API name will be marked with the same indicator of compile error in Android Studio (i.e. a red squiggly underline) to inform developers that this is a required task.

```
141  @UniqueIdentifierAnnotation(
142          purpose = {UIDPurpose.tracking_user_data_collected_from_multiple_apps_on_this_device},
143          purposeDescription = {"Google may use the advertising ID from the device on which the
144          uidType = {UIDType.ADVERTISING_ID},
145          scope = {UIDScope.PER_DEVICE},
146          resettability = {UIDResettability.USER_RESETTABLE_IN_SYSTEM_SETTINGS})
147  AdRequest adRequest = new AdRequest.Builder().build();
148  mAdView.loadAd(adRequest);
```

Figure 4.5: The entire @UniqueIdentifierAnnotation is automatically generated because Coconut is pre-programmed with the implicit data collection behavior of Google AdMob library.

Coconut Helps Generate Annotations and Checks the Consistency Between Annotation and Code Using Code Analysis.

To reduce cognitive load when annotating personal data practices, and to mitigate errors in annotation due to carelessness or misconception of how the API works, Coconut analyzes the code and infers the value of certain fields of an annotation or even the entire annotation. For example, the '`datatype`' field in Figure 4.2b and the entire '`@UniqueIdentifier`' annotation in Figure 4.5 are automatically generated. In addition, Coconut also automatically locates the object that contains the personal data which the annotation should be attached to.

Additionally, if there is a discrepancy between the value of these fields in the annotation and the behavior of the code, Coconut will report an "annotation-code inconsistency warning" and highlight the corresponding field name in red. I use a different color than normal warnings in Android Studio and other types of warnings in Coconut, so that developers can establish a mapping in their mental model between the color and the type of issue. This feature helps developers stick to the most updated and accurate understanding of their apps' behavior.

Coconut Provides Real-Time Feedback on Potential Privacy Issues.

Coconut offers a "privacy-concern warning" when any potential violation of privacy principles is detected (e.g. network traffic not encrypted; using hardware identifier when not necessary; collecting user data for purposes that may not be clear to users), or when the developer may not know that there exists less privacy-invasive approaches to achieve similar goals (e.g. changing parameters to get coarse-grained data when high accuracy is not needed; using APIs designed for privacy [103]). The corresponding field value is highlighted in purple to notify the developer.

To address the issue of developers not being able to balance the trade-off between privacy and other factors, Coconut provides developers with alternative recommendations in the tooltips. If possible, I also provide quickfixes that can directly apply the recommended change in the code. As a result, developers' awareness of alternatives increases, so that they can make more informed design decisions. The quickfix feature may also lower the threshold for adopting recommended practices.

Coconut Gathers Personal Data Practices in One Place to Facilitate Review.

Coconut offers *PrivacyChecker*, a tool window that gathers all the personal data practices described by the annotations. It displays the data used internally and the data that may leave the

[1]https://docs.oracle.com/javase/tutorial/java/annotations/

the internal runtime environment in separate panels, and categorizes them by personal data types (See Figure 4.4).

The content in the panel is constructed dynamically, so the developer will always have a clear understanding in the personal data practices of a certain version. Developers can also review what privacy issues still exist based on unresolved Coconut "privacy-concern warnings"

4.2.3 Implementation

I built a proof-of-concept prototype of Coconut, which consists of two parts: *an Android Studio plugin* and a *privacy annotation support library*. The plugin was developed using the IntelliJ Platform SDK[2]. I use this SDK to monitor API usage by capturing file changes in the IDE in real-time, to inform developers of privacy errors and warnings by registering errors/warnings using code inspection APIs, to generate pre-filled annotation skeleton and fix privacy issues by manipulating the code via the syntax tree, and to present the personal data use in the app in a tool window. The version for my lab study comprises 7770 lines of Java code, most of which is used for establishing the basic framework and handling the user interface.

The current implementation supports analyzing a selected group of system APIs and third-party libraries that use personal data. The scope was determined with the goal of providing sufficient flexibility in the programming tasks of my lab study (described in Section 4.3.2). Given the current framework, this list can be easily extended. Recent research has demonstrated the long-tail distribution in the usage of third-party libraries [42], which suggests that it is feasible to cover a wide range of personal data collection behaviors caused by third party libraries. There are always-running threads that monitor the use of these APIs. When a target API is detected, the system will infer the target location of annotations for this API, check if all required annotations are provided, valid, and consistent with the actual code behavior, and also check for other potential privacy issues. The privacy errors and warnings and the automatic quickfixes provided by Coconut are presented in Table 4.1.

Privacy annotations are customized Java annotations, which need to be predefined in Coconut's privacy annotation support library. The *source* and *sink* annotation annotate the code when the data is acquired and when it leaves the app. The *third-party lib* annotation is used to specify some configurations not defined in the code. For example, AdMob offers developers a web-based management system to specify whether the location-based advertising service is enabled. And I request the developer to also specify the same thing in the annotation so that Coconut can correctly infer the code behavior.

4.2.4 Pilot Study and Feedback

During the development process of Coconut, I solicited early feedback from six developers to improve usability. I accepted suggestions that I felt were feasible for the first prototype of Coconut. For example, I observed that developers tended to quickly test how to use new APIs. However, an error that prevents compilation (e.g. a missing or incomplete annotation) made it harder to just try out an API. Consequently, I updated Coconut to let developers temporarily ignore annotation

[2]https://www.jetbrains.org/intellij/sdk/docs/welcome.html

compile errors. However, missing annotations still appear as errors, and need to be addressed before building the final app, because I still treat these annotations as a requirement.

4.3 Evaluation Methodology

4.3.1 Participants

To evaluate the usability and effectiveness of Coconut, and to gain a better understanding of how developers handle users' personal data with and without my tool, I conducted a series of in-lab studies with 18 participants from March to May 2018. My participants were recruited from two sources. The first source was developers on LinkedIn. I used "Android Developer" as a keyword and set the location filter to my local area to make sure developers can come to my lab to participate in the study. I found 30 qualified candidates in total and then contacted them using LinkedIn InMail. The second source was computer science students at my university. I posted an advertisement in a related Facebook group and email lists, and also put up physical posters for this study in my school. The study was approved by my university's IRB.

Among the 18 participants (14 males, 4 females), 8 self-identified as professional Android developers. All participants were over 18 years old and were familiar with using Android Studio. I used a between-subjects design, with a control group (not using the plugin) and an experimental group (using the plugin). I controlled the number of professional Android developers to be the same in each group. Both groups had an average of 2.5 years of Android development experience. Participants in the experimental group published an average of 2.3 apps on Google Play Store, and the control group 2.2. See Table 4.2 for more information.

4.3.2 Study Task Design

Study participants were asked to complete two programming tasks: a warm-up task and a main task. The warm-up task helped participants get familiar with the programming environment, and also served as a reference of their expertise in developing Android apps that handle personal data. In the warm-up task, developers needed to handle a run-time permission request for location, collect the actual location data, and display the current latitude and longitude on the main UI.

When designing the main task, I focused on three key aspects: the feasibility of finishing it within a limited amount of time, the scope of the privacy principles it can cover, and the ecological validity of the use cases. The high-level goal of the main task was to build a weather app. I drew on a widely-used feature in popular weather apps such as The Weather Channel[3] and AccuWeather[4], which is to provide location-based weather information. Multiple privacy principles should be considered here. First, the granularity of location should be subject to the purpose of getting weather information, which does not need to be very precise (purpose limitation). Second, the app needs to send the location data to a third-party weather provider service to get the weather information, which entails data sharing practices for the core functionality of the

[3]https://play.google.com/store/apps/details?id=com.weather.Weather
[4]https://play.google.com/store/apps/details?id=com.accuweather.android

app and should be clearly conveyed to the users in advance, to conform to the privacy principles about providing privacy notices, especially for data practices that may be implicit to end users.

In addition to the core functionality, I also included some requirements for monetization and analytics. These are common purposes for using personal data, but fewer end users are aware of them, which may lead to privacy concerns. For app monetization, I asked developers to integrate a banner ad using Google's AdMob library, which is the most popular advertising library on the market. Prior research has demonstrated that using these libraries can cause severe privacy concerns because they can collect users' location data in the background when the location permission is granted, which most users and developers are not aware of [30]. For analytics, I asked developers to store the location data locally and assume it would be used for analyzing users' location history. Since location history can disclose sensitive information about a person, it should be stored securely. I also required developers to collect a unique identifier for the user and to store it with the location data. The selection of a unique identifier affects whether the personal data is properly anonymized, and is also subject to the purpose limitation principle.

For both the warm-up and main task, developers were free to choose any system APIs and parameters to use for collecting location data, generating/acquiring the unique identifier, and storing the data. I provided skeleton code that handled things not related to the focus of this study, such as updating the UI and initiating the network request to the weather web API[5], to save time and let participants focus on privacy.

4.3.3 Study Procedure

The study lasted for 1.5 to 2 hours. All participants came to my lab. I provided them with a laptop (Mac OS or Windows based on their preferences) and an Android phone for development and testing. Both Android Studio IDE and my Coconut plugin were installed. This study follows a between-subjects design. Both groups did the warm-up task without the plugin enabled, and only the experimental group had the plugin enabled for the plugin training process and the main task. Participants were compensated with a $75 Amazon gift card.

After introducing the study goals and logistics, both groups started with the warm-up task, which required them to obtain the current latitude-longitude location data and display it on the screen. Both groups had 40 minutes for the warm-up task. The warm-up task was designed to familiarize the participants with the development environment. It is also designed to compare the performance of the two groups when both in the control condition.

Next, the experimenter walked participants in the experimental group through a tutorial of my Coconut plugin, which used the same fictional use case in Section 4.2.1. The experimenter explained the semantics of the source and sink annotations, explained the meaning of each field of the annotations involved in this example, showed how to expand the field definition tooltip by hovering on the field name, and introduced the three types of feedback from Coconut: "missing-annotation error", "annotation-code inconsistency warning", and "privacy-concern warning". The tutorial was always available to the user during the study. This training process was to familiarize them with programming with Coconut, especially the concept of annotations.

Then both groups were asked to complete the main task of building a weather app. Only

[5] api.openweathermap.org

the participants in the experimental group had the plugin enabled. Both groups had one hour to work on the main task. I explicitly asked developers from both groups to imagine they were developing an app that would be used by real users and to take privacy into consideration during the development process.

After finishing all programming tasks, I asked participants to fill out an exit survey that had three sections. The first section asked them to write a privacy policy for the weather app they just built. When writing the privacy policy, both groups could refer to the code they just wrote. The experimental group could also use the PrivacyChecker to review their data practices, as documented in their annotations. The second section had some factual questions about the personal data practices of their app. The experimental group had a third section which solicited feedback about the Coconut plugin.

Finally, the experimenter briefly interviewed the participants, asking about what considerations they had for privacy when developing the app, what they thought about the plugin (only for the experimental group), and the rationale behind some of their behaviors, such as ignoring web resources that contained more privacy-preserving options, ignoring particular privacy errors/warnings or not addressing them properly, and not correctly answering the factual questions in the exit survey. Notes were taken during the process.

The screens of the laptop and the Android phone were videotaped for later analysis, and no audio was recorded. Screen recordings were played back to developers during the exit interviews to help prompt their memories. During the study, developers could use any web resource. However, I required them to only use native APIs or Google Play Service APIs when obtaining personal data due to the scope of APIs currently supported by Coconut.

In the training process, I would answer any question they had regarding the plugin. During the programming tasks, I only answered clarification questions on task requirements, what they could do (such as which part of the code they could change), and what resources they could use. If the participant did not successfully finish the warm-up task within the specified time, I would offer hints on what they were doing wrong (for both groups) afterwards. In this way, I could observe more use of Coconut in the experimental group during the main task when ensuring both groups received the same amount of help.

During each session, one experimenter observed the participant and took notes. These notes include the actions the developer took, the search queries they used, and the thoughts they expressed verbally. After the study, the experimenter reviewed screen recordings to count the time spent on each task and subtask and document some coding and information foraging behaviors.

4.3.4 Privacy Policy Evaluation Methodology

To avoid potential bias, I invited two external judges for evaluating the privacy policies created by participants, as collected from the first section of the survey. One was a Ph.D. student studying usable privacy; the other was a master's student from a privacy engineering program and also had research experience in usable privacy.

I invited the judges to come to my lab for this evaluation session. During the session, all information was presented to the judge in one spreadsheet. Data from the two conditions were aggregated and stripped of any group information. I first introduced to them the expected personal data practices if the entire task was completed, and presented all valid responses of privacy

policies with a description of the actual behavior of each app, since not all developers finished all task requirements. I created two different spreadsheets for these judges. The responses and app behavior description in these two spreadsheets were the same, with the order randomly shuffled to minimize any ordering bias.

I asked the judges to first skim through all of the responses and then came up with a set of criteria for judging these blurbs of privacy policies. I relied on their expertise and did not give detailed instructions on how to judge the quality of privacy policies. Judge 1 referred to the Fair Information Practices[6] as the main criteria, and also added two more requirements "providing true information" and "avoiding jargon" for evaluating the truthfulness and readability of privacy policies. Judge 2 did not refer to any specific material. He considered requirements including "providing accurate description", "providing useful information", "using plain language", "explaining all data being collected in the app", "specifying the purpose of using personal data", "providing users with ways to control data use", "informing users of data sharing practices". Overall, their standards for good privacy policies almost overlapped. I asked the judges to assign the rating for each response from 1 to 10, with 1 being the worst quality and 10 the best, based on the rubrics they developed. They were also asked to think aloud and explain the reason for each rating. The experimenter took notes in the meanwhile. The two judges conducted the evaluation separately and independently.

4.3.5 Research Questions

Nudging developers to adopt better privacy practices while programming can be challenging. For example, some developers may habitually ignore and fail to address warnings, and it is unclear whether developers can understand the underlying privacy concerns when filling the annotations. As such, I aimed to examine the overall effectiveness of Coconut using the following research questions:

- RQ1: Can Coconut help developers avoid more privacy violations?
- RQ2: Can Coconut help developers gain a better understanding of their app's personal data practices (RQ2.1) and write better privacy policies (RQ2.2)?
- RQ3: Do developers consider Coconut as useful and usable?

Previous studies on how developers think about and handle privacy mostly used retrospective inquiries. In contrast, my lab study allowed us to closely observe how developers deal with privacy issues while programming. Thus, I added a fourth research question:

- RQ4: What kind of challenges for privacy do developers face while programming and when they have been primed with the notion of privacy in advance of the main task?

4.4 Results

4.4.1 Results of the Warm-up and Main Task Completion Are Similar Between Two Conditions

I first present the results about task completion and the time spent on each task. Table 4.3 presents an overview of how participants in the control and the experimental group performed in the warm-up task and the main task.

For those who completed the warm-up task, the average time for the control and the experimental group was 27 and 24 minutes respectively. For those who completed the main task, the average time for the control and the experimental group was 41 and 47 minutes respectively. The average time spent on annotations (including completing the annotation, reading the tooltips, attending to and resolving issues specified in the annotation) was 10 minutes. The number of people who completed the main task was the same, while developers in the experimental group spent slightly more time than those in the control group, possibly because of the extra cost of annotation and the time spent on thinking about and dealing with privacy considerations. I will further detail how developers perceive this cost in the following section as well as in the discussion section at the end.

The success rate of the warm-up task was similar across the two groups, which suggest the app development expertise was well balanced. I did not expect the success rate of the warm-up task would be low, since I pre-tested these tasks with a convenience sample of graduate students who had experience with Android programming and mobile privacy. My results suggest that obtaining location data, which includes requesting location permission for the app, selecting the location API to use, and debugging, can be very challenging for average developers. Note that the main task had a higher completion rate, because parts of the main task overlapped with the warm-up task.

4.4.2 Coconut Can Help Developers Write More Privacy-Preserving Code

Overall, the comparison between the two groups shows an improvement for privacy when using Coconut. More developers using Coconut only collected coarse-grained location data for weather information, only shared coarse-grained location data with the AdMob library, used private storage for sensitive personal data, and selected the proper unique identifiers (Table 4.3).

I also examine why some developers using Coconut stayed with less optimal solutions, such as getting fine-grained location data when the purpose did not require this level of accuracy. I observed that some developers tended to ignore warnings. It could be that my choice of using different colors from normal warnings in Android Studio led to developers not recognizing the warning. Participant E4 also mentioned that he did not always trust the auto-generated annotation, and he would like to first manually verify them several times to develop trust. This issue of trust was discussed further as one of the limitations of my methodology in Section 4.5.4.

[6]https://iapp.org/resources/article/fair-information-practices/

4.4.3 Coconut Can Help Developers Better Understand an App's Behavior

Table 4.4 presents the responses to the second section of my post-study survey, which tested developers' comprehension of their apps' behaviors. Some questions were not applicable for developers who did not finish all required tasks, so some of the total counts (the denominator in the fraction) was less than nine (size of each group).

For the overall comparison between these two groups, I only calculated descriptive statistics because of the missing values due to the uncompleted tasks. I accumulated the number of all valid answers and correct answers to calculate a overall correct answer rate. The overall rate of correct answers for the experimental group was 88.14%, higher than the control group (66.67%). The habit to ignore warnings and the insufficient trust in the auto-generated content may explain some incorrect answers of developers using Coconut.

I expected developers who finished the weather feature to answer that the data was transmitted to the server that hosts the weather API, and developers who integrated the banner ad to answer that the data was transmitted to Google's servers. However, most developers in the control group failed to identify these two facts, especially the one for the ad library. The follow-up interviews showed that the auto-generated annotations for this third-party library helped Coconut users learned about this data sharing practice. The interview results also suggested that although developers from both groups would know that the location was sent to a weather website if asked directly about it, it was harder to recall it from memory without the prompt of annotations.

4.4.4 Coconut Can Help Developers Write Better Privacy Policies

I collected 18 privacy policies written for the weather app, and there was one invalid answer from E2, which just contains one vague phrase "highly recommended". This may be because that person did not know what privacy policy means. In the following result analysis, I discarded this outlier. I calculated an average of the scores from the two judges as the final store for each privacy policy.

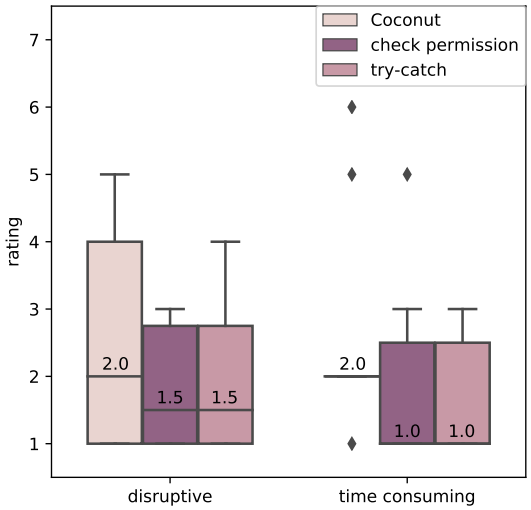
The median of the aggregated final scores for the privacy policies of the experimental group and the control group were 5.875 and 2.750 respectively, and the former was significantly higher (Mann-Whitney U Test[7], $U = 6.0$, $p < .005$). This suggests that using Coconut can help developers write better privacy policies.

Overall, the experimental group's privacy policies covered more points in the two external judges' criteria, including "purpose specification", "statement truthfulness", "giving opt-out options", "discussing security protections", "openness of implicit data practices (such as data storing and data egress)", and "readibility". Several aspects exhibited the most salient improvement, including "purpose specification", "statement truthfulness", and "openness of implicit data practices".

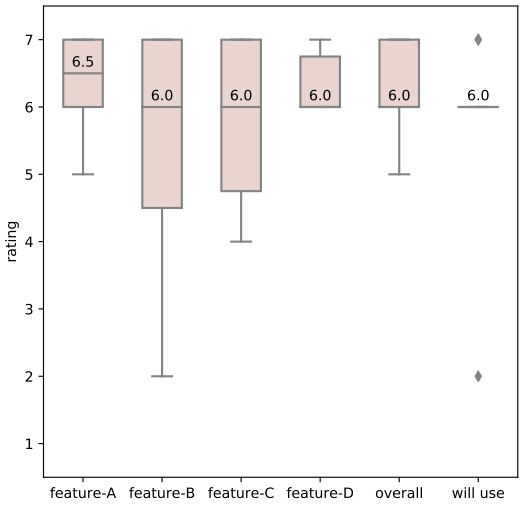
4.4.5 Coconut Is Perceived as Useful and Usable

Since the requirement of adding annotations can incur more cost to the developers, I specifically asked them to evaluate if they perceived annotations as disruptive and time-consuming. I also

[7]I use Mann-Whitney U Test because the ratings do not follow a normal distribution.



(a) The results of how disruptive and time-consuming developers perceived about the new requirement of adding annotations (Coconut), and some existing requirements in Android Studio, including adding code to check whether corresponding permissions are granted before invoking personal data API (check permission) and adding try-catch blocks (try-catch) on a 7-point likert scale (1 for not disruptive/time-consuming at all, 7 for very disruptive and time-consuming).



(b) Perceived usefulness of Coconut and key features, as well as how much developers are willing to use Coconut, on a 7-point likert scale (1=not useful/willing to use at all, 7=very useful/wiling to use). feature-A: generate pre-filled annotation skeleton; feature-B: navigate between the API call and annotation for the API call; feature-C: change annotation to the speculated value (when an inconsistency is detected); feature-D: change code to privacy-preserving options.

Figure 4.6: Results of the plugin evaluation part of the survey: most developers found Coconut and features of Coconut very useful, and considered the cost of adding annotations as moderate and comparable to existing requirements. The median values of each group of results are marked in the box.

established a baseline by asking them to rate other kinds of coding tasks, such as adding a permission check before using a protected API and adding try-catch blocks to handle exceptions. The results (presented in Figure 4.6a) suggest that developers perceived a moderate cost for adding annotations, which was comparable to the cost of these existing requirements. Besides, they may gradually feel more comfortable with it after using it as part of their regular development process for a while. When asked why they did not consider the annotation to be very disruptive, some participants ascribed that to the quickfix that generates annotation skeletons and the flexibility to defer filling the annotation.

My survey results also suggest that developers considered many features of Coconut to be very useful, and had great interest in using it for their future projects (See Figure 4.6b). Participants explicitly mentioned how they benefited from Coconut in the interviews, including thinking more about privacy and recognizing privacy issues that they were previously unaware of because of the annotations (E1, E4, E5, E6, E7), gaining more knowledge of their app's behavior because of the auto-filled values in an annotation (E2, E3, E6, E7, E8), getting to know more options and making better design decisions due to the real-time feedback (E5, E6, E9), and streamlining the process of reviewing the personal data practices using the PrivacyChecker overview panel (E1, E4, E6, E9).

4.4.6 Challenges to Handling Personal Data Properly Observed While Programming.

My lab study also provided us with an opportunity to observe how developers deal with privacy concerns while programming. I present some new challenges and how Coconut helped address them.

Challenges for Privacy in the Information Foraging Process

Similar to prior work that studied the implications of information sources on code security [16], I also observed challenges for privacy involved in the information foraging process. Developers used a wide range of information resources during the development process, including official tutorials, API documentation, Q&A sites such as Stack Overflow, tutorials from other websites, and code examples on GitHub. Many of the official tutorials were designed to take privacy into consideration. For example, the code example in "Get the last known location"[8] only requests the '`ACCESS_COARSE_LOCATION`' permission, so developers might not collect location data too fine-grained for their purposes if they just followed the default option. Similarly, the "Best practices for unique identifier" tutorial[9] gives an extensive overview of principles and recommendations for specific use cases to choose the proper unique identifier that protects users' privacy.

However, according to the observed developers' behavior and the exit interview results, developers did not make proper use of these resources to improve privacy, for four reasons.

First, some developers already had some experience with the APIs and a rough implementation plan. As such, they either searched very specific keywords that did not elicit these resources

[8] https://developer.android.com/training/location/retrieve-current
[9] https://developer.android.com/training/articles/user-data-ids

in the search result, or directly skipped them because they wanted to just see a code snippet that could refresh their memory of how to use a certain API (C2, C4, E8).

Second, some developers found or had the expectation that there would be too much information in such documentation, and instead favored Q&A sites that had more concrete examples to show *"how this API works in context, more than just API doc that explains what's the functionality of each API in detail"* (C7).

Third, some official tutorials did not cover all necessary steps, which caused trouble for novice developers. Although 14 participants opened the "Get the last known location" tutorial which recommended developers to use Google Play services location APIs, only 3 of them successfully implemented the required feature with these APIs. Most developers got stuck when setting up the Google Play Services in their project because there were no direct code examples in that tutorial to refer to. As a result, many developers sought other resources for help, and finally switched to using '`LocationManager`' APIs, which were easier to implement, but had fewer granularity options, and required the developers to manage the low-level location provider specification manually.

Fourth, developers did not always not fully grasp the recommendations and the rationale behind them. E3 admitted that although he saw the principle "Avoid using hardware identifiers" in the "Best practices for unique identifiers" guidelines, he did not pay much attention to it because he was lazy. Furthermore, I frequently observed during the coding process (and confirmed in the interview) that developers tended to directly jump to code examples or keywords related to code. For example, the same "Best practices for unique identifiers" document directly caused C6 to search how to acquire Android ID when it actually opposed using Android ID for his case, because he immediately noticed the keyword SSAID (Android ID) without paying attention to the context where the keyword appeared. The results further motivated the need for a tool that could actively check with the developer about their understanding of the behavior of the code and remind them of better options.

The Common Strategy of "Getting the App to Work First" May Get in the Way of Privacy.

When interviewed about the rationale behind the decision-making process, some developers referred to a common theme: "doing things just for getting the app to work first" (C7, E2, E4). Additionally, I frequently saw study participants do fast testing of APIs, which can act in conflict with privacy requirements.

The location collection task was the most complicated one for most developers because it involved a run-time permission request and because there were multiple APIs (e.g. location APIs from '`LocationManager`', '`FusedLocationProviderApi`', and '`FusedLocationProviderClie`
and parameters to choose from. Furthermore, sometimes the sensor data may not be available or might update too slowly. When the app did not work as expected, developers with little experience in these APIs tended to try out each solution they found online or simply tweaked some parameters in the API call until it worked. Meanwhile, some decisions that may be less privacy friendly (such as using fine-grained location) were made because the app happened to work after changing the parameter that determined the granularity, although it was not the right solution (C7).

My plugin can be helpful in mitigating this issue. For example, E8 in the experimental group

also faced the problem of not getting the location update fast enough. The right solution is to change the parameter that controls the minimum update time interval. At first, she changed the parameter that controls the granularity of the location data, and happened to get the app to work temporarily. However, after this change, the parameter '`LocationDataType.FINE_GRAINED_LOCATION` was highlighted in purple, which indicated a potential privacy concern. Then she kept investigating this problem and finally found the correct solution. This example shows that my annotations and real-time code inspection can help developers stay aware of any new privacy concerns in their code. Even if they decide to not deal with the concerns immediately, the warning can work as a reminder so they will not forget about it and handle it in the future.

4.5 Discussion

4.5.1 The Value of Privacy Annotations

My study results suggest that, although asking developers to do annotations for privacy involves some extra burden, developers do not perceive them as very disruptive or time-consuming, while also appreciating the value it provides. Here I discuss four reasons why these annotations improved privacy.

First, the process of completing the annotation may lead developers to think through the implications of their code on privacy. Although the task switching between normal development work and filling annotations could incur more cost in time and cognitive demands, many developers appreciated it because they think that they would be more willing to make modifications for privacy at the early stage of app development. This also resonates with P5's example in the problem finding interviews, which demonstrated that considering a security/privacy issue at late stages may cause extra cost that could have been avoided. Besides, the predefined fields of annotations could remind them to think about a wide range of privacy concerns. For example, some developers (E5, E6, E7) mentioned in the interview that the annotation helped them select the appropriate granularity of location based on the purpose. Although they may not have heard about the "purpose limitation" principle, they could still unconsciously apply that in their decision-making process by drawing connections between the purpose and the location granularity specified in the annotation.

Second, the privacy annotation may help developers be more aware of how the app uses personal data. This is not only because developers could reflect on their data practices when filling the fields of the annotation and correct misconceptions when seeing the real-time feedback highlighted on the annotation, but also because the annotation was kept as part of the code and summarized in one place. Many of my participants considered the overview panel to be potentially useful for large-scale project development and maintenance.

Third, developers may learn more options via the annotation and make better trade-off between privacy and other factors such as usability. More privacy-preserving alternatives can be presented in the real-time feedback, quickfixes, and the pre-defined values for some fields (such as the `LocationDataType` that provides several different granularities of location data). As a result, developers will make more informed design decisions and make a better trade-off between privacy and other factors.

Lastly, doing annotation may motivate self-taught developers to learn more about privacy. Many Android developers did not receive formal privacy training, and the situation becomes worse when they do not use tutorials that promote best privacy practices or get a wrong idea from it (discussed in Section 4.4.6). Annotation can be helpful because developers may use the keywords from the tooltips or the auto-generated content to search for related guidance. I observed developers searched the new keywords after attending to the annotation and also revisited a material that was opened but not paid enough attention to before.

I want to note that the current design and implementation may be more useful to developers who worked in small teams and have limited knowledge and bandwidth to deal with privacy requirements. The nudge to better privacy practices may not be as effective for developers who were subject to the decisions of other people in a large team, and I would like to probe into this problem in the corporate and collaborative context in the future.

4.5.2 Design Recommendations for Addressing Annotation Maintenance Issues

Because annotations do not directly affect functional code, it is possible for developers to specify annotations that were not consistent with the actual behavior of the code. In my current implementation, I tried to tackle this problem by inferring some field values using heuristics, continuously checking whether the value specified in the annotation was consistent with the inferred value and providing real-time feedback to developers in the IDE. My study results suggest that this strategy is effective at assisting developers to maintain valid annotations.

However, I identified some other barriers to maintaining accurate annotations. The first was a lack of trust in automatically filled values in the annotation. Besides letting users gradually develop trust over time, I can also consider adopting some approaches studied in other intelligent systems, such as to provide explanations of how the value was speculated, or even just to provide the origin of the speculation [65]. The second barrier was that developers may understand certain terms differently. For example, developers expressed different interpretations of the '`WHILE_IN_USE`' option for the '`visibility`' field. Some people considered that if an activity was paused but not destroyed, it should still be in use, while other people thought an activity was only in use if the user interface is active. Both stronger automatic analysis techniques that check and enforce a unified semantic and advanced modeling of developers' perception of terms that describe the app status could be helpful.

4.5.3 Generalizability of Using Annotations to Enhance Privacy in Other Programming Languages

In this work, I explore and demonstrate the potential of using customized annotation to help developers handle privacy specifically for Android apps written in Java. I also expect this idea to be applicable to other languages, because annotation/attribute is also used in other mainstream programming languages. For example, C# supports adding customizable attributes to various types of programming elements, which was similar to Java's annotations. As a result, privacy annotations can also be used in C# code by implementing support libraries.

4.5.4 Limitation of the Evaluation Methodology

Acquiring a large sample for lab studies that involve observing programming process has long been recognized as difficult. First, it is often hard to recruit software developers who usually have a much higher income than the study compensation [17]. Second, the length of such studies can be several hours due to the programming task, which can be hard to scale up. In my study, I recruited 18 developers for a between-subject study, which was similar to many prior studies with similar designs [35, 90]. However, the small sample size may still have implications on the confidence to draw conclusions from the study results. Besides, although I have recruited developers with diverse development backgrounds from multiple sources, half of my participants were still college student developers, which may not be representative enough of the entire population of Android developers.

Although I have tried to design tasks that contain realistic use scenarios and explicitly asked both groups to treat it as an app with real users, the lab study method still has its inherent limitations. Because of the constraint on the time to finish the tasks, some of them may start building the app with less careful plans and exhibit more fast prototyping behaviors than usual. Besides, developers in both groups may pay more attention to privacy than usual, because they were primed by the notion of privacy before working on the main task. Regarding the use of Coconut, developers who used this plugin for the study may react more actively and positively towards the plugin's suggestions and warnings. On the other hand, their performance may also be negatively impacted by the unfamiliarity and lack of trust of the tool because of the short-term exposure in the lab session.

Some potential benefits of Coconut need to be examined with a larger and less controlled study. For example, when developers work on a large-scale project, they may find the annotations and the summary panel more useful for maintaining a correct understanding of the apps' behavior; when they are not subject to the study task requirements, the process of doing annotations and the feedback conveyed through the annotations may trigger more radical changes in the design of the app to improve privacy.

4.5.5 Future Work

In the future, I would like to explore using annotations to improve privacy in two directions. The first is to enhance the current tool. For example, I have considered integrating taint analysis or other more sophisticated static analysis techniques to further reduce the burden on developers. The second is to use annotations to benefit end users and auditors. Specifically, I would like to help developers create privacy policies with annotations and design a better privacy notice interface that can be automatically generated based on these annotations.

Table 4.1: Privacy issues that can be detected in the current proof-of-concept prototype for the lab study. These issues are detected by combining the code analysis and the annotations completed by the developer. Automatic quickfixes are also provided for developers, which can be applied with one click.

Severity	Privacy issue	Explanation	How to fix
Error	Missing valid annotation	The required annotation is not attached to the target variable, or when the annotation is not completed with valid values	Coconut can generate prefilled annotation skeletons
Warning	Inconsistent annotation	The values in the annotation are inconsistent with what is speculated from the code. For example, such a warning will be generated if the user specifies PRIVATE_ACCESS in the Storage annotation while Coconut finds the data is actually stored in the public area by analyzing the API parameters.	Coconut can automatically modify the annotation to match the actual code behavior or automatically modify the API call to adapt to the value specified in the annotation
Warning	Potential violation of purpose limitation	The current data collection does not match the best practice for the purpose specified by the developer. For example, such a warning will be generated if the user collects a hardware ID (e.g. MAC address) for tracking user behavior within the app.	Coconut can automatically modify the code to acquire a UID that fits the purpose specified by the developer best
Warning	Implicit data collection that may not be expected by users	The visibility is specified as IN_BACKGROUND in the corresponding source annotation, or the background data collection is initiated by a third-party library whose behavior is already known	The tooltip of the warning reminds the developer to provide explicit explanation of this data collection behavior in the privacy notice for users.

Table 4.2: Background information of the lab study participants regarding Android development (yrs of exp: how many years of experience do they have in Android development; active?: whether they were actively working on any Android development project as a software developer; pro?: whether they had worked as a professional Android developer; all apps: how many Android apps they had developed; playstore apps: how many apps were published on the Google Play store; made privacy policy?: whether they had the experience in making privacy policies.)

ID	yrs of exp	active?	pro?	all apps	playstore apps	made privacy policy?
C1	4	✓	✓	>5	3	✓
C2	2.5	✓	✓	3	1	
C3	3	✓	✓	3	3	
C4	4			4	2	
C5	2	✓		>5	3	
C6	4		✓	>5	8	
C7	1			3	0	
C8	1	✓		3	0	
C9	1	✓		3	0	
E1	4	✓	✓	>5	4	
E2	1	✓	✓	2	0	
E3	4	✓		>5	12	✓
E4	3	✓	✓	3	1	
E5	1			1	1	
E6	1	✓		3	0	
E7	4			3	0	
E8	2.5		✓	3	1	
E9	2			3	2	

Table 4.3: Overview of lab study results. C1-C9 are the nine participants in the control group, and E1-E9 the experimental group using Coconut. Some developers did not complete all the tasks, denoted '–'. Practices better for privacy are in bold. The meaning of each column are as follows. "warm-up": time spent on warm-up task; "main": time spent on main weather app; "weather loc.": granularity of location for weather info (fine-grained: about 10m accuracy, coarse-grained: about 100m); "ad loc.": granularity of location collected by AdMob library; "storage": whether data stored locally is only accessible to this app (private) or also other apps (public); "UID": type of unique identifier in the weather app (GUID is custom globally unique IDs. Google Instance ID and GUID are user-resettable, app-level unique identifiers which are recommended. Android ID and Telephony ID are hardware identifiers in some versions of Android which are more privacy invasive. See more at [53])

ID	warm-up	main	weather loc.	ad loc.	storage	UID
C1	22'34"	32'25"	**coarse**	fine	**private**	**Google Instance ID**
C2	19'54"	38'47"	fine	fine	**private**	**GUID**
C3	38'27"	37'30"	fine	fine	public	Android ID
C4	–	53'19"	fine	fine	**private**	**GUID**
C5	–	–	**coarse**	fine	**private**	–
C6	–	–	fine	fine	–	pseudo UID
C7	–	–	fine	fine	**private**	Telephony ID
C8	–	–	fine	–	–	–
C9	–	44'19"	fine	fine	–	**GUID**
E1	15'14"	60'	fine	fine	**private**	**GUID**
E2	–	–	**coarse**	**coarse**	–	–
E3	22'51"	30'34"	**coarse**	fine	**private**	**GUID**
E4	25'27"	51'14"	fine	fine	**private**	**GUID**
E5	–	–	fine	–	–	–
E6	–	–	**coarse**	–	–	–
E7	17'29"	34'29"	**coarse**	**coarse**	**private**	**Google Instance ID**
E8	38'58"	58'44"	**coarse**	**coarse**	**private**	**GUID**
E9	–	–	**coarse**	**coarse**	**private**	–

Table 4.4: Percentage of correct answers to factual questions regarding apps' behavior. For the fraction after the percentage, the denominator indicates the number of people that answered this question, and the numerator indicates the number of people who answered it correctly. The last row ('Total') calculates the percentage by directly accumulating the number of total answers and total correct answers in each group.

	Control	Coconut
What is the granularity of the location data that you collected for getting the local weather info?	66.67% (6/9)	88.89% (8/9)
What is the granularity of the location data that you collected for the analytics purpose and were buffered on the phone?	75.00% (6/8)	100.0% (6/6)
How frequently does the app access location data for getting weather information?	77.78% (7/9)	75.00% (6/8)
How frequently does the app store location data for analytics purposes?	75.00% (6/8)	100.0% (6/6)
Are users able to reset the unique identifier that you choose?	100.0% (7/7)	80.00% (4/5)
What is the scope of use of the unique identifier that you chose?	71.43% (5/7)	100.0% (5/5)
Did you store the location data in a way that is only accessible to your app?	50.00% (3/6)	83.33% (5/6)
Did you store the unique identifier in a way that is only accessible to your app?	83.33% (5/6)	80.00% (4/5)
Where will the location data be transmitted to in the current version of the app? (Please list all places that you can think of)	11.11% (1/9)	88.89% (8/9)
Total	66.67% (46/69)	88.14% (52/59)

Chapter 5

Honeysuckle: Annotation-Guided In-App Privacy Notice Generation

---

This chapter was adapted from my published paper:

- Li, Tianshi, et al. "Honeysuckle: Annotation-guided code generation of in-app privacy notices." Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies 5.3 (2021): 1-27.

5.1 Overview

Effective privacy notices are essential for helping users make informed privacy decisions. In the context of mobile apps, usable privacy researchers have advocated various in-app privacy notices that are concise, informative, and user-friendly [20, 145]. This body of work, along with increasing consumer concerns about privacy and an encouraging trend in recent privacy laws (e.g., GDPR) and policies (e.g., Google Developer Policies), has led to smartphone operating systems introducing many privacy features such as runtime permissions, background location access alerts, and visible indicators of ongoing audio recording.

However, while system features like these can increase data transparency to some extent, they are often coarse-grained [171] and limited to data types predefined by the operating system [12, 13]. As such, this leaves much of the responsibility of providing effective in-app privacy notices on app developers' shoulders, which is a difficult task for several reasons. First, it has been repeatedly found that developers tend to focus more on security factors over aspects of privacy such as data transparency [71, 98, 148]. Second, developers' incomplete understanding of the concept of "privacy" may create inherent barriers to the adoption of techniques like in-app privacy notices. Third, the design and implementation of effective in-app privacy notices can be a challenging task even for developers who care about transparency issues. For instance, devel-

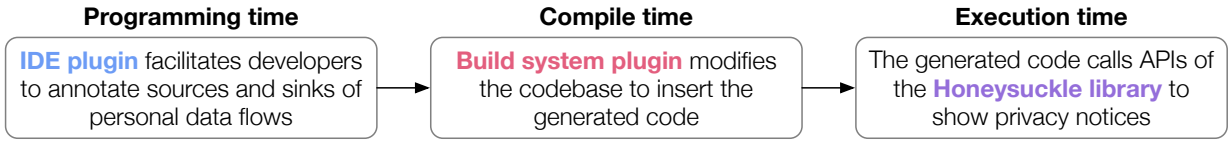


Figure 5.1: Honeysuckle consists of three components: an IDE plugin, a build system plugin, and a privacy notice library. As illustrated by the flowchart, the three components each play a role at the programming time, the compile time, and the execution time of an app's life cycle to help developers implement in-app privacy notices. The IDE plugin facilitates developers to annotate the code, and the annotation will later be used to generate privacy notice code. The generated code will be inserted into the app during compile time by my build system plugin. Finally, the generated code calls APIs provided by my library to show privacy notices during runtime.

opers do not always know all data practices of their apps, especially when the app behavior gets complicated and involves the use of third-party libraries [30, 154], or when the data practices are not well documented [95]. Fourth, developers have limited knowledge about design choices and best practices for privacy notices, which has led to uninformative or even misleading ones in real-world apps [111, 129]. Lastly, there is little developer support to help developers keep pace with ever-changing legal and policy requirements, and systematically improve their apps' data transparency [98].

In this chapter, I present Honeysuckle, a developer tool consisting of a plugin for the Android Studio IDE, a plugin for the Gradle build system, and a privacy notice library (Figure 5.1). Combined, these components jointly support a new approach of implementing in-app privacy notices, namely *annotation-based generation of privacy notices*. Instead of treating each privacy notice as a separate feature to implement, Honeysuckle only requires developers to (a) use Java's existing annotation feature to annotate *each data source and sink* with descriptions of data practices, e.g. declaring the purpose of data use; and (b) optionally modify a configuration file to fine-tune the frequency and visibility of the privacy notices. Honeysuckle can then transparently generate code at compile time that links to my privacy notice library, implementing the privacy notice UIs based on these privacy annotations, the configuration file, and code analysis results. Figure 5.2 illustrates how annotations are used to generate privacy notices and demonstrates a list of example privacy notices that can be generated using Honeysuckle.

My novel annotation-based approach is grounded in three key design considerations that can benefit both developers and users. First, it gives developers control over the generated UI to provide context information and configure the behaviors of privacy notices at a relatively fine granularity. Second, it reduces the implementation and maintenance cost of in-app privacy notices by minimizing redundant information and providing alerts for inconsistency. Third, my IDE plugin offers features that can help developers track and annotate all places that collect, store, and transmit sensitive user data (i.e., data sources and sinks), which improves the accuracy and comprehensiveness of information conveyed in the privacy notices. Overall, I believe Honeysuckle demonstrates a viable path forward to reducing the barriers to adopting in-app privacy notices while also improving the consistency of in-app privacy notices across apps by different developers.

To evaluate Honeysuckle, I conducted a within-subjects programming study with 12 Android

developers. During the study, each participant was asked to implement and integrate in-app privacy notices (as demonstrated in Figure 5.2) for two popular open-source apps, a notification manager app with over 1M installs and a note-taking app with over 100K installs. A challenge, however, is that the current state of the art is to build these privacy notices manually, which would not be a good baseline for comparison since it clearly takes a great deal of time and energy to design and implement these notices. That is, I felt that I would not learn much against offering no support. Instead, I decided to compare my annotation-based approach (a relatively unfamiliar approach for participants) against simply using my own privacy notice library. I felt that this approach would be a reasonable approximation of how developers or large organizations might provide reusable privacy notices.

My results show that the annotation-based approach proposed by Honeysuckle significantly reduced the time and the cognitive load of implementing fine-grained, contextualized privacy notices for the two open source apps than using my library directly. Within the 40-minute allocated time, all 12 developers completed all the required privacy notices using the annotation-based approach, while only 2 developers completed the same task using the library. All participants considered the annotation-based approach much easier to use and learn than the library-based approach because the clear step-by-step guidance provided by my Honeysuckle IDE plugin effectively reduced the learning curve, the use of quick-fixes and code generation allowed developers to write much less code, and the use of annotations reduced redundancies and made the code easier to comprehend.

This work makes the following contributions:

- I present the design and implementation of Honeysuckle, an Android developer tool that offers *annotation-based generation of privacy notices*. Honeysuckle consists of an IDE (Android Studio) plugin, a build system (Gradle) plugin, and a privacy notice library.
- I conducted a controlled, within-subjects programming study to evaluate these two approaches of implementing in-app privacy notices supported by Honeysuckle. My results show that the annotation-based approach helped developers build in-app privacy notices faster with significantly lower cognitive load than manually implementing and integrating them using a library.

5.1.1 Motivating Example: Library vs. Annotation

Below, I offer two scenarios to help convey the intuition of how Honeysuckle's two approaches for in-app privacy notices work. Bao is an Android developer working on a social media app and has just finished two features using location data. The first feature silently accesses the latest GPS location when the user refreshes the news feed, to optimize the relevance of the results based on the current location (mainly implemented in the function `optimizeLocallyRelevantNews`). The second feature allows users to view their current location in a map and send the location to selected friends.

She aims to implement three types of in-app privacy notices to enhance data transparency, as recommended by the usable privacy literature. First, she wants to show a dialog that explains both purposes before location permission requests are made, to help users make informed decisions [106, 162, 171]. Second, she wants to show just-in-time notifications when the app

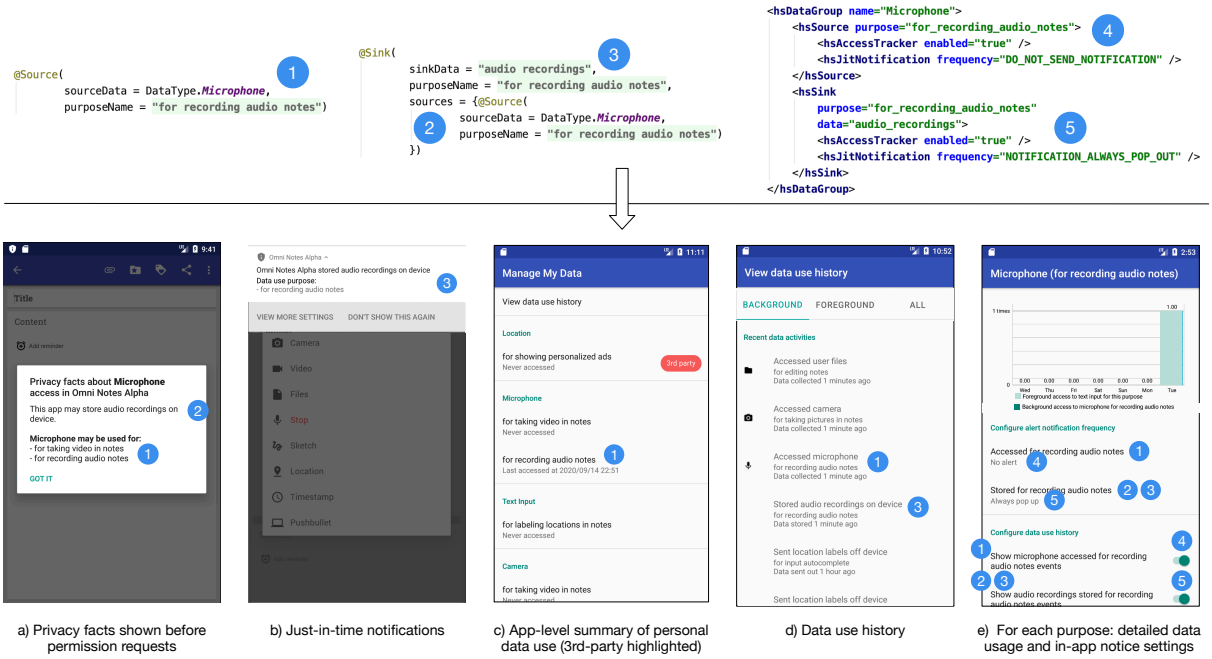


Figure 5.2: The top row shows the annotations and UI configuration XML that developers need to add or modify to generate privacy notices. The bottom row shows the five privacy notice features supported by Honeysuckle ("a" through "e"). I use the same number (1 through 5) to indicate which part of the annotation is used to populate which part of the generated UI, and how the configuration XML guides the presentation of the UI. The design of all these features were inspired by proposals from usable privacy research and privacy regulations. These examples demonstrate the breadth of privacy notices that Honeysuckle can generate with *one annotation* for each data source and sink.

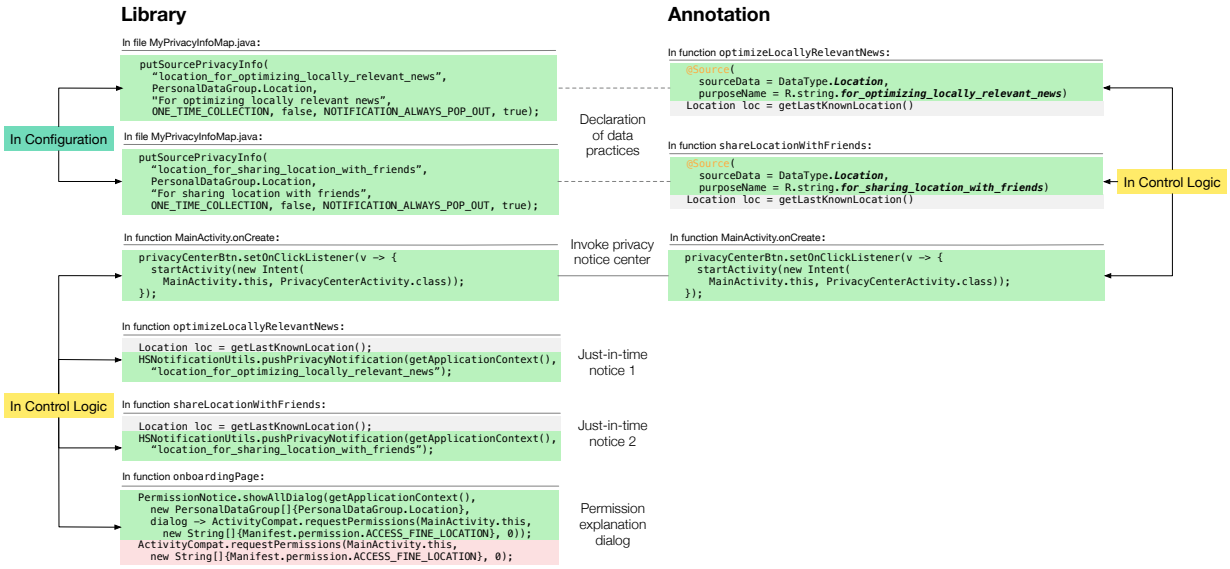


Figure 5.3: An illustration of code changes required to implement the same in-app privacy notices for two location features using the library and the annotation-based method. In the code, green highlights indicate lines that were added; red indicates lines that are removed; grey indicates lines that are unchanged. Between the two conditions, the dashed lines connect code that conceptually serves the same goal, and the solid line connects the same code. We can see that using annotations substantially reduces the amount of code that developer needs to write. The annotation-based approach also supports a more uniform format while the library-based approach requires developers to handle low-level implementation tasks using different APIs and declare data practices in a different place from the actual sources and sinks (outside the control logic).

accesses location data to notify users about the data collection behavior and the corresponding purpose [145]. Lastly, she wants to implement a privacy notice center to provide an overview of all data practices and the purposes of this app, and allow users to configure the notice visibility and frequency based on their preferences [145]. Figure 5.2 shows the reference designs of these privacy notices.

One straightforward way to implement in-app privacy notices is to treat different types of privacy notices as different features and handle them individually. Hence, in the first scenario, I demonstrate how Bao can implement the three target features using the Honeysuckle library APIs. The "Library" column of Figure 5.3 illustrates the main code changes that Bao needs to make. She needs to manually track code that handles location data collection and location permission requests and modify the code to implement the privacy notices. She needs to declare information about the data practices such as the purpose in a separate file from the actual data source or sink. She also needs to choose the proper APIs for different types of privacy notices (e.g., `HSNotificationUtils.pushPrivacyNotification` and `PermissionNotice.showAllD` in this example), and to integrate the privacy notice center into the app. This entire process involves a lot of manual work. Informally, I observe that while this library approach is feasible, there will be challenges in terms of implementation and maintenance for apps with more data practices, especially if the types of required privacy notices change [10, 98]. I also re-iterate that currently for Android, there is no support for building these kinds of privacy notices.

The second scenario demonstrates an alternative approach, namely shifting from "implementing the features" to "declaring the data practices". This time Bao only needs to declare the data practices by adding annotations to code that handles the collection of location data in different contexts and manually integrate the privacy center so she can decide where to embed it. Honeysuckle can then automatically generate other code for in-app privacy notices (which still uses my library under the hood) and insert it at the right place in the source code. This method is a variation of the classic idea of "model-based user interface design" [121], which aimed to reduce developers' workload by having them focus on *what* rather than *how*. In addition, my Honeysuckle IDE plugin runs an always-on code analysis to detect code signatures of APIs that may be related to the collection, storage, and transmission of sensitive user data, so as to prompt developers to add these annotations in real time and provide quick-fixes to streamline the task. This can further reduce the burden of manually tracking all data practices and may improve the coverage and accuracy of the in-app privacy notices accordingly.

5.1.2 Background: Why Developers Can't Implement In-App Privacy Notices Right?

Currently, developers are responsible for implementing in-app privacy notices with proper content, in proper formats, and with proper timing. However, prior research has found that developers often lack awareness, knowledge, and motivation to comply with privacy requirements [71, 95, 98, 148], which leads to several challenges to the adoption of in-app privacy notices.

The first challenge is developers' lack of awareness and knowledge regarding the importance of data transparency and how to use in-app privacy notices to improve transparency. Sheth et al. [148] conducted an online survey with developers and found that they preferred technical

measures like data anonymization to mitigate privacy risks, perceiving privacy policies as less effective. Li et al. [95] interviewed Android developers and found most of them do not know the best privacy practices listed in the official Android documentation. Liu et al. [111] analyzed the custom rationale messages of permission requests in Android apps and found that a large portion of these messages merely paraphrased the permission request, which corroborated the findings of the interview study. Altogether, these findings suggest that many developers may not have enough awareness of the benefits of in-app privacy notices nor knowledge of how to design them.

The second challenge is in having a comprehensive and precise understanding of data practices across the entire app. Schaub et al. [145] clearly indicated that the first step in designing effective privacy notices is to thoroughly understand the data practices of the app. However, this goal can be hard for developers, even for apps that they participated in developing. One reason is that implementation details are not documented on time [95, 109]. Another is that developers are not always aware of the behaviors of third-party libraries, which might automatically collect user data [30, 154]. Overall, not addressing these issues may result in uninformative or even misleading privacy notices that are harmful to users' privacy.

The third challenge is that implementing in-app privacy notices increases implementation and maintenance costs. Although the design of in-app privacy notices has been studied a lot, there is little research in the implementation of these in-app privacy notices. There are also few developer tools to facilitate implementation, other than libraries that help handle permission requests required by the operating system [11] or comply with recent privacy laws such as requesting for consent before data collection [14]. Furthermore, the currently available developer tools have narrow purposes, such as "providing the consent dialog required by GDPR for the Google Admob library", which offers little help with maintaining an app if there are new privacy laws or app store policies to comply with. In fact, recent research examining an Android developer subreddit found that developers complained about this type of unnecessary cost and had negative attitudes towards complying with privacy requirements [98].

In this work, I present Honeysuckle as a tool to help developers with the design and implementation of in-app privacy notices. The design of Honeysuckle is largely driven by the goal to address the three challenges above. In particular, my annotation-based code generation allows developers who are not experts in privacy to follow best practices in privacy notice design because those best practices are already embodied in the generated interfaces. Honeysuckle also saves on implementation and maintenance work because low-level implementation details are hidden from developers, with some details that can be customized using annotations and a configuration file. Furthermore, my IDE plugin can automatically scan all potential sources and sinks and prompt developers to annotate them to ensure comprehensive coverage in the generated privacy notices.

5.2 Honeysuckle Design and Implementation

To help developers who are not privacy experts implement in-app privacy notices following the best practices, I designed and implemented Honeysuckle. My design is mainly driven by the three goals:

D1 Give developers control over the privacy notice UI.

D2 Reduce the implementation and maintenance cost of privacy notices.

D3 Increase the accuracy and coverage of privacy notices regarding the actual app behavior.

D1 is a basic requirement as developers need to have sufficient flexibility to customize the generated interfaces to tailor to the app behavior. In addition, this design goal is also crucial for leveraging developers' knowledge about their apps to generate in-app privacy notices that can accommodate different users' privacy preferences under different contexts [106]. **D2** is motivated by the implementation and maintenance cost challenge as explained in Section 5.1.2. Since developers usually treat privacy as a secondary concern [28, 95], minimizing the workload of implementing privacy features is an important requirement for voluntary adoption. **D3** is motivated by the challenges of developers lacking awareness and knowledge about privacy and often not having a precise and up-to-date understanding of their own apps' data practices as discussed in Section 5.1.2.

Honeysuckle consists of three components: an Android Studio plugin, a Gradle plugin, and a privacy notice library. Developers can use the Honeysuckle library alone to implement privacy notices in a conventional way, which gives developers control (**D1** ✓), albeit at the cost of still significant implementation and maintenance effort (**D2** ✗). Developers may also not be able to keep track of all data practices due to challenges detailed in Section 5.1.2, which can result in inaccurate or incomprehensive privacy notices (**D3** ✗)

Alternatively, developers can use the entire Honeysuckle system to annotate the code and modify a configuration file to generate code that calls the library to create in-app privacy notices. This method greatly reduces the implementation and maintenance effort (**D2** ✓) and provides the same control over the generated UI. (**D1** ✓). Furthermore, my Honeysuckle Android Studio plugin provides features that help increase the accuracy and coverage of data practices declared in the annotations to improve the generated privacy notices (**D3** ✓).

In the following, I first present the design of the in-app privacy notices supported by Honeysuckle. Then I describe how I designed Honeysuckle to support the two approaches mentioned above to implement these in-app privacy notices. I describe the implementation of the main modules of Honeysuckle at the end.

5.2.1 Privacy Notice Designs and Rationales

I present the design of the in-app privacy notices supported by Honeysuckle and explain their relationship with best practices advocated by prior literature. My design decisions concern two aspects: timing and content.

Timing

According to Schaub et al. [145], the timing of these notices is a key consideration. I designed three types of privacy notices corresponding to three timing options described in that paper: *at setup*, *just in time*, and *on demand*.

T1 *at setup*: A dialog shown before permission requests that proactively explains the purposes of the permission request and the related data egress and retention behavior (Figure 5.2

(a)).

T2 *just in time*: A just-in-time notification that appears immediately after the data is accessed or sent off the device by the app (Figure 5.2 (b)).

T3 *on demand*: A privacy center that contain a hierarchical overview of all sensitive data practices in this app. Unlike the previous two types, the privacy center shows up on demand, so it can contain more in-depth information about the apps' data practices (Figure 5.2 (c)(e)(f)).

Content

As discussed in Section **??**, privacy notices should reduce potential notification fatigue and habituation by highlighting unexpected and sensitive data practices. my design is based on recommendations of prior research, common practices of real-world systems, and requirements of privacy laws. I detail each design and the supporting literature below.

I1 Data use purpose: Prior research has recommended apps to explicitly explain their purposes of data use to users especially before permission requests to help them make informed decisions and also feel more comfortable with sharing data with the app [106, 162, 171]. These recommendations were later adopted by the Android official documentation about best practices regarding permission requests [51].

I2 Third-party data use: Research has showed that 3rd-party data use causes most privacy concerns because they are unlikely to be expected by users and are often used for purposes such as targeted advertising that do not provide direct benefits to users [42, 156]. Many privacy laws and app store policies now require developers to explicitly specify data sharing with third parties in privacy policies.

I3 Data egress and retention: Data egress and retention can both lead to higher secondary data use and data breach risks. Researchers have proposed program analysis techniques to automatically detect [64] and alert users about these sensitive data practices [113].

I4 Background access: Background data access is likely unexpected by the user, and therefore deserves more explicit alerts [138, 145] For example, both iOS and Android allow users to only grant permission to location collection in foreground. iOS also occasionally alerts users about background location collection.

I5 Data access frequency and time: Prior research has demonstrated that showing the when and how frequently an app has accessed sensitive data can nudge users to improve their awareness of sensitive data practices and take more action to protect their privacy [20, 29].

5.2.2 Honeysuckle Library-Only Mode

I designed a library for implementing all the proposed features. Figure 5.3 shows an example of how to use the library to implement in-app privacy notices for two location features in a social media app. Developers need to use three different APIs to implement privacy notices shown at different times.

For **T1**, developers need to replace all permission requests with `PermissionNotice.showAllDialog` function calls to show the permission explanation dialog.

For **T2**, developers call the function `HSNotificationUtils.pushPrivacyNotification` after all data sources and sinks to show a just-in-time notification when the data is collected or sent off the device. A string identifier of the current data usage needs to be passed as a parameter to retrieve the purpose of the current data usage and configurations about how to show the privacy notice declared in a separate file (see the `MyPrivacyInfoMap.java` in Figure 5.3 as an example; details will be provided later).

For **T3**, developers call the system API `startActivity` and pass `PrivacyCenterActivity.class` as a parameter to open the privacy center. `PrivacyCenterActivity` is an activity class defined by the Honeysuckle library. It can automatically load data practices declared in the configuration file and populate the privacy center with these data practices at runtime. Note that developers need to manually call this API both when using the library alone and when using annotations to generate privacy notices, because developers should be able to determine where to embed the entry point of the privacy center in their apps.

In addition to the three APIs, developers also need to declare what information will be presented in the privacy notices (based on information types listed in Section 5.2.1) and default configurations about how to show the privacy notices for each source and sink using the `putSourcePrivacyInfo` and `putSinkPrivacyInfo` APIs. Each API call creates and registers an instance in a *global privacy information map* that is used to populate privacy notices at runtime, so the first parameter of the API call needs to be a unique identifier for retrieving the instance. Developers need to pass five parameters to `putSourcePrivacyInfo`: the data type by choosing a predefined option, the purpose of data use, whether the data is used by a third-party library, whether the data will be stored or sent off the device, and whether it is a one-off, periodic, or continuous data collection. For each data sink, developers need to provide all of the parameters above, as well as a list of the identifiers of all the sources of the data flow to the sink, and a string that describes the type of data that will leave the sink node. My Honeysuckle library can automatically determine whether the app collects sensitive data in the background, hence developers do not need to handle this themselves.

Regarding how and which type of privacy notices is shown, developers need to pass two parameters. One is an enum parameter that determines how frequently the just-in-time notification will pop up (e.g., one possible option is `NOTIFICATION_ALWAYS_POP_OUT` as demonstrated in Figure 5.3). The other is a boolean parameter that indicates whether to record the timestamps of a data collection, data retention, or data egress activity and show them in the privacy notice center (see Figure 5.2 (c) (d) and (e)).

5.2.3 Honeysuckle Annotation-Based Code Generation Mode

Although the Honeysuckle library can help generate in-app privacy notices, it comes at a significant development cost due to all the repetitive code that has to be added. For instance, developers need to explicitly specify information already encoded in the source and sink API such as the data type, whether it is a third-party API, and whether it is a one-off or periodic data access. Furthermore, using the Honeysuckle library alone leads to higher maintenance effort since the code that handles privacy UI is scattered across the project. These limitations suggest that the

library approach may not effectively achieve design goal **D2**. In the meanwhile, using the the Honeysuckle library alone may also not satisfy the design goal **D3** because developers may not recall all the data practices of their app and not keep up-to-date documentation [30, 95].

To address this problem, I propose *annotation-based code generation*. The key idea is to move from having developers directly implement in-app privacy notices to having them declare relevant privacy-related information using *custom Java annotations and a UI XML configuration*. my *build system plugin* can then automatically generate code that implements in-app privacy notices based on that information at compile time.

To help developers add annotations that cover all sources and sinks and include accurate information (addressing **D3**), I designed the Honeysuckle *IDE plugin* to give developers real-time feedback to help them identify missing annotations or annotations with errors. My IDE plugin also provides several automation features, such as quick-fixes to annotation errors and automatically generating a default UI XML configuration for each source and sink. These features further streamline the actions developers need to do at programming time, thus helping with **D2**.

Annotation and UI XML Configuration

I put the information related to data practices in annotations that are directly embedded in the control logic right next to the corresponding data sources and sinks and put the information about the UI behavior in a separate UI XML configuration file (see examples in Figure 5.2). The annotations and the UI XML configuration together serve the same goal of the privacy information map in the library mode (Section 5.2.2), while the new method leverages the software design principle of separation of concerns to result in code that is easier to read, maintain, and analyze.

Table 5.1: `@Source` and `@Sink` annotation definition in Honeysuckle.

Annotation type	Annotation fields
@Source	sourceData (DataType enum value) purposeName (int, String resource ID)
@Sink	sources (@Source list) sinkData (int, String resource ID) purposeName (int, String resource ID)

Table 5.1 demonstrates the definition of the two main annotations: `@Source` and `@Sink`. While programming, developers need to annotate the sources and sinks of the sensitive data flows in their app using these two annotations. The two annotations *only* need the purposes of data use and the connection between sources and sinks from developers, because other information in privacy notices can be automatically inferred and inserted into the generated code at compile time.

I design the UI XML configuration to provide developers with some options to control how to show the privacy notices at the granularity of `<data type, purpose>` pairs. Note that multiple API calls can be used collectively to achieve the same purpose. In my terminology, they are considered as the same sources and sinks.

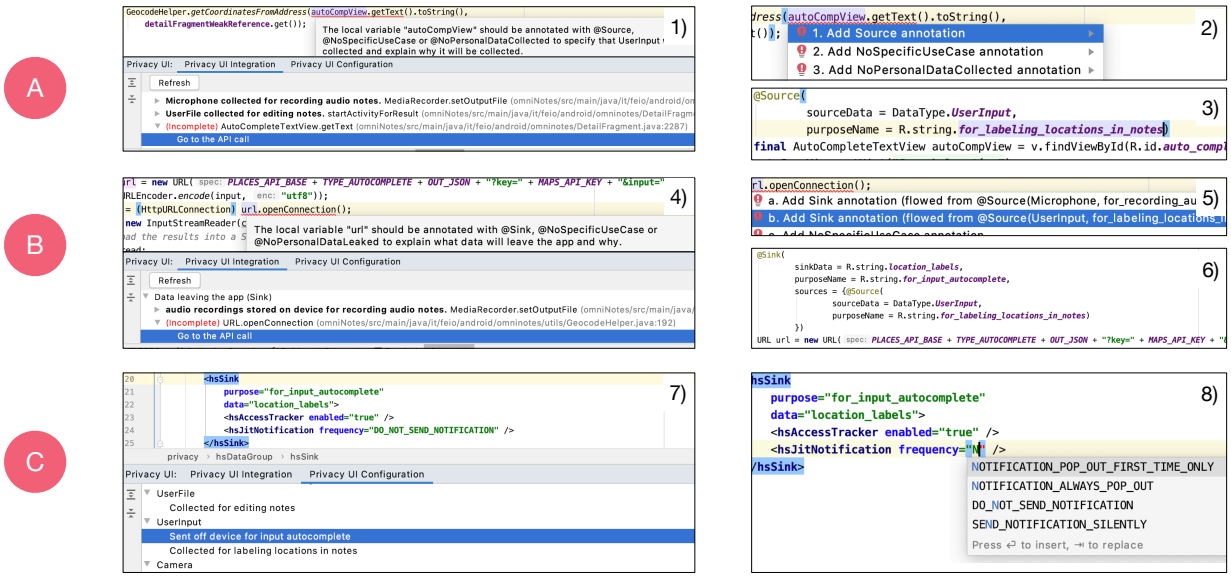


Figure 5.4: This figure demonstrates the main features of the Honeysuckle IDE plugin via an example use case. First, the app developer opens the `Privacy UI Integration` panel (A-1) to see what code accesses sensitive user data (i.e., data sources) as detected by the plugin. Then she can use the quick-fix "Add Source annotation" (A-2) to add an annotation for each of these sources and complete required fields in the annotation (e.g., purposeName) (A-3) to resolve the "(incomplete)" errors. Similarly, she needs to annotate "sinks" that may store or send sensitive user data off device (B-4) with the help of quick-fixes (B-5). To indicate how data flows, she also needs to explicitly specify all sources (represented by the source annotations) that may reach this sink (B-6). Lastly, she can modify an XML configuration file automatically generated by the IDE plugin to fine tune the UI behavior (C-7). For example, she can adjust whether to show JIT notification for each source and sink to avoid overwhelming users with unimportant notifications (C-8).

Honeysuckle IDE Plugin

The two main design goals of the IDE plugin are to help improve the accuracy and coverage of annotations and streamline the editing of annotations and the UI XML configuration. To achieve this goal, my IDE plugin provides developers with step-by-step guidance to walk through a usual workflow for implementing in-app privacy notices using the annotation-based approach as demonstrated in Figure 5.4.

This workflow can be roughly divided into three steps (corresponding to A, B, C in Figure 5.4), although there is no strict order restriction. In the first two steps, developers can see all the source and sink API calls that remain to be annotated marked in code and listed in the "Privacy UI Integration" panel (Figure 5.4 (1) and (4)). Then they can use the quick-fixes to add a partially filled `@Source` (or `@Sink`) annotation to the source (or sink) code. There are three design considerations for the quick-fixes that I want to highlight. First, for the `purposeName` field, my quick-fix will generate a "for_" prefix to nudge developers to use expressions like *"for labeling locations in notes"* that provide concrete explanations of the data use purpose rather

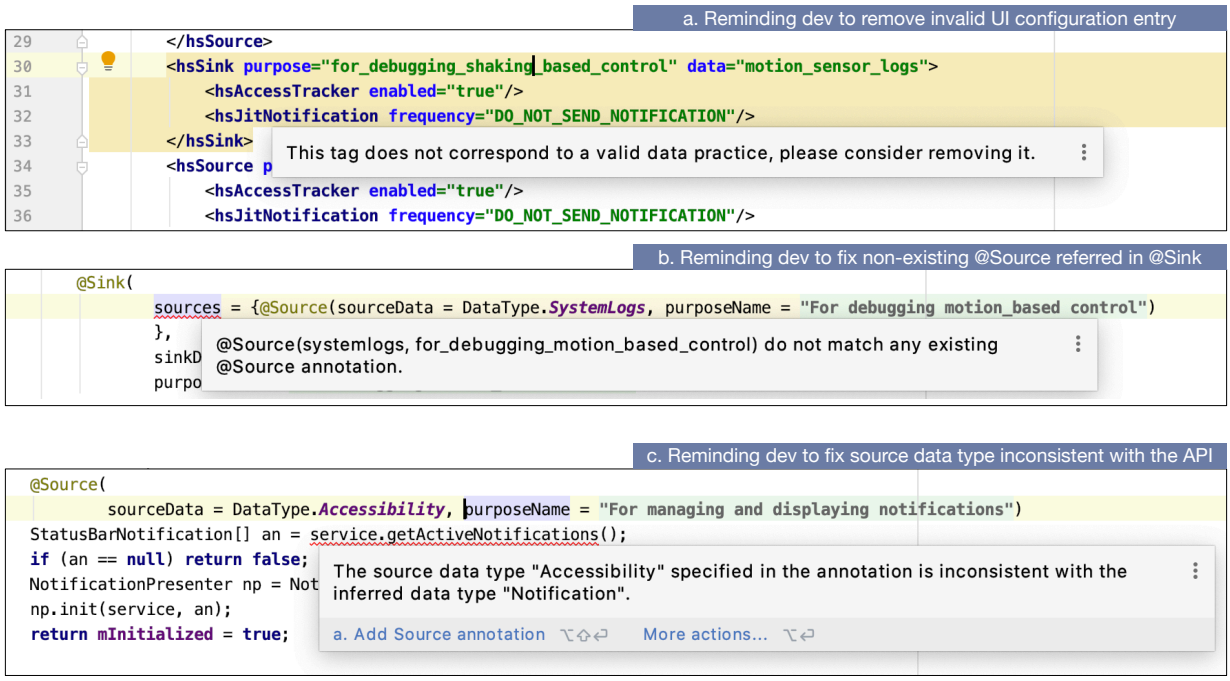


Figure 5.5: This figure demonstrates three types of auto-checks that can be conducted by the Honeysuckle IDE plugin. The general design principle is that the IDE plugin only aims to provide timely reminders to developers about annotations and UI configuration entries that may be invalid or outdated, and expects the developers to take the initiative to fix the issue.

than simply paraphrase generic permission requests [111]. This consideration is also explained to developers in error messages for missing annotations. Second, for each sink, I have multiple quick-fixes that each generate a `@Sink` annotation with an existing `@Source` annotation filled in the `sources` field (Table 5.1). I also have quick-fixes to add more `@Source` annotations to `@Sink` annotations. Third, since false positives are inevitable in automatically detected sources [77] and sinks [24], I allow developers to use `NoPersonalDataCollected` to explicitly indicate that sensitive user data is not collected at this point to suppress the alerts. However, there is a risk that developers may provide false information and I discuss potential mitigating approaches in Section 5.4.4.

In the third step, developers can use the "Privacy UI Configuration" panel to examine the current UI XML configuration in the panel or in the original code. To minimize the code that developers need to write, my IDE plugin automatically updates the configuration file if new sources and sinks have been added. Therefore, developers only need to modify this file if they want to change certain options. Furthermore, code auto-completion is provided when developers edit the configuration file (Figure 5.4 (8)).

In addition to the usual workflow, my IDE plugin also runs code inspection continuously to help developers identify and fix invalid or outdated annotations and UI configuration entries (Figure 5.5). To raise developers' awareness, annotations that contain these errors will be both marked in the code and indicated in the two panels. To achieve a balance between convenience and control, I only show the errors and provide some quick-fixes to help resolve the errors, thus

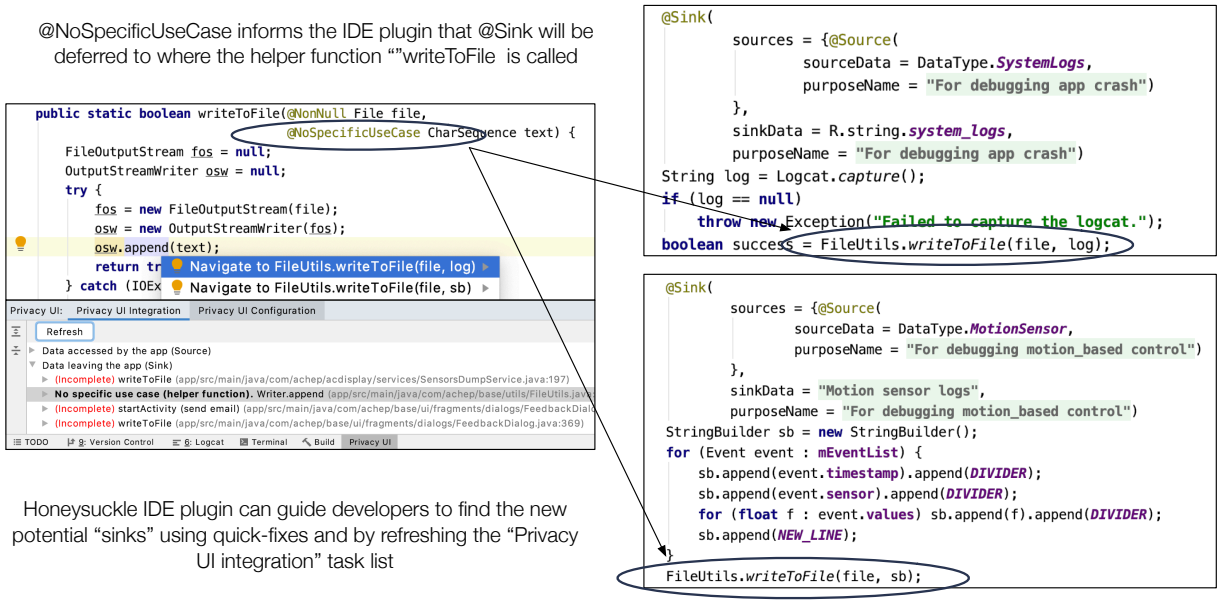


Figure 5.6: Annotation and IDE design example: Sometimes an app might have helper functions to collect, store, or transmit data, which leads to a tricky case that one sensitive API call can be used for multiple purposes. In this example, the app creates a helper function “writeToFile” that calls a system API “osw.append(text)” to write data to files. However, because the helper function is called from two different contexts which store different data for different purposes, directly adding a @Sink annotation to the “text” variable is not sufficient for distinguishing between these two contexts when the app is running. Therefore, I propose a new annotation @NoSpecificUseCase to allow developers to provide a cue to Honeysuckle to skip the @Sink requirement for the current level of the function call stack, so developers can annotate @Sink for helper function calls to distinguish data use in different contexts.

relying on developers to take the initiative to make the actual changes.

During the development of Honeysuckle, I observed that some developers may create helper functions that are often called under different contexts for different purposes. In this situation, directly annotating the source or sink in the helper function will have to conflate the different purposes and therefore fail to provide contextualized privacy notices. In the following, I use an example of a `writeToFile` helper function (Figure 5.6) to demonstrate how my Honeysuckle IDE plugin help handle this task using a new annotation: `@NoSpecificUseCase`.

My method is analogous to a Java method throwing an exception and require the caller to handle things. When a developer chooses to use the `@NoSpecificUseCase` annotation to indicate that the original sensitive sink API can be used for multiple purposes in different situations, the IDE expands the sink API list to also include the helper function itself. The IDE plugin then analyzes which parameter of the helper function can reach the initial sink target variable (`text`), and identifies the second parameter as the new target variable of the helper function. Then the developer can either use the quick-fixes to navigate to the helper function calls or refresh the “Privacy UI Integration” to reveal additional tasks due to the addition of the helper function as a new sink. This can be a recursive process in which developers can keep adding

multiple `@NoSpecificUseCase` until a specific use case is available. For `@Source` annotations, the process is similar except that the return value of the helper function will be treated as the new target variable.

Honeysuckle Build System Plugin

As alluded earlier, my build system plugin can generate code for providing in-app privacy notices using the Honeysuckle library based on the annotations, the UI XML configuration, and the code analysis results. A specific design goal of my build system plugin is that when generating the in-app privacy notice code, there should be as little impact on the programming, debugging, and execution of other code as possible. One related design choice is that the mapping between the line number and code should remain the same after modification, so that developers can debug the app with the source code as usual. Towards this end, Honeysuckle appends all the additional code for each source and sink in the same line.

5.2.4 Honeysuckle Implementation

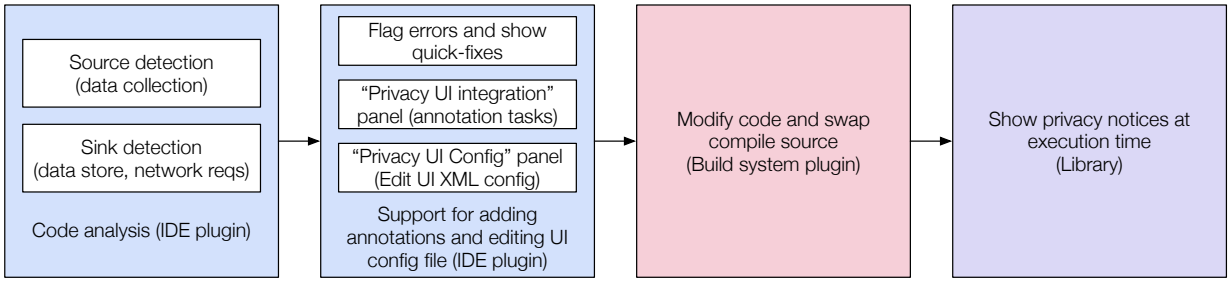


Figure 5.7: Honeysuckle system overview. The three colors correspond to the three component: the IDE plugin, the build system plugin, and the library.

My system consists of three main modules: an IDE plugin (implemented for Android Studio), a build system plugin (implemented for Gradle), and an Android library for creating in-app privacy notices. Figure 5.7 presents four main functionalities of Honeysuckle and I introduce the implementation for each of them below.

IDE Plugin: Code Analysis

The code analysis subsystem of Honeysuckle was built on top of the analysis engine of Coconut [95], which supports real-time API call detection based on method signature matching. I select 568 source APIs and 76 sink APIs using the same criteria as in Coconut and feed their method signatures into the system. Source APIs are defined as APIs that directly trigger sensitive data collection and sink APIs are APIs that store or send data out of the phone. First-party APIs were selected by reviewing the Android documentation and Third-party APIs were selected using stacktraces of third-party libraries collected by Chitkara et al. [42]. Honeysuckle covers 18 sensitive data types, two of which were adapted from Coconut (location and unique id) and the other 16 types were added by us. The sensitive data types are listed in the appendices (Table **??**).

To evaluate the code analysis subsystem, I selected 75 open-sourced Android apps on GitHub by searching for recently updated repos (by July 2020) that contain a Google Play link and declare dangerous permissions in the manifest. I cross-checked the API calls detected by Honeysuckle against the API calls detected by doing a method signature search on the Smali code[1] of these apps and obtained the same results. This shows that Honeysuckle can accurately detect all known source and sink APIs. Furthermore, Honeysuckle detected 23 sensitive API calls per app on average ($median = 12$, $std = 40$) and detected at least one sensitive API call for 70 out of the 75 apps. This suggests that although I did not aim to optimize for coverage, Honeysuckle can already detect a sizable number of sensitive API calls so developers do not need to manually keep track of them. I discuss the limitations of Honeysuckle code analysis and potential enhancement in more depth in Section 5.4.3.

IDE Plugin: Support for Adding Annotations and Editing UI Configuration

My IDE plugin runs custom code inspections continuously using the API `BaseJavaLocalInspectionTool` to maintain a record of all the current data practices, annotations, and UI XML confifuration. With this information, I implemented the "Privacy UI integration" panel and the "Privacy UI Config" panel using the `ToolWindowFactory` API. The quick-fixes are implemented using the `LocalQuickFix` API. All the above APIs are provided by the IntelliJ Platform SDK.

Regarding the feature to automatically generate UI XML configuration for new sources and sinks, I chose to initiate the modification of the configuration file only after the user clicks the tab to switch to the "Privacy UI Configuration" panel. This method can help avoid generating configuration for intermediate editing results.

Build System Plugin

I chose to develop a plugin for Gradle, which is the most widely used build system for Android development. I implemented custom Gradle tasks and then packaged the tasks into a plugin that can be applied in any Android project by adding one line of code similar to importing a library[2].

As the app starts compiling, my Gradle plugin will send a compile start signal to a local server run in the IDE plugin so IDE plugin can dump the code analysis results. Then the Gradle plugin inserts privacy notice UI code in the project at proper places based on the sensitive API calls detected by the IDE plugin, the annotations provided by developers for these API calls, and the UI configuration file. After finishing compilation, the source code will be reverted back to the original version that only contains the annotations but not the privacy notice UI code.

Library

The main functionality of my Honeysuckle library is to provide APIs for different types of in-app privacy notices. The permission explanation dialog is implemented using the `AlertDialog`

[1] Smali is a human-readable intermediate code format generated by decompiling an app apk which expands every API call to its complete method signature.

[2] Developing Custom Gradle Plugins. https://docs.gradle.org/current/userguide/custom_plugins.html

system API. The just-in-time notification is implemented as an Android notification using the system API. The privacy center is mainly implemented as a `Preference` UI, which is a common way for Android apps to implement app settings. I used an open-source library `MPAndroidChart`[3] to implement the data access frequency diagram in the privacy center. I also defined my custom Java annotations (e.g., Table 5.1) in the Honeysuckle library.

5.3 Honeysuckle Evaluation: Comparing the Annotation and Library Approach

To evaluate the effectiveness of the two methods supported by Honeysuckle in helping developers implement fine-grained, in-app privacy notices for real-world projects, I conduct a within-subjects programming study with 12 Android developers to examine the following research questions:

RQ1 What are their effect on developers' time performance in implementing privacy notices?

RQ2 What are their effect on developers' cognitive load of implementing privacy notices?

RQ3 What are their effect on developers' perceived usefulness and usability?

RQ4 How do developers think about the two methods?

5.3.1 Participants

Out of the 12 participants, 11 were freelance Android developers recruited from a freelancing platform (Upwork). The other participant signed up for the study after seeing my recruiting ads posted on Twitter. In a pre-study survey, I asked participants to provide demographic information. 3 participants self-identified as female, and 9 as male. The average age is 31.3 ($std = 9.49, min = 22, max = 56$) years old.

I also requested my participants to provide their background in Android development. My participants reported 4.3 years of Android development experience on average ($std = 1.5, min = 2, max = 7$). 8 of 12 participants self-reported that they have released Android apps on an app store (e.g., Google Play, F-Droid). One participant answered that his most popular app achieved 10M+ installs, another participant 1M+ installs, and a third participant 10K+ installs. All participants considered themselves familiar with Java and Android Studio.

Lastly, I asked them about their prior experience in handling private data and using Java annotations in general. 11 out of 12 self-reported that they have created Android apps that access users' personal data (e.g., geolocation, device ID), with the other participant answering "I'm not sure". However, when choosing from a list of different types of privacy notices, most participants reported only creating privacy policies and requesting system permissions to provide privacy notices. This finding supports my working assumption that developers rarely adopt other types of privacy notices for their apps. Regarding the use of Java annotations, all participants have at least used some simple features such as adding `@SuppressLint` to clear IDE warnings/errors. 3 out of 12 participants reported that they have previously used annotations for code generation.

[3] https://github.com/PhilJay/MPAndroidChart

5.3.2 Control and Test Conditions

The test condition of my study is to use Honeysuckle to modify a real-world app to implement the five types of example privacy notices. The control condition is to use the library I developed for Honeysuckle to generate the example privacy notices introduced in Section 5.2.2. To minimize the carryover effect in within-subjects study design, I chose two popular open-source apps, AcDisplay[4] (a notification manager app) and OmniNotes[5] (a note-taking app) and ask each participant to work on different projects using the two methods. Both projects use Java as the main programming language and are of similar complexity. OmniNotes contains around 231 Java files and 21714 lines of Java code; AcDisplay contains around 331 Java files and 59489 lines of Java code. Regarding sensitive data use, OmniNotes contains 10 sources and 2 sinks, and AcDisplay contains 13 sources and 4 sinks.

Note that I chose to use the library as the baseline condition instead of asking developers to achieve the task without any support, because otherwise UI design and implementation issues would clearly dominate the amount of the time on task and it would be unequal comparison with annotations and do not generate insightful results. I searched for Android libraries that support developers to implement these three in-app privacy notices but to no avail, likely because there has yet to be sufficient need to drive the development of such libraries. Hence I asked participants to use the Honeysuckle library in the library condition, which is also what my generated code calls to show the privacy notices, to provide a relatively equal comparison between the two approaches.

I sequentially assigned participants to the following groups based on their signup order:

1 modify AcDisplay using annotations + modify OmniNotes using library

2 modify OmniNotes using library + modify AcDisplay using annotations

3 modify OmniNotes using annotations + modify AcDisplay using library

4 modify AcDisplay using library + modify OmniNotes using annotations

5.3.3 Procedure

I conducted my study remotely via Zoom because my participants resided in multiple locations. After the participant signed the consent form, the experimenter turned on screen recording to record the entire session for further analysis. The entire study took around 2.5 hours and mainly consisted of two programming tasks (corresponding to the two conditions). Developers had at most 40 minutes for each programming task, and then spent 5 minutes to fill out a NASA-TLX questionnaire [74] to measure the cognitive load and a Technology Acceptance Model Scale (TAMS) questionnaire [49] to measure the perceived usefulness and usability of my tool. For the two programming tasks, developers modified different apps to implement privacy notices using different methods. Since I wanted to evaluate the programming methods rather than examine developers code comprehension skills, I provided documentation of the apps' data practices containing all the information needed to make the privacy notices so the developer could focus on using the provided tools to achieve the goal.

[4] https://github.com/AChep/AcDisplay
[5] https://github.com/federicoiosue/Omni-Notes

Before each programming task, the experimenter used a toy app as an example to teach the participant how to use annotations and the library to achieve the task. Specifically, the experimenter first implemented privacy notices for half of the sources and sinks in the toy app, and then asked the participant to complete the rest of privacy notices. During this process, the experimenter answered all the questions the participant had regarding the method of the current condition. For the library condition, I ensured that developers had access to the example app code during the main programming task so they could refer to them when needed.

After completing the two programming tasks and filling out the surveys, the experimenter asked several follow-up questions regarding their thoughts about the two methods, such as which part of the tool was hard to use, what could be improved, and what they thought about the generated privacy notice UIs. Upon completion, each participant was compensated with 75 USD. This study was approved by my university's IRB.

5.3.4 Study Environment

I set up a virtual workstation using a VNC server on a Google Cloud virtual machine, so all participants can work on the programming tasks in this virtual workstation. The experimenter can view and interact with the same interfaces during the study to do live coding during the warm up tasks and to provide guidance to help developers quickly get used to this new environment.

5.3.5 Quantitative Results (RQ1-3)

I present the results to the first three research questions below.

Time on Task (RQ1)

All developers completed the task in the Honeysuckle condition in the allotted time, while only 2 developers (P7 and P9) completed the task when using the library. Furthermore, for these 2 developers, it took them longer time to finish when using the library than using the annotation. P7 used 22 minutes to finished the task using the annotation and 37 minutes using the library. P9 used 14 minutes to finished the task using the annotation and 27 minutes using the library.

Cognitive Load (RQ2)

I analyzed the six dimensions of cognitive load measured by NASA-TLX on a scale of 1 to 100 (lower is better) [74]. Honeysuckle achieved a statistically significant reduction in cognitive load using a paired t-test (p ¡ .001, t=5.09). Figure 5.8 shows that Honeysuckle helped reduce the cognitive load across all six dimensions, especially for mental load and perceived success.

Usefulness and Usability (RQ3)

I analyzed the perceived usability and usefulness of Honeysuckle and the privacy notice library measured using TAMS on a scale of 1 to 7 (lower is better, 1=extremely likely, 7=extremely

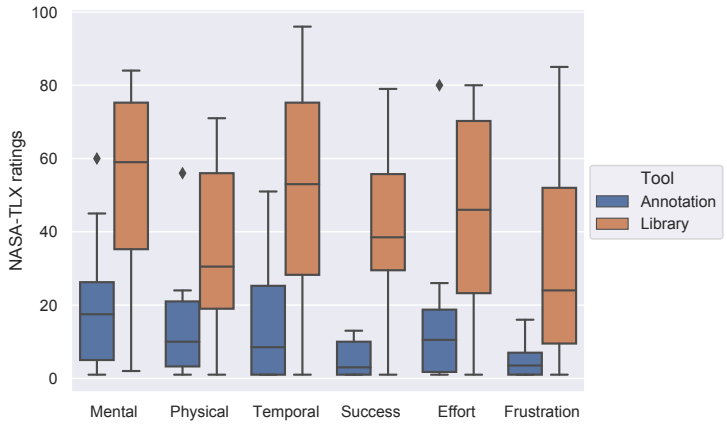


Figure 5.8: NASA-TLX ratings of the Annotation and Library conditions (lower is better). Using the annotation-based approach of Honeysuckle significantly reduces the overall cognitive load as compared to using the library to implement the same runtime privacy notices. The error bars represent the 95% confidence intervals.

unlikely) [49]. I then calculated the sum of all ratings in the usability subscale and the usefulness subscale and compare them between the two conditions. For both usability and usefulness, Honeysuckle received significantly better (lower) ratings under paired t-tests (for usability, p = .02, t=2.76; for usefulness, p = .02, t=2.67). The mean usability rating of Honeysuckle is 1.45 and the mean usability rating of the library is 2.56. The mean usefulness rating of Honeysuckle is 1.27 and the mean usefulness rating of the library is 2.30. Overall, developers perceived both tools to be useful and usable, and perceived Honeysuckle to be more useful and usable than the library.

5.3.6 Qualitative Feedback (RQ4)

One researcher transcribed the interviews, conducted thematic analysis following previous guidelines [36], and held regular meetings with other authors to review the analysis process and discuss the findings. The identified themes are categorized as *feedback on programming methods* or *feedback on the generated privacy notices*.

Feedback on Programming Methods

When comparing the two methods, all participants preferred the annotation method a lot than the library method. Without being asked, P2, P7, P10, P11 said they hoped to see a public release of Honeysuckle so they can use it for their own projects. For example, P7 said that *"I hope any of these can be launched. Anyone of them would be cool, but my preference would be the plugin."* This echos the fact that there is very limited developer support for building in-app privacy notices and suggests both the Honeysuckle library and annotation can improve app's transparency when also making developer's life a lot easier. Below, I present themes emerged from participants' explanations of their preferences.

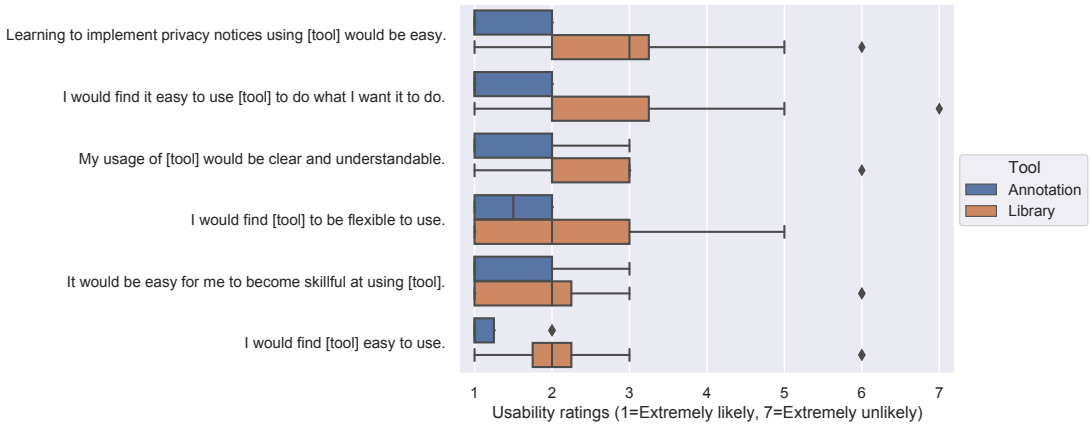


Figure 5.9: Perceived usability ratings of the annotation-based approach and the library (lower is better, 1=extremely likely, 7=extremely unlikely). Overall, the annotation-based approach was perceived as more usable than the library. The error bars represent the 95% confidence intervals.

Figure 5.10: Perceived usefulness ratings of the annotation-based approach and the library (lower is better, 1=extremely likely, 7=extremely unlikely). Overall, the annotation-based approach was perceived as more useful than the library. The error bars represent the 95% confidence intervals.

Productivity. Six participants (P1, P3, P4, P7, P11, P12) mentioned that they felt a lot more productive when using the annotation method than the library. Specifically, they felt using annotations allowed them to write less code and achieve the task faster. For example, P4 mentioned using annotations could *"avoid a lot of parameters for calling the APIs"*, which is because my annotation design reduces a lot of redundancies by automatically generating multiple types of notices using one annotation. Developers also praised the automated source and sink detection and quick-fixes offered by Honeysuckle IDE plugin. For example, P3 mentioned *"Just being able to go through some automated steps makes it more efficient to add annotations"*.

Mental Load. Nine participants (P1, P2, P4, P7-P12) said that using annotations helped reduce the mental load in many ways. For example, P8 loved the automated suggestions offered by the IDE plugin because *"It simply tells me where I need to add annotations and I don't need to consciously take effort to do that"*. On the contrary, using the library *"felt very tedious"* (P2) and they found it challenging to *"remember what API to use"* (P7). Moreover, using the library requires the developer to *"manually track many things at the same time"*, including naming the sources and remembering the names when defining connections between sources and sinks in the privacy info map (P9). When using annotations, this task is made much easier by the quick-fixes of the IDE plugin.

Learnability and Usability. Nine participants (P2, P4-P7, P9-P12) discussed the learnability and usability of the two methods. Some people found the annotation tool easy to learn. For example, P11 said *"The tool is new to me, but I didn't need to think much to figure out what to do. The plugin has made it very simple."* Conversely, some participants shared a slightly different feeling that *"the annotations may have slightly steeper learning curve, but after getting the hang of it, using it is way easier"* (P7). The difficulty was mainly in understanding how to specify the source in a sink annotation (P2, P10) and handle helper functions (P2, P5, P10). This result echos the fact that only three out of the twelve participants had used annotations for code generation before the study. However, the unfamiliarity with annotations did not seem to hinder the usage of annotations for privacy notice generation. P4 even said *"Normally I don't like annotations, but for this task I like it. It's very easy to use."*

Code Comprehension. Four participants (P1, P7, P8, P11) mentioned that using annotations made the code *"organized"* (P1) and *"very easy to read"* (P8). P8 further explained that *"all the configuration goes to the XML, so the annotation only keeps simple information"*.

Teamwork. P4 and P9 explicitly discussed different benefits of using annotations and the IDE plugin for teamwork. P4 liked the quick-fixes for generating annotation templates because it helps everyone in a team code in the same format. P9 was thinking about the situation when developers working in a larger group that have a separate privacy team. He explained that *"The IDE automatically analyzes all the sources and sinks, so the privacy team can focus on the privacy work such as checking if privacy notices have been implemented for all of them."*

Feedback on the Generated Privacy Notices

When asked about their favorite design among the three privacy notices as a user, eight participants picked the privacy center (P1, P3, P5-P9, P12), three picked the permission notice (P4, P10, P11), and only one picked the just-in-time notification (P2). Participants preferred the privacy center the most because it contained more information about the app's data practices and it was

available to the user at any time. They also generally found the permission notice and the just-in-time notification useful, for they use a *"streamlined format"* (P2) and provide *"finer-grained detail than what you get with regular permissions"* (P3).

Some people also touched upon potential issues about these privacy notices, categorized into two themes:

Notification Overload Problem. P3 and P5 expressed their concerns about the notification overload problem specifically for the just-in-time notice design. P3 elaborated her concerns using the Starbucks app as an example,

> *"I use the Starbucks app, and I guarantee you it'll be blurting at you all the time some privacy notifications, but would that improve my experience? I'm not sure because I kind of know at the back of my head that they're probably doing that and I guess I'm okay with they know where I am"*.

Similarly, P5 said *"receiving many notifications from an app can be annoying from a user perspective"*. I have considered this issue when designing Honeysuckle and allow developers to control default visibility for each just-in-time notification to balance user experience and privacy. However, there are still issues such as developers can only make static decisions so the generated UI can not accommodate to different privacy concerns of different users. In Section 5.4.4, I discuss potential ways to further improve privacy using the annotations.

Factors That Impede Adoption. Some participants also explicitly discussed or implied factors that may prevent them from using Honeysuckle in real life. First, although many participants have been familiar with system permissions, in-app privacy notice was quite a foreign concept to them. For example, P4 was confused when first seeing the permission notice and asked *"Is this extra dialog useful?"* P3 mentioned that she had worked with many clients but had never encountered a requirement of implementing in-app privacy notices. Second, currently Honeysuckle has to rely on developer's input for some fields in the annotation such as purposes and can not automatically verify them, causing the risk of generating misleading privacy notices if the developer does not provide accurate information. P1 said *"You can not ensure that the developer will be an honest developer and he will tell you his true intentions of the data"*. However, I want to note that this is a general challenge facing any kind of privacy notices. I discuss potential mitigation methods in Section 5.4.4.

5.4 Discussion

I now discuss the benefits of using the Honeysuckle system for annotation-based privacy notice generation, the limitations of Honeysuckle, and other opportunities to improve data transparency and control using annotations.

5.4.1 Annotations as a Bridge to Connect Users and Developers

Previously, Coconut [95] demonstrated that annotations (and other metadata with similar attributes) can help educate developers and nudge them towards adopting best practices for privacy in their apps. In this work, I demonstrate that annotations are also effective in helping developers generate useful and usable privacy notices.

There are many reasons why annotation-based privacy notice generation can address common challenges regarding the implementation of in-app privacy notices. Since annotations are added next to sensitive API calls, developers do not need to explicitly specify any context, which saves on a lot of effort. There is a clear relationship between the annotation and the generated interfaces, which helps with code maintenance and collaboration for teamwork. My IDE plugin can automatically detect data sources and sinks and remind developers to add annotations, which reduces developers' cognitive load and increases the coverage and accuracy of the information declared in annotations. I carefully designed Honeysuckle to handle complicated scenarios in real-world apps, such as having multiple API calls serving the same purpose and having the same API call serving different purposes. More importantly, my approach allows developers to build effective privacy notices without having to be a privacy expert, since the generated interfaces automatically follow the best practices.

My quantitative lab study results demonstrated that the annotation-based approach consistently outperformed the library-based approach in terms of time on task, cognitive load and perceived usefulness and usability to developers. I also showed in my qualitative analysis that participants felt adding annotations was a better approach than using the library. They preferred using annotations because it improved productivity, reduced mental load, was easier to learn and use, made code easier to comprehend, and could provide benefits for teamwork. Overall, adding annotations was perceived as a simple task and only led to a small extra effort, because the annotation and UI configuration XML design allowed them to write much less code and the IDE plugin offered sufficient suggestions and quick-fixes to streamline the task. They felt the Honeysuckle library more complicated to use and required more work than annotations, but still considered it useful since no such tool exists either. Note that both approaches were new to developers who participated in my study. Only three out of the twelve participants had previously used annotations for any kind of code generation, though all participants picked up the annotation-based approach and finished the task much faster than the library-based approach.

In a broader sense, some of my design considerations may increase the likelihood for Honeysuckle to be adopted, based on lessons learned about why model-based user interface design (MBUID) [118, 157] did not achieve widespread adoption. Instead of solving the generic UI implementation issue, I focus on privacy notice generation specifically and try to minimize the implications on regular app development tasks. Furthermore, given the low adoption of in-app privacy notices, developers likely have not established preferences for building their own interfaces, one of the barriers that prevented the adoption of MBUID [157].

5.4.2 Increasing Developers' Incentives to Improve Privacy

Many privacy-related requirements currently in place for Android and Google Play impose many restrictions on what data developers can use [98]. Conversely, I argue that requiring developers to create more privacy notices to improve transparency will cause little to no impact on the apps' functionality, which makes developers potentially more amenable to them. Furthermore, by making it easier to create privacy notices, we may be able to nudge developers who care about users' privacy to adopt more effective designs.

As suggested in prior research [34, 106], providing more detailed privacy notices for legitimate data practices is likely to not only improve users' perceived comprehension and control of

their privacy, but also make users more comfortable with allowing the app to access their data and trust the app more. Nowadays, there seems to be a growing interest in both academia and industry to build applications that can provide benefits to users and preserve privacy at the same time, but the common technical approach is still to minimize data practices, such as using differential privacy, federated learning, and on-device training and inference. With the current permission system, there is still a gap between users' perceptions and the actual data practices [98], and my tool can potentially address this problem and make developers' effort in protecting user privacy more visible to users.

5.4.3 Limitations of Honeysuckle

I chose a relatively simple code analysis method for Honeysuckle implementation, which is to search for all API calls that match a predefined source and sink API list. Although the code analysis subsystem is largely adapted from Coconut [95] and is not my main contribution, I want to discuss limitations in the current implementation and point out potential mitigation methods. First, the detected sinks may not actually store or send *sensitive data* off the device. Although I allow developers to manually add annotations to mark non-sensitive sinks, this process can be tedious and error-prone, especially for teamwork. Given that researchers have built tools to statically analyze data flows of compiled Android apps [24], a potential solution is to investigate how to conduct in-IDE data flow analysis over the source code and give developers hints and warnings for more efficient and accurate annotation filling. Second, my sensitive API list does not guarantee a full coverage. To improve the coverage, we can integrate existing methods [26, 139] to automatically identify source and sink APIs for different versions of Android. For third-party libraries, Honeysuckle solely relies on a curated list of popular third-party libraries. Since prior work suggests the usage of third-party libraries in Android apps follows a long-tail distribution [42], a potential solution is to combine a predefined third-party API list to cover the majority of cases with on-demand analysis to handle edge cases. Furthermore, Honeysuckle does not support customizing the styles and themes of the generated UI. This feature is straightforward to implement by allowing developers to specify their own style and theme XMLs [6] to Honeysuckle.

5.4.4 Future Work: Going Further from Honeysuckle and Privacy Notice Generation

Despite the promising benefits, a developer tool for generating privacy notices alone can not solve all challenges facing the adoption of privacy notices and privacy by design in general. Therefore, I propose some follow-up work ideas to further improve privacy based on Honeysuckle and annotation-based privacy notice generation.

[6] https://developer.android.com/guide/topics/ui/look-and-feel/themes

Engaging Other Stakeholders to Enforce and Stimulate Accurate Privacy Notice

Honeysuckle relies on developers to provide accurate input to generate useful privacy notices. This could lead to the issue of "dishonest developers", which was also brought up by my participants (Section 5.3.6). Hence, I discuss potential methods to enforce or stimulate accurate privacy notices beyond building developer tools. First, the internal privacy auditing teams or the app store need to check if developers have provided correct information in their annotations. As I have alluded to earlier, to supplement the free-form purpose string, we can add another annotation field that requires developers to declare a purpose category, using a set of predefined options. Then app reviewers may be able to infer the purpose category to cross check the annotations. An alternative approach is to generating code to log, synthesize, and help users visualize what data is collected, retained, or sent off the device. The rationale here is to give users, privacy researchers, and consumer advocates the ability to cross check what data practices are declared by developers against actual data practices. Similar ideas can be found in both research and commercial systems. For example, the iOS background location history dialog renders the location traces collected by the app on a map along with a rationale string provided by the developer. Harbach et al. [72] demonstrated visualization design using personal examples to improve the communication of privacy risks.

Other Opportunities of Improving Transparency and Control Using Annotations

Currently, Honeysuckle works as a standalone tool, but I envision that privacy annotations, the Honeysuckle IDE plugin, and build system plugin could be more tightly integrated into the Android ecosystem. For example, annotations can be embedded into compiled apps, which would let the operating system generate unified interfaces for all apps and allow users to manage global settings about the appearances and frequency of notices, as well as when and where to see notices to avoid notification overload. As another example, having annotations embedded into compiled apps could make it easier to check the behavior of apps by app stores, researchers, and consumer advocates. By expanding the kinds of annotations developers have to add, it would be possible to generate a summary of the app's behaviors, such as what websites or cloud services it accesses and what data it sends. This kind of summary could augment the existing privacy interfaces, help users make decisions about whether to install an app (e.g. see PrivacyGrade.org), and help developers fulfill new requirements such as providing privacy labels for iOS apps.

Stepping back, I believe that a compelling vision for privacy is to have developers annotate their code just once, which can then help those developers with a number of privacy-related tasks (so that they will want to include accurate annotations), as well as make apps easier to audit by users and third parties. A challenge here is defining a unified and complete enough set of annotations that can cover all of these use cases.

Chapter 6

Matcha: An IDE Plugin for Creating Accurate Privacy Nutrition Labels

6.1 Overview

Privacy nutrition labels are short, uniform, machine-readable notices that allow users to learn about how their data is collected and used at a glance [56, 87, 89]. Inspired by this idea, the two major mobile app stores, Apple app store and Google Play store, introduced a privacy label section in 2020 and 2022 respectively.[1] This section is displayed on the public page of each app and presents data use information self-reported by the app developers. As of August 26, 2022, 60% of apps on the Apple app store and 44% of apps on the Google Play store have filled out forms to create these labels. However, researchers [27, 92, 102, 104, 181] and consumer advocates [62] have raised numerous concerns about these labels given their current design. One key concern revolves around the accuracy of the labels themselves, which could fundamentally undermine the entire effort if consumers lose confidence in the stated data practices.

Currently, developers alone are responsible for accurately creating the privacy label. However, recent research found that this seemingly straightforward task was very challenging for developers [102]. First, developers' understanding of their app's data practices may be incorrect or incomplete due to memory errors or unexpected data collection from third-party libraries [28, 30, 95, 119, 161]. Second, developers may not be capable of correctly translating their understanding to the privacy label because of misinterpretations of the terminology used to describe the data practices [102].

Using automated program or network analysis can provide insights into an app's data usage, but it is not a perfect replacement for developers in this process. The output of these analyses are not directly useful to end users and need to be converted to more high-level summaries of data practices. However, automated techniques for detecting data flows [24, 57] and categorizing them into concepts such as data types [77, 123] and purposes [81, 167] inevitably suffer from imperfect accuracy. Furthermore, after data leaves the device, it is infeasible to determine how data is further stored, shared, or repurposed without access to the backend server. In these situations, the developer's knowledge is needed to elucidate the detailed data usage and correct errors made

[1] "app privacy details" on the Apple app store and "data safety section" on the Google Play store

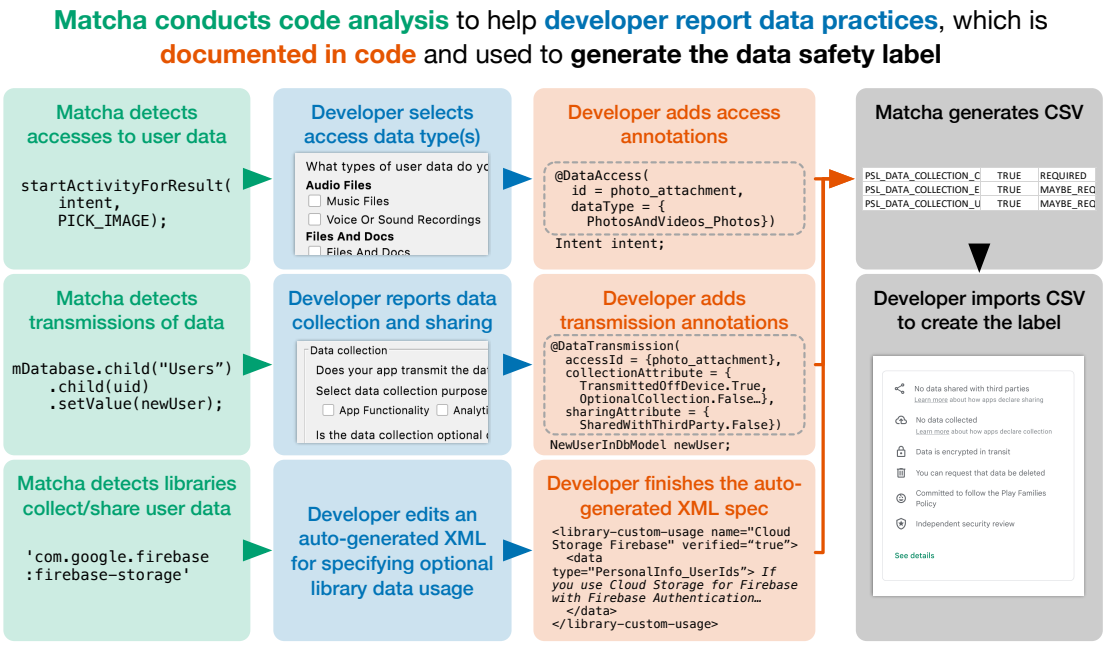


Figure 6.1: An overview of how Matcha helps developers create accurate privacy labels. Matcha conducts code analysis to provide suggestions about code that accesses user data and transmits data out of the app, as well as 3rd-party SDKs that collect and share user data. Then it guides developers to confirm, refine, or reject the suggestions by adding custom Java annotations and modifying an auto-generated XML file, which account for first-party and third-party data practices respectively. Finally, Matcha uses the annotations and XML to generate a CSV file that can be imported into the Google Play developer console to create the required label.

by the automated analysis.

This work explores a new design space for tools that can improve the accuracy of standardized privacy notices by leveraging the synergies between developers and automated program analysis. I present the design, implementation, and evaluation of Matcha, a plugin for the Android Studio IDE to help developers create accurate Google Play data safety labels. Figure 6.1 summarizes how Matcha works: First, Matcha analyzes the app's codebase and provides suggestions about first-party code that accesses or transmits user data and third-party SDKs that automatically collect or share user data. Then it asks developers to confirm or reject the suggestions by adding custom Java annotations and editing an auto-generated XML spec file for first-party and third-party data practices respectively. Finally, Matcha uses the annotations and XML to generate a CSV file that can be imported into the developer console to create the label.

The design of Matcha is inspired by my prior research on privacy-enhancing IDE plugins based on code annotations. Annotations have been used as a tool to increase the developer's awareness of privacy issues and reinforce best practices [95], facilitate the documentation of data practices [95], and streamline the implementation of in-app privacy UIs [100]. However, a fundamental problem remains unanswered: How to support developers to add annotations that *accurately* represent their app's data practices? With Matcha, I aim to investigate this problem in depth.

The design challenge lies in how to achieve a good balance between reducing developers' burden and soliciting accurate information from them. To achieve this goal, I first analyzed the Google Play data safety label to design the annotations and the XML spec that covers all the required information for generating the label. I then conducted pilot studies using an initial prototype and found that developers' overconfidence and incorrect mental model of what the plugin can or cannot do made them reject correct suggestions by the plugin. Furthermore, I noticed that my participants generally lacked basic knowledge about Java annotations, which became another barrier to providing accurate information. These findings informed my final design of the Matcha IDE plugin.

Creating privacy labels requires adequate knowledge about the app. Therefore, we evaluated Matcha with 12 developers working on their own apps. Matcha helped 11 out of the 12 participants improve the accuracy of their data safety labels as compared to using the Google Play developer console. Our analysis showed that Matcha was effective in addressing errors due to misunderstanding of the task of creating data safety labels, misunderstanding of the third-party libraries' data practices, forgetfulness, and misunderstanding of code behaviors. All developers favored Matcha to the developer console mainly due to the improved label accuracy, the user-friendly interface and learnability, and the educational benefits it provides in understanding their app's data practices. Drawing on my experiences, I discuss general design recommendations for developer tools for creating accurate standardized privacy notices. Matcha has been open-sourced and released on the plugin store [2].

My main contributions include:

- The design and implementation of Matcha, an IDE plugin for helping developers create accurate Google Play data safety labels.
- Evaluation studies with Android developers working on their own real-world apps ($N = 12$), demonstrating the efficacy and usability of Matcha.
- Design recommendations for developer tools for creating accurate standardized privacy notices.

6.1.1 Matcha Use Case

The example below demonstrates the typical workflow for using Matcha to create a Google Play data safety label. Carol is notified by the Google Play store that she needs to create a data safety label for her app. She tries the default approach, which is to answer questions about how her app collects and shares user data on the Google Play developer console. However, the task is overwhelming and she is not sure whether she has provided all the information accurately. Then she discovers Matcha, an IDE plugin designed for this task, and gives it a try.

In Matcha's "Privacy Label" guidance panel, she sees five tasks for creating the label (Figure 6.2-A) and clicks on the first task. She finds a short tutorial (Figure 6.2-C) and a list of subtasks (Figure 6.2-B), and learns that she needs to annotate data access API calls detected by Matcha. Her job is to verify whether each API call indeed accesses user data, and if so, specify what types of data it accesses. In the first subtask, she uses Matcha's quickfix to add the annotation (an IDE feature for repairing code issues) (Figure 6.3-D). A dialog pops up, guiding her to

[2]Matcha project website: https://matcha-ide.github.io

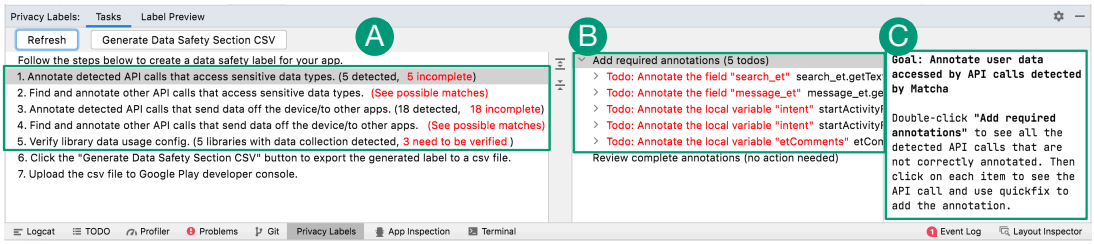


Figure 6.2: An overview of the required tasks in the "Privacy Labels" guidance panel. On the left, there is a summary of the five tasks for providing the required information (A). The subtasks for the currently selected step are displayed in the middle (B). And a short tutorial for the current step is available on the right (C).

choose the data type "In App Search History" from a list of predefined options (Figure 6.3-E). Matcha then automatically adds a `@DataAccess` annotation to the variable `search_et` (Figure 6.3-F). Carol defines a unique id `R.string.user_search_history` for this access so she can refer to it later. She completes the other subtasks in the same manner.

API-based detection may not be sufficient at times, such as when the developer uses an uncommon library. Therefore, in the second task, Matcha uses keyword search to help her identify accesses to data types not detected in the first task (Figure 6.4-G). She double clicks each data type to skim through all the detected keywords. Then she identifies a few more data types accessed by the app (Figure 6.4-H) and uses Matcha's quickfix to add the additional `@DataAccess` annotations.

Next, she performs the third task, which is to annotate API calls that may transmit data out of the app. These data transmissions may count as data collection and sharing, unless they does not send user data out of the app (e.g., a hard-coded request to fetch a web page) or fall under certain exemptions. To help discern whether any user data collection or sharing occurs, Matcha needs information from Carol that cannot be automatically inferred. Carol applies Matcha's quickfix and answers the questions in the pop-up dialog (Figure 6.5-I). The quickfix then adds a `@DataTransmission` annotation (Figure 6.5-J). After adding annotations for all the detected API calls in the third task, she reviews the keyword search results regarding other data transmissions in the fourth task, and adds additional annotations.

The final task relates to data automatically collected and shared by third-party SDKs, which are a main cause of data collection [42]. Some SDKs always collect or share user data, while others allow developers to configure the data use. Matcha can detect the most popular commercial SDKs on Google Play. For the data use configured outside of the app code, it generates an XML file to help the developer review all the potential data collection and sharing behaviors and the conditions under which they occur. When Carol reviews the XML, a tooltip shows up pointing to the condition under which the user name will be collected by the Firebase Authentication SDK (Figure 6.6-K). She considers it irrelevant to her use of the SDK and then removes the corresponding `<data>` tag. After verifying all the instances, she sets the attribute `verified` to `true` to indicate the completion status to Matcha (Figure 6.6-L).

After providing all the required information, Carol switches to the "Label Preview" view to check out the resulting label (Figure 6.7-M). She notices that her app both collects and shares

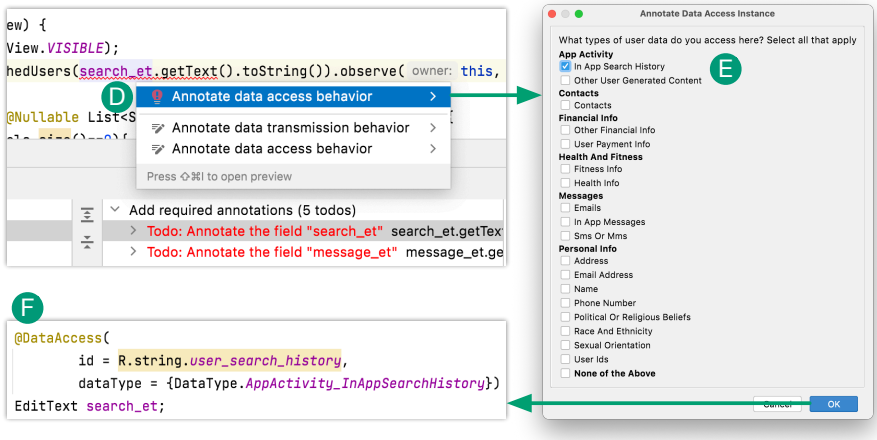


Figure 6.3: For each detected API call that may access user data, the developer can use the quickfix (D) to add the annotation. The quickfix displays a dialog asking the developer to select the data type(s) accessed by this API call from a list of predefined options customized for the API (E). Matcha then adds the annotation at the proper place and uses the developer's answer to fill in the `dataType` field value (F). The developer needs to declare a unique ID (`R.string.user_search_history`) for the access instance for future reference.

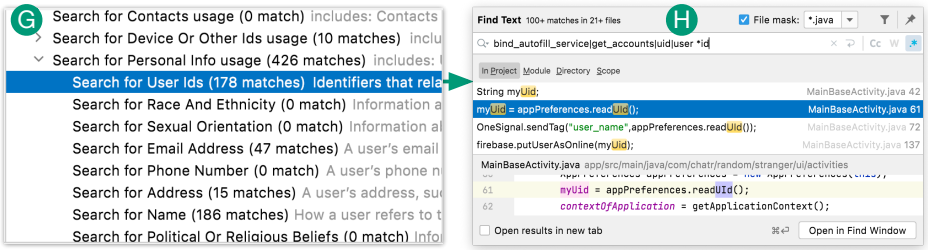


Figure 6.4: An example of finding more data access instances based on keyword search results to complement the API-based detection.

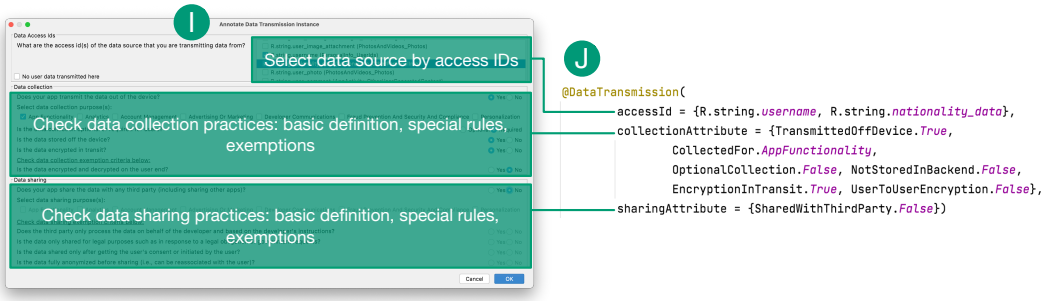


Figure 6.5: An example of the dialog for guiding the developer to annotate a data transmission instance (I) and the relationship between the answers provided through this dialog and the resulting `@DataTransmission` (J). The information solicited from the developer includes where the transmitted data is originally accessed, whether the data is collected or shared and if so, whether the collection and sharing practices are exempt from disclosure per Google's definitions.

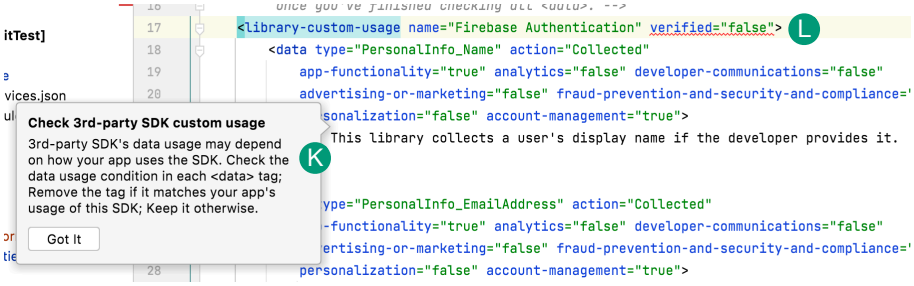


Figure 6.6: An example of editing the auto-generated XML specification of a third-party SDK's data practices to make it compatible with the app's usage of the SDK. Each `<data>` tag contains a condition (K) under which certain data type is collected or shared by the SDK. The developer can keep or delete the `<data>` tag based on how they use the SDK and finally set the `verified` attribute to true (L).

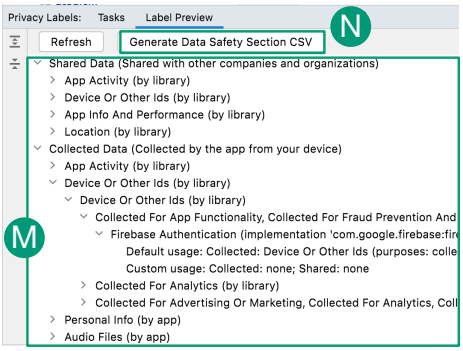


Figure 6.7: The developer can preview the data safety label using "Label Preview" (M) and generate the data safety section CSV by clicking the button (N). The layout is similar to the preview feature on Google Play developer console, while Matcha provides more features to help the developer understand the connection between the code and the label, such as whether the data collection or sharing is caused by the app, by libraries, or by both. The developer can expand each record and trace back to the corresponding line of code.

data, while the sharing is only caused by the third-party SDKs in the app. She then clicks the "Generate Data Safety Section CSV" button (Figure 6.7-N) to export the data safety label into a CSV, which she later uploads to the Google Play developer console to fulfill the requirement.

6.1.2 Background and Related work

Website privacy policies are notoriously long and difficult to read [116]. To tackle the problem, researchers proposed "privacy nutrition labels" more than a decade ago, to offer a clear, uniform, and succinct format for disclosing data usage. Many variants have been proposed for websites [87], mobile apps [89], and IoT devices [56]. Prior research has shown that standardized labels can help users find information about how their data is used faster [88], improve comprehension of privacy practices [88], and nudge consumers to make more privacy-conscious purchase choices [56, 89].

Apple introduced the App Privacy section to the Apple App Store in December 2020, marking the first large-scale deployment of privacy nutrition labels in real life. Google followed with the Data Safety section, their version of privacy nutrition labels, to the Google Play Store in May 2022. The introduction of privacy labels to the two major app stores has multiple potential benefits. First, users can directly gain a better understanding of an app's data use [88, 116]. Second, it gives developers a systematic and structured way to disclose their data practices to end users [102]. Third, the standardized and machine-readable format facilitates the research and deployment of novel formats of privacy notices to help users further synthesize, analyze, and compare app data practices [145].

However, researchers have identified numerous problems with these privacy labels. One issue is the prevalent inaccuracies in these labels. Balash et al. [27] found that many apps seemed likely to collect user data but did not declare any data collection in their labels. Li et al. [104] suggested that many Apple privacy labels may be outdated. Kollnig et al. [92] found that many apps used tracking libraries and sent data to known tracking domains but reported no data collected. Xiao

et al. [175] found apps whose data flows were inconsistent with their privacy labels. Other work focuses on usability issues. Zhang et al. [181] interviewed 24 lay users about the Apple privacy labels and uncovered problems with the usability, understandability, and effectiveness of these labels. Li et al. [102] studied the usability and understandability of Apple privacy labels from the developer's perspective and identified many barriers that prevent developers from creating accurate privacy labels. These findings suggest that the accuracy and usability problems are interdependent.

I present Matcha, an IDE plugin, to improve the accuracy of privacy labels by addressing the usability and comprehension challenges for developers [102]. Although I focus on Google data safety labels because they support importing labels generated by external tools, I consider the developer-in-the-loop, machine-facilitated idea generalizable to other types of privacy nutrition labels and standardized privacy notices. I derive recommendations for designing developer tools in the same vein from in-depth studies on Matcha and discuss them at the end of this chapter.

Other tools also exist for creating privacy nutrition labels for iOS or Android apps. For example, Privado.ai[3] is a commercial tool for creating a Google Play data safety label. Gardner et al. [63] reports on the preliminary feedback from developers on a tool for creating an Apple privacy label. However, I are the first to conduct in-depth studies to show that my tool (Matcha) can improve the accuracy of the privacy label created for real-world apps by the original app developers. As compared to these tools, Matcha streamlines the process for creating privacy labels by integrating the functionality into the IDE rather than using a separate website. This makes it easier for developers to check the actual app implementation when needed, following a code-centered approach. Developers found the annotations flexible to use and felt it would be even easier to add if they were doing it while implementing the app.

6.2 Matcha Design and Implementation

In this section, I present the design goals of Matcha, how my design fulfills these goals, the findings and improvements from my pilot developer studies, and my system implementation.

6.2.1 Design goals

I drew upon prior literature to inform my design goals. Li et al. [102] discovered that one obstacle to creating accurate Apple privacy labels was that developers needed to quickly process a large amount of new information, including lengthy and complicated definitions of concepts like data collection, linking, and tracking. The similarities between the Apple and Google label filling process suggest that Android developers may also suffer from information overload issues. I argue that developers will benefit from more scaffolding, which leads to my first design goal:

D1 The privacy label questions should be deconstructed and situated within the context in which developers handle the specific code that deals with user data.

Developers know how their apps work, but prior research suggests they suffer from issues such as forgetfulness [95], lack of knowledge about other team members' code [95] and third-

[3] https://www.privado.ai/data-safety-report

party SDK's data practices [30], and misinterpretations of terms in privacy labels [31, 102]. Hence, I set the second design goal:

D2 The tool should help developers overcome limitations in their ability to create accurate privacy labels.

Automated code analysis can be used to identify some user data practices that need to be reported in the data safety label. However, it still has limitations. First, it mainly analyzes data practices within the client app and cannot provide information about how data is used after it is sent from the device. Second, although algorithms exist to infer purposes of data use [81, 167] and data types that are not tied to a specific API [77, 123], they do not have perfect accuracy. This suggests that automated code analysis cannot be relied on completely, which leads to my last design goal.

D3 The tool should give developers control to refine or reject the automated analyses when they are not accurate.

To satisfy **D1**, I designed Matcha as an IDE plugin. I adopted the idea of using annotations to document data practices in code from prior work [95, 100], allowing developers to only answer the related privacy label questions for each data access and transmission point by adding annotations. I divide the design of Matcha into two parts. In the first part, I focus on determining what types of information to solicit from developers and in what format. In the second part, I focus on the interaction design to help developers provide accurate information about their data use.

6.2.2 Matcha Code Format Design

Data collection and sharing are the two core concepts in the Google Play data safety label. Currently, the developer needs to report data types that are collected and shared and supply additional details such as the purposes of collection and sharing, whether the data is processed ephemerally (not stored on the backend), and whether the collection is optional or required. Moreover, they need to check whether the data collection and sharing practices fall under any of the exemption rules so it does not need to be disclosed on the label. Prior research has suggested that developers do not have the time and ability to comprehend and apply these complex concepts [102]. To address this issue, I designed the specific code format of developer input as a scaffolding for providing all the required information accurately.

Privacy annotations for explicit data flows within apps

To avoid misunderstandings of *data collection* and *sharing* per Google's definitions, I design the Matcha privacy annotations based on two more specific concepts: *data access* and *transmission*. Data access refers to code that reads user data into the app's internal memory, and data transmission refers to code that sends the data off the device or to other apps on device. The relationship between the two groups of concepts is illustrated in Figure 6.8. The annotation design is summarized in Table 6.1. The `@DataAccess` and `@DataTransmission` annotations indicate where the app accesses and transmits user data, respectively. Since data collection and sharing only happen with data transmission, I design the `collectionAttribute` and `sharingAttribute` fields in `@DataTransmission` to answer the additional collection

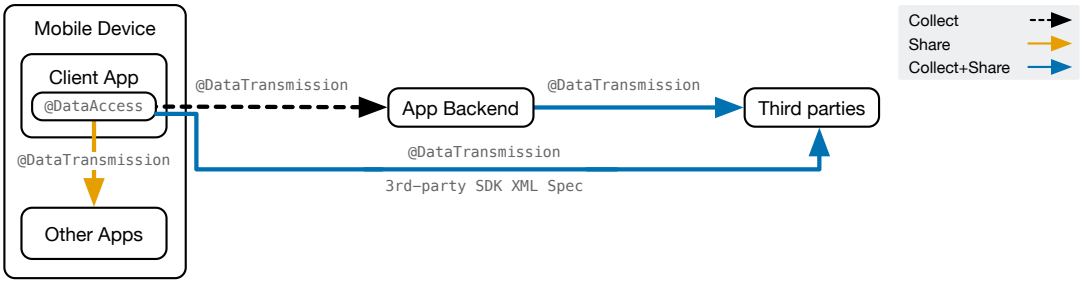


Figure 6.8: An illustration of the mapping between the developer input (i.e., annotations and the SDK XML spec) and Google's definitions of data collection and sharing. For example, the developer needs to add a `@DataAccess` annotation when the data is accessed within the app, and then add a `@DataTransmission` annotation about how it is collected at the app backend and further shared with third parties. By asking developers to add data access and transmission annotations rather than directly deal with the label-specific concepts, Matcha reduces errors due to misunderstanding of the label terms while keeping developers in control of the label.

and sharing questions from the developer console and check the exemptions for collection and sharing. Developers can also use `@NotPersonalDataAccess` and `@NotPersonalDataTransmission` to explicit indicate no data is accessed or transmitted.

XML spec for implicit data flows due to third-party SDKs

A third-party library's data collection and sharing may depend on how the app uses it. For example, in the motivating example the Firebase Authentication library only collects the user's display name if the developer provides it (Figure 6.6). Therefore, I design an XML file to allow developers to customize the third-party data disclosure based on their use of the library. The XML addresses two limitations of only using annotations. First, since much of the configuration of library data use happens outside of the app, such as in the web console, it is hard to determine a fixed location for the annotation to check if the required annotation is added. In the XML, the developer can set the `verified` attribute to true to indicate that they have finished checking a specific library. Second, the data collection and sharing conditions cannot fit into the annotations. In the XML, Matcha generates a `<library-custom-usage>` tag for each library and populate it with multiple `<data>` tags that contain the collection and sharing conditions. Developers can choose to either keep or remove the data tag based on the condition.

6.2.3 Pilot Test of Matcha

To achieve **D2**, I offer suggestions for data access and transmission and quickfixes for adding annotations based on automated code analysis. To achieve **D3**, I let developers have the final say, namely, they can ignore any suggestions, and the label creation only relies on the annotations and the XML spec that they can modify. This raises a crucial question for my interaction design: *Are developers capable of correctly comprehending the suggestions and building on them to create an accurate data safety label?* I tackled this question by conducting IRB-approved pilot studies with five developers on their own apps using an initial prototype that provides basic support for

Table 6.1: The table shows the four annotations I designed for the task and their field members that hold different types of information needed for the label. I provide more details about how `@DataTransmission` handles collection and sharing in Section 6.6.

Annotation	Fields	Note
`@DataAccess`	id	A unique ID defined by the developer to refer to this access when later annotating data transmissions.
	dataType	A list of Enum values of predefined data types accessed by the app and held in the annotated variable
`@NotPersonalDataAccess`	–	For explicitly indicating no personal data is accessed here. It is useful for dismissing an irrelevant data access suggestion.
`@DataTransmission`	accessId	A list of IDs indicating where the transmitted data is originally accessed. The IDs are previously defined in `@DataAccess`.
	collectionAttribute	A list of Enum values of collection-related information.
	sharingAttribute	A list of Enum values of sharing-related information.
`@NotPersonalDataTransmission`	–	For explicitly indicating no personal data is transmitted out here. It is useful for dismissing an irrelevant data transmission suggestion.

adding annotations and editing the XML spec. The participants first created the label for their apps using the developer console and then using Matcha. I asked them to think aloud during the process. Below, I summarize issues that emerged from the studies and how they informed the improvement of my system design and implementation. The final design is introduced in Section 6.2.4.

Ignoring unexpected suggestions:

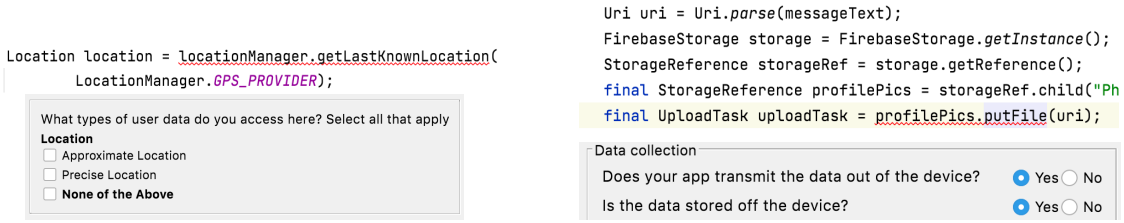
Some developers appeared to be affected by confirmation bias, namely they ignored suggestions about data practices that did not match their expectations. For example, P3's app had a feature for sharing with other people the user's high score, the game screenshot, and a short message provided by the user. Although the data is not very sensitive, it still falls under the "App activity" data category per Google's definition. However, P3 quickly added the `@NotPersonalDataAccess` annotation to dismiss the Matcha's suggestion. He explained that "*I don't think high score is personal info...because I only have one data access and one data transmission.*" This problem suggests that developers tended to place more trust in their own understanding than in the system's advice when they are in conflict. This created significant obstacles to using automated code analysis to correct the developer's misunderstandings. To address this problem, I consulted the guidelines for human-AI interaction [21] because building trust is the main goal of constructing efficient AI-infused systems. Specifically, I designed more proactive and contextualized guidance to help developers establish a clearer mental model of what Matcha can do and how Matcha's suggestions work.

Difficulty of handling Java annotations:

My participants faced challenges in adding the annotations due to unfamiliarity with Java annotation syntax. For example, Java annotations can only be added to a few specific code elements, such as a variable declaration. This troubled developers when they needed to manually determine where to add the annotation. Another example is when developers declare multiple variables on the same line separated by a comma (e.g., `EditText nameText, ageText;`). If the developer wanted to apply different annotations to each variable, they needed to first separate the variables into different lines and then add annotations. During the pilot tests, I found that these seemingly trivial issues with annotations greatly hindered my participants' abilities to accomplish this task. Therefore, I further optimized the support for adding annotations in the final version of Matcha to help developers automatically trace where to add the annotations and reformat their code.

Error-prone direct editing of annotations:

Some errors were because developers had to manually edit annotations to complete all the fields. First, since my participants were first-time users of Matcha and unfamiliar with the Google data safety label design, they felt overwhelmed by the number of fields they needed to manually complete and the number of predefined values they can select from. Second, my initial version only supported adding an annotation with one data type at a time; however, during my pilot



(a) Data type options customized based on the API call.

(b) Transmission and storage pre-checked based on the API call.

Figure 6.9: Examples of quickfix dialogs customized based on the API calls to avoid errors and improve learnability and usability.

studies I observed that it was common for some data accesses to fall under multiple data types, such as a user's name also used as a user ID. Although direct editing may be more efficient for expert users, it was too error-prone and confusing for novices. Therefore, I created a dialog for guiding the developer to fill out all the required information to generate the annotation.

6.2.4 Final design of the IDE plugin

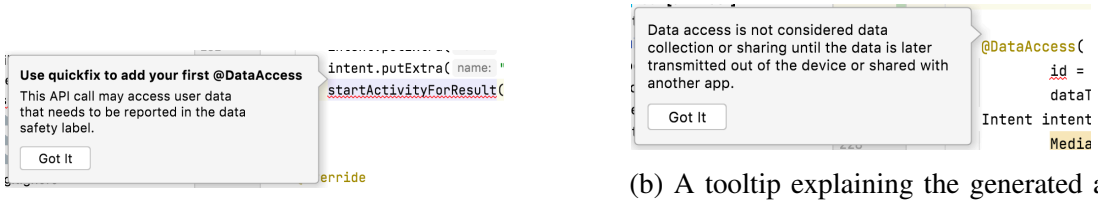
The final design of the Matcha IDE plugin is informed by the findings from the pilot tests. I present an overview of the five tasks for creating the label and the main supporting features for these tasks.

A five-step process

The Matcha label creation process consists of five steps. The first two steps are for adding data access annotations and the next two steps are for adding data transmission annotations, so the transmission annotations can refer to the access IDs defined in the access annotations. In the first and third steps, Matcha guides developers to do a precise API-based search, by which they can see all the detected API calls without the required annotation. The detected API calls without a proper annotation will also be flagged as an error within the code editor with a red squiggly underline. Developers need to check each API call to add the annotation. In the second and the fourth steps, Matcha uses fuzzy keyword-based search to present a list of data types that are not covered in existing data access annotations. Then the developer can double click each data type to show the keyword search results in a pop-up dialog. Adding annotation is a voluntary task at the second and the fourth steps, which means the developer only needs to add an annotation when the detected keywords relate to accessed or transmitted user data. In the last step, the developer modifies the auto-generated XML spec file to adapt it based on their usage of the library.

Quickfixes for adding annotations

To address the issue with direct annotation editing due to unfamilarity with Google data safety label and Java annotations, Matcha offers quickfixes with a dialog to guide the annotation creation (see Figure 6.3 and Figure 6.5). For detected API calls, the quickfix can automatically locate



(a) A tooltip suggesting next action.

(b) A tooltip explaining the generated annotation.

Figure 6.10: Examples of the contextualized, proactive tooltips to give developers just-in-time support and education about the annotations.

which variable to annotate. The developers can also click on any variable and use the quickfix to add an annotation. The quickfix for access API calls can narrow down the list of data type choices. For example, if the detected API is `LocationManager.getLastKnownLocation`, the available choices are only approximate location, precise location, and none of the above (Figure 6.9a). The quickfix for transmission API calls can auto-fill some transmission attributes. For example, if the detected API is from the Firebase Cloud Storage for uploading a picture to a cloud-based database, the "data being transmitted off the device" and "data being stored" options are automatically checked (Figure 6.9b). I note that expert users can still modify the generated annotations in any way they want while using the dialog to constrain and validate the developer's input could both reduce the chance of novice users accidentally making errors and make the tool easier to learn and use [128].

Contextualized and proactive guidance

To help developers understand what Matcha can do and how Matcha suggestions work, I designed just-in-time tooltips that automatically pop up next to the related code when the developer first encounters certain types of Matcha suggestions and give instructions on what action to take (Figure 6.10a). Matcha also provides tooltips that are only informational (no action needed), such as explaining what the auto-generated annotation has to do with the creation of the data safety label (Figure 6.10b). In addition to the just-in-time tooltips, Matcha also offers a more systematic introduction of each step in a help panel (see Figure 6.2).

Label preview

To help developers better understand how the data safety label is generated, I design a label preview panel (see Figure 6.7). For each data type that is collected or shared, a note of "by library," "by app," or "by app and library" is provided, indicating the source of this data collection or sharing. After expanding each data type, it will show further information like the data collection or sharing purposes, as well as the related code links. These links allow the developer to check which annotations, custom library usage records in the XML file, or third-party libraries that always collect user data led to the generation of this particular data safety label entry.

6.2.5 Matcha System Implementation

The Matcha IDE plugin was developed using the IntelliJ Platform SDK[4]. My implementation comprises 14,986 lines of Java code. My API-based detection was built upon the code analysis subsystem of Coconut [95]. For data access, I augmented the Coconut API list with APIs from the official guidelines for this task[5]. For transmission, I kept the APIs about network requests from Coconut and added on-device sharing API calls for the sharing-only condition. My final API list contains 91 data access APIs and 45 data transmission APIs.

I implemented the keyword search using the IntelliJ SDK's `findManager.findInProject` API. My keyword list comes from three sources: (1) keywords of permissions mapped with specific data types in Google's official guideline[6], (2) keywords manually extracted from the official definitions of the required data types[7], and (3) keywords extracted from open-sourced Android projects using a tf-idf algorithm. My final keyword list contains 180 unique keywords.

Matcha scans the app's build.gradle file to detect third-party libraries and automatically fills in their required data collection and sharing practices in the generated label. It also uses the conditions of optional data collection and sharing from the SDK's official guidelines for this task to create the `<data>` tags in the XML spec. I consulted the Google Play SDK Index[8] to find official guidelines of popular commercial SDKs. I also found additional third-party SDKs developed by Google that provided these guidelines. My final SDK list contains 58 unique third-party SDKs.

6.3 Matcha Evaluation

I conducted remote studies with 12 developers on their own apps in August, September, and November, 2022. I observed each participant creating the data safety label first using the Google Play developer console and then using Matcha. Each interview took 1.5 to 2 hours. The study was approved by my institution's Institutional Review Board.

6.3.1 Study Design Considerations

It is challenging to evaluate the efficacy of interventions for improving the accuracy of a privacy label. Unlike many developer tools that can be evaluated with uniform tasks in well-controlled settings [54, 95, 100, 165, 166], a tool for creating the privacy label must be evaluated by developers who have adequate knowledge about the app. Otherwise, it is possible that the errors are simply due to unfamiliarity. Two potential methods exist for achieving this goal. The first method is to ask participants to develop an app with specific data practices and then create the label. This method ensures that participants have a sufficient understanding of the app and the data practices can be fully controlled. However, even developing a small Android app can cost

[4] https://plugins.jetbrains.com/docs/intellij/welcome.html
[5] https://developer.android.com/guide/topics/data/collect-share
[6] https://developer.android.com/guide/topics/data/collect-share
[7] https://support.google.com/googleplay/android-developer/answer/10787469?hl=en#data_types
[8] https://developer.android.com/distribute/sdk-index

thousands to tens of thousands of dollars, making it too costly for a user study. Hence, I used the second method, which is to ask participants to work on a real-world app they already developed.

However, asking participants to work on their own apps leads to new issues. First, I cannot obtain the ground truth of the data practices of these apps to verify the developer's answers. Second, since everyone is working on a different app, I cannot make direct comparisons between their performances. Third, it is hard to recruit many participants who already have an Android app and are willing to install a plugin to analyze their code and allow researchers to look at their code. Consequently, the common between-subjects study design, which would randomly assign participants to the baseline or experimental condition and compare results under different conditions, is not possible for us.

Given these challenges, I decided to use a mixed-methods study design to answer this research question: *How effective is Matcha for correcting the errors in the privacy label of the same app created by Google's official tool?* I conducted remote interviews to observe how developers created the label using the Google Play developer console and then using Matcha. I asked the developer to verify the changes caused by Matcha to compensate for the unavailability of the absolute ground truth. My less-controlled, more realistic study setting placed higher requirement on the robustness of my tool, so I spent additional time testing and improving my prototype to make it work with arbitrary apps [68].

6.3.2 Participants

12 developers participated in my study. I coded the data iteratively and stopped recruiting until reaching a theoretical saturation in my qualitative analysis [75]. This sample size is the consistent with the evaluation studies of novel programming tools in past research [54, 100, 110, 182]. Most participants were recruited from freelancer websites, including eight from Freelancer[9] and one from Upwork.[10] The other three participants signed up by filling out a pre-screening survey after seeing my recruitment messages distributed on Twitter, Slack groups, or from personal connections. The pre-screening survey asked them to provide an Android app that they recently developed, their role(s) in the development process and included questions to test their Android development knowledge.

Once the candidate participants passed the screening and decided on an app to use for the study, I asked them to fill out the pre-study survey and schedule the interview. In the pre-study survey, I asked my participants to confirm that they were comfortable installing the plugin to analyze their code and sharing their screen to show part of the code to the researcher during the interview. Each participant was compensated $70 for participating in the interview.

My participants were from eight different countries and the apps were developed either as part of their job, a hobby, or for a course project. Nine out of the 12 participants had experience in publishing apps on the Google Play store, though some of them chose not to use Google Play apps for my study due to NDA restrictions. My sample included six Google Play apps, with the most popular one having more than one million downloads. I append details about each participant in Section 6.7.

[9] https://www.freelancer.com
[10] https://www.upwork.com

6.3.3 Study Procedure

At the beginning of the study, I briefed the participant on the study goals and obtained their consent for audio and screen recording. Before the main tasks, I first provided the participants with a brief introduction to the Google Play data safety label. Then I asked the participants to introduce the app they had selected.

In the first task, the participant created the label for the selected app using the developer console. Participants logged into the developer console using an account I created for the study. I asked them to handle this task as they normally would and encouraged them to use any resources they would normally consult, except for the app's current label if available. In the second task, the participant created the label using Matcha. I first asked them to download the plugin and helped them set up the environment. I created a short tutorial video (2 minutes 42 seconds) for Matcha and asked them to watch it before working on the task. If they were not sure about how to fill in certain information, I asked them to answer based on their best understanding and explain their rationale. After creating the label, I asked them to import the CSV into the Google Play developer console to create the label. Participants were asked to think aloud during both tasks. I discuss the potential implications of a learning effect due to the two-task within-subjects study design along with other methodological limitations in Section 6.3.5.

I conducted a brief semi-structured interview after the main tasks. I asked the developer to examine the discrepancies between the two labels and tell us which version was more accurate and what caused the error. Then I asked them to tell us which tool they preferred and why, and asked how easy or hard it was to use Matcha, whether anything was surprising or confusing in Matcha, and whether they learned anything about their app from the study.

6.3.4 Qualitative Analysis Method

I qualitatively coded the interview transcripts along with the screen recordings. First, I coded important user actions, such as when and how they added annotations and modified the XML spec entries. Then I coded the developer's verbal responses using a bottom-up open coding method [143] to synthesize interesting points that emerged during the think-aloud process of the main tasks and the post-study interviews. I met with other team members weekly to discuss the findings and derive themes from the findings. The coding process was facilitated by the software MAXQDA. I provide my complete codebook in Section 6.9.

6.3.5 Methodological Limitations

My study method has some limitations that require caution for interpreting certain results. First, using Matcha after using the developer console may result in a learning effect: the increased familiarity with the task might have contributed to the improvement in accuracy caused by Matcha. However, the process of using the two tools and the questions the developers answer are both very different, which suggests the learning effect may be small. My qualitative analysis of the error causes provides further insights into which improvements were caused by Matcha. Second, the identified errors are not exhaustive since the ground truth is not available. As such, the errors analyzed in this chapter should only be interpreted as the errors that Matcha can help detect and fix,

rather than all possible errors. However, the code analysis performed by Matcha should result in more accurate results than previous research that has only relied on developers' reflections [102]. Third, my study design may lead to a sampling bias, as all developers developed the app individually or in a small team. Consequently, my findings may not apply to developers who work in a relatively large company. Lastly, my study design is focused on studying the use of Matcha on a completed app, while Matcha can also be used during the app development process. I defer the latter use case to future research.

6.4 Results

Overall, I found that Matcha helped developers correctly report a lot more data collection and sharing practices. Both objective measurements and subjective feedback suggest that Matcha was easy to learn and use. All participants preferred Matcha over Google's tool because it improved label accuracy, contextualized questions in the code, and helped them learn more about data safety labels and the app's data practices.

6.4.1 Matcha Improved Label Accuracy

When reviewing the two labels created in the study, my participants considered all Matcha labels correct, except for F2, who correctly classified gender as "other personal information" in the console while then misclassifying it as "sexual orientation" using Matcha. Since misclassification errors do not affect the number of data types and purposes, I do not consider it separately in my quantitative analysis. All participants except for F6 used Matcha to fix some errors in their labels. The proportion of participants who have fixed errors in their labels using Matcha was 11/12 (approximately 92%) with a Wilson confidence interval [172] of (64.6%, 98.5%) at the 95% confidence level.

Matcha helped report 1.8 times as many data types collected or shared by the app (92 vs. 52) and 3.0 times as many purposes for data collection and sharing (212 vs. 70) as compared to the baseline Google developer console. I append further per-participant label information in Section 6.8.

6.4.2 Types of Errors Fixed by Matcha

My first analysis examines the errors corrected by Matcha in terms of the error type (e.g., under-reporting vs. over-reporting), the error source (e.g., first-party vs. third-party), and the related data practices. Table 6.2 presents an overview of the errors and I provide more detailed per-participant breakdown in Section 6.8.

Under-reporting vs. over-reporting

The great majority of errors corrected by Matcha are under-reporting errors (77%). In other words, more data practices were missed in the baseline label and were added by Matcha than the other way around. This demonstrates that using Matcha can help developers report more

Table 6.2: Analysis of errors fixed by Matcha. The baseline column refers to the number of data types and purposes reported using the developer console, and the added and removed column refer to what Matcha helped add or remove as compared to the baseline version. Most fixed errors are related to under-reporting issues (much more data types and purposes added than removed) and caused by third-party libraries.

	Data Type			**Data Purpose**		
	Baseline	Added	Removed	Baseline	Added	Removed
1st-party collect	21	8	13	34	15	20
3rd-party collect	18	44	11	22	107	15
1st-party share	1	2	1	2	2	2
3rd-party share	12	20	9	12	64	9
Total	52	74	34	70	188	46

comprehensive data practices. Note that Matcha also helped report more collected and shared data types for the three Google Play apps than their real-world labels. There were also a number of over-reporting errors corrected (23%).

First-party vs. third-party

A significant fraction of errors corrected by Matcha were caused by third-party libraries that automatically collect or share data (78%) than by first-party code (22%). This was mostly due to the use of Firebase services for basic functionality and for analytics, as well as other advertising and utility libraries.

Fixed errors related to different data practices

More errors fixed by Matcha were related to data collection (70%) than data sharing (30%). However, I want to note that the improvement for data sharing might be more essential, because no data sharing was reported in baseline labels, whereas some data collection was already reported in baseline. This suggests developers had more severe awareness gaps regarding data sharing. Furthermore, sharing data with third parties is more sensitive [42], which means the Matcha labels can better inform users of the privacy risks.

6.4.3 Matcha Helped Tackle Common Challenges for Creating Accurate Privacy Labels

I identified four themes in participants' explanations of errors fixed by Matcha. I found that Matcha helped address common issues that can lead to misunderstanding of data practices and inaccurate privacy nutrition labels. [30, 95, 102].

Help tackle data safety label knowledge gaps (F2, F4, F5, F8, F9, F10, F11, F12)

Matcha was able to help fix errors due to misunderstandings of the data safety label taxonomy. One recurring issue was developers failed to pay attention to unfamiliar data types. For example, F2 said "*I did not even think of 'other user-generated content.*"' Matcha helped them focus on a specific API call and a subset of data types related to the API, which helped correct these errors. Matcha also helped correct errors related to misunderstandings about data collection and sharing. For example, F8 initially thought he should report the list of installed apps as collected while the data was only used on device and therefore did not count as collection per Google's definition. In Matcha, the quickfix dialog explicitly asked whether the data is transmitted out of the device rather than whether the data is collected, which resolved the problem without the developer having to read and comprehend the lengthy definitions of data collection himself.

Help reduce errors related to third-party libraries (F1, F2, F3, F5, F8, F9, F11)

I found that Matcha helped correct errors due to a range of misperceptions about third-party libraries. Some developers did not know they should consider third-party libraries when creating the label. For example, F3 considered the data collected by Firebase was "*collected by a different platform*" but not part of his app. Some developers were unaware of data collected and shared by libraries. For example, F2 explained for the under-reporting errors due to Firebase cloud storage that "*I didn't know that the library was doing that on its own behind the scenes.*"

Interestingly, Matcha has also helped developers who already had some expectation of the third-party data practices. Most of them solely relied on their understanding of the libraries' data practices to answer the questions, which resulted in inaccurate results. For example, F1 said "*our data is sent to Firebase server, that's why I am selecting these*" as he thought out aloud during the first task. However, later Matcha revealed that Firebase collected more data then he expected. Only F11 searched the exact data practices of an advertising library used in his app. However, he was not aware that there was a specific guide provided for this task and chose to refer to the privacy policy. The ambiguous wording of the privacy policy then caused errors in the first label that were later corrected by Matcha. Matcha also fixed errors related to other Firebase libraries that he did not expect to collect user data.

Help reduce errors due to forgetfulness (F1, F2, F3, F6, F7, F10, F12)

One common source of errors that Matcha helped fix is due to forgetfulness. They could simply be an oversight – "*it just escaped my mind*" (F3), or have deeper reasons. For example, F6 forgot the use of a third-party library and explained that "*I didn't actually recall that because I was not in charge of this part.*" F7 forgot he integrated the Admob library a long time ago. Both errors were caught by Matcha. In line with findings from earlier research [102], I found developers mostly answered questions from memory when using the developer console. Matcha's systematic review of data practices helped developers to find and disclose more data types than they would have otherwise. For example, although F2 checked the Firebase database in the first task, he forgot certain tables and therefore missed certain collected data types, which he later caught with the help of Matcha.

Help tackle technical knowledge gaps (F1, F2)

Matcha could prompt the developer to research unfamiliar technical concepts when adding the annotations. For example, F2 was unsure about the precisions of the location data. Matcha helped him figure it out by promping him to add annotations for the `LocationManager.getLastKnownLocatio` API, which in turn led him to search for this API online. Matcha also helped correct misundertandings about permissions. For example, F1 thought the Admob library could not obtain approximate location data because the app did not request location permissions. However, he later learned from Matcha that the Admob library used the user's IP address to derive the approximate location, which was not controlled by the permission system.

6.4.4 Perceived Benefits and Challenges of Using Matcha

When asked to compare the two tools, all participants preferred Matcha over the developer console. I analyzed their answers and identified four themes of benefits offered by Matcha. I also analyzed the questions developers brought up during the think-aloud process and the post-study interview to gain insights into the challenges of using Matcha.

Benefit of Matcha: Improved accuracy (F1, F2, F4, F7, F8, F10, F11)

The primary benefit of using Matcha was the improved accuracy, which could outweigh the time cost. As put by F2, "*Accuracy wise, I would prefer the tool Matcha...For an app that I'm going to publish on Google Play, I would use Matcha, because like, I emphasize accuracy over efficiency.*"

Benefit of Matcha: Ease of use (F2, F5, F6, F7, F9, F11, F12)

Many participants considered Matcha easy to learn and use. F6 felt the quickfixes for adding annotations were "*pretty convenient*". F7 said "*I thought it might be complicated. But when I started a bit, it becomes easier to use*". F2 considered it would be easier if he annotated his code as he developed the app. It was observed that those who devoted more efforts to providing accurate information in the initial task via the developer console tended to find more value in the ease of use of Matcha. For example, F11 was the only person who did an in-depth search into the data practices of his app due to third-party SDKs in the first task (as mentioned in Section 6.4.3). He expressed a preference for Matcha because "*it is very easy, it saves a lot of time, and plus it is more accurate as compared to the Google Play console, which is very lengthy, and I have to read through all the options and then check the boxes, and I have to consult the documentation.*"

Benefit of Matcha: Informative tool (F2, F6, F5, F7, F8, F9)

Many participants liked Matcha because it was informative and helped them learn a lot about their app and the data safety label. For example, F8 felt that using Matcha helped him gain a better understanding of his app: "*Before using your plugin, I was quite sure that I have submitted all the information that I'm getting, but after using the plugin, I am more knowledge about what's going on in my app.*"

Benefit of Matcha: Better engagement (F2, F6, F10)

Some participants liked that Matcha contextualized all the questions around specific code, which better engaged them with this task than the developer console. F6 explained that:

> *The developer console does have everything written on it, but it's hard to actually relate that to your own code, because it's just a bunch of instructions. While the plugin could remind you of what you have written. For my example, I don't actually remember if I ever imported the WeChat SDK. I really don't remember that. And the console wouldn't actually remind me of anything.*

Benefit of Matcha: Better flexibility (F3, F6, F12)

Some participants considered using code to specify data practices gave them more flexibility. F3 mentioned that he preferred the plugin because "*It gave me the flexibility I needed and made me feel like I was still doing development work. All I had to do was add annotations and it generated labels for me.*" F12 even thought of the benefits of Matcha's annotation-based method for future app updates: "*If I update the app to another version, it's easy to change the annotation and create a new CSV file.*"

Challenge of using Matcha: Redundancy of suggestions and developer input (F2, F6, F5, F8)

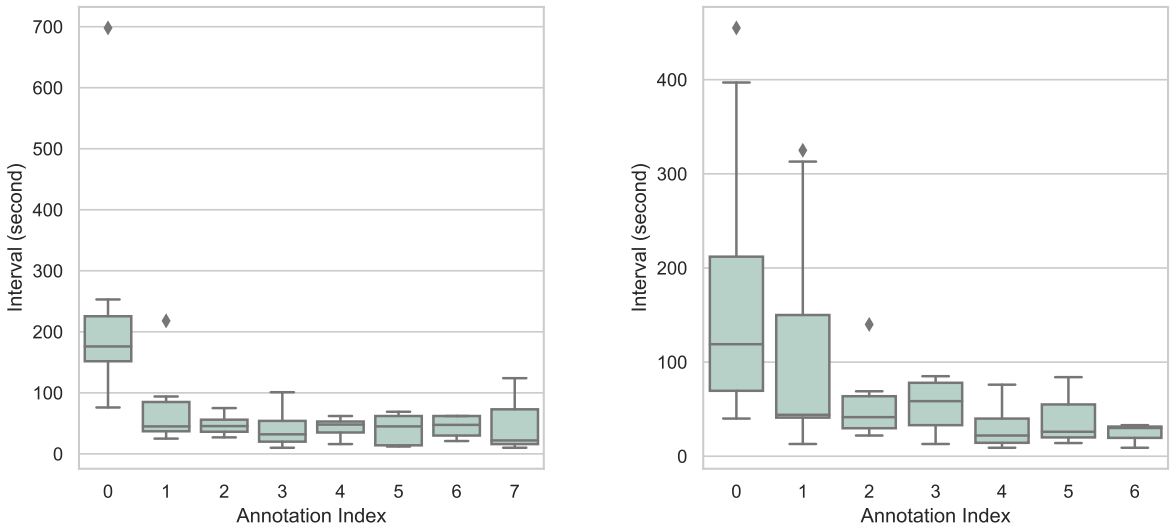
A recurring issue raised by developers was regarding the redundancy in Matcha suggestions and developer input. One source of redundancy was in the keyword search results. Some keywords were considered too generic and resulted in many false positives. For example, F6 felt that the term "search" for the target data type "search history" was not effective because it was often used in code for unrelated purposes. Another source of redundancy was in the required input from the developer. Since the API calls that collect the same type of data could be used for different purposes, I asked the developers to always separate them with different annotations. However, F2 complained about the need to annotate `requestLocationUpdates` after annotating `getLastKnownLocation`, which both get the same type of data and for the same purpose in his use case. Future work may consider improving the design with more context-aware suggestions.

6.4.5 Efficiency of Matcha

My analysis of developers' action traces offers insights into the efficiency and learning curve of Matcha.

Overall time performance comparison

It took my participants 30 minutes on average to complete the task using Matcha ($std = 15$ minutes), while it only took 9.8 minutes using the developer console ($std = 9.3$ minutes). Although developers were able to complete the task faster using the developer console, they did so at the cost of accuracy.



(a) Time to add an access annotation

(b) Time to add a transmission annotation

Figure 6.11: Distributions of time to add an access annotation (i.e., `@DataAccess` or `@NotPersonalDataAccess`) and a transmission annotation (i.e., `@DataTransmission` or `@NotPersonalDataTransmission`.), ordered by the indices of annotations for each participant. For the reliability of the results, I only include the annotation indices with at least three people's data. The two figures show that it took participants longer to add the first access annotation, while the time drastically decreased at the second annotation and became stable afterward, suggesting an easy learning curve.

Time for adding annotations

I further analyzed the time for adding privacy annotations, which is a novel task I introduced in Matcha. I demonstrate that adding each annotation only requires a little amount of time. Each access annotation (`@DataAccess` or `@NotPersonalDataAccess`) took 1.3 minutes on average to add ($std = 1.6$ minutes). Each transmission annotation (`@DataTransmission` or `@NotPersonalDataTransmission`) took 1.3 minutes on average ($std = 1.6$ minutes) to add. I further show in Figure 6.11 that the time of adding both types of annotations decreases over time, suggesting my participants' performance increased with practice.

6.4.6 Developers' Reactions to Matcha Suggestions

Finally, I analyze developers' reactions to the detected API calls and libraries to demonstrate the importance of keeping developers in the loop to refine the ambiguity and inaccuracy in the Matcha suggestions. Matcha detected 10 access API calls and 6.3 transmission API calls on average per app, which led to 4.9 access annotations and 5.2 transmission annotations added by participants on average per app. Note that the number of required annotations are fewer than the detected API calls because multiple API calls can share one annotation. Among the 59 ac-

cess annotations, 40 were `@DataAccess` annotations; among the 62 transmission annotates, 28 were `@DataTransmission` annotations. Others were considered as irrelevant and resulted in a `@NotPersonalDataAccess` and `@NotPersonalDataTransmission` to dismiss the suggestion. For example, the user password and files provided by the app rather than the user were the two types of data most commonly labeled as `@NotPersonalDataAccess` in my study; and data stored locally on device, network request without user data, and data transmission practices that meet the exemption criteria were the common reasons for `@NotPersonalDataTransmission`.

Furthermore, F6, F7, and F10 each added a `@DataAccess` annotation for a data type that was not detected by API calls based on keyword matches, showing the potential of drawing on the developers' knowledge of the app to complement the API-based code analysis results.

6.5 Discussion

6.5.1 Developers and Automated Code Analysis: Better Together

I demonstrated that Matcha can improve the accuracy of Google data safety labels (Section 6.4.1 and Section 6.4.2). Meanwhile, I found that using Matcha improved the developer's knowledge of their app's data practices and overcome misunderstandings in data safety labels (Section 6.4.4 and Section 6.4.3). In general, I believe that the use of Matcha makes developers and code analysis handle the part of work they are most effective at and benefit from each other. It is useful to keep developers informed and involved during this task, not only because their knowledge can complement and refine the code analysis results (Section 6.4.6), but also because they are responsible for privacy in their app. At present, developers are largely motivated by platform requirements to consider privacy, and their focus is largely limited to how to satisfy specific requirements [99]. Using Matcha showed a nice side effect to evoke more in-depth learning for privacy (Section 6.4.3), which might motivate and support further improvements in the app privacy design.

6.5.2 Using Annotations as a Uniform Privacy Language for Developers

I argue that it might not be suitable for developers to directly handle the high-level concepts of privacy labels due to common misinterpretations [102], such as reporting on-device data access as data collection when only data transmitted off the device is considered as collected (Section 6.4.3). Matcha offers an alternative solution, which is to let developers add annotations and edit the XML spec, and let the tool translate them to the privacy label concepts (Figure 6.8). This not only reduced the knowledge burden on developers, but also helped developers effectively reflect on their data use (Section 6.4.4) and allowed for better flexibility (Section 6.4.4). Furthermore, using annotations allows developers to provide privacy information in a more granular and contextualized format, which can potentially be used to automatically generate various types of privacy notices to help relieve developers from the work of upgrading privacy interfaces [100]. Hence, the annotations and XML specification may have the potential to become a privacy domain-specific language for developers, bridging the low-level code behavior and the high-level user-facing disclosure.

6.5.3 Design Implications for Developer Tools for Creating Standardized Privacy Notices

I have shown the design of Matcha successful in helping developers substantially improve the accuracy of their data safety label (Section 6.4.1 and Section 6.4.2), being easy to learn and use (Section 6.4.4 and Section 6.4.5), and being preferred by all study participants (Section 6.4.4). Below, I discuss general design recommendations for developer tools for creating standardized privacy notices based on what worked well in Matcha.

Contextualize the task around code

My studies showed that when using the developer console, developers rarely checked the code to verify their understanding. Moreover, developers had trouble systematically reviewing the code on their own. The design of Matcha suggests that showing questions around the related code can help developers provide more accurate answers and learn more about their app (Section 6.4.4).

Provide scaffolding for embedding privacy information into code

Despite the promising benefits of making privacy information part of the code, it is difficult for developers to manually handle the task given the complexity of the required information and the difficulty of handling an unfamiliar programming feature (e.g., the annotation). In Matcha, the use of the quickfix dialog helped ensure an accurate and correctly-formatted input as well as easing the learning curve. The dialog allows for more space for presenting the full questions in a structured format and verifies the developer's input before it is submitted. This method both provides sufficient guidance for novice users and flexibility for expert users.

Break down the high-level task into low-level questions

Standardized privacy notices need to lump low-level practices into higher-level categories to improve the clarity of the notice to lay people. However, as developers often have misperceptions of the standard taxonomy [102], it is helpful to break down each high-level concept into lower-level questions that probe each aspect of the concept separately. For example, Matcha helped the developers correct their misunderstanding of "data collecion" and "data sharing" by separating data accesses and transmissions and asking about special cases and exemption conditions explicitly when the developer added transmission annotations (Section 6.4.3).

Use proactive guidance and actionable suggestions

One of the main challenges in creating accurate labels is that developers tend to be overconfident in their answers and unaware of their own errors and knowledge gaps (Section 6.2.3 and Section 6.4.4). To better engage the developers in this type of tasks, I used proactive guidance such as the just-in-time tooltips and the errors flagged in code for API calls without a proper annotation (Figure 6.10). I also tried to break down the entire task into smaller actionable steps. My suggestions of accesses and transmissions are grounded in these actionable steps to force the developer to interact with them and ponder on them.

Use precise and specific suggestions

A fundamental challenge in this type of task is to balance the recall and precision of the suggestions. My study showed that the precise and specific API-call based suggestions were better received than more generic keyword-based suggestions (Section 6.4.4). Although it is still necessary to have something like the keyword-based detection that emphasizes a good recall rate, it would be more effective if the precision is also improved so the correct suggestions are not buried in a large volume of irrelevant suggestions.

6.5.4 Future Directions on Tool Design, Platform Actions, and Research Goals

In this work, I took the first step to explore building developer tools for creating accurate standardized privacy notices. There are still many challenges for this type of task, such as how to design a developer tool that is suitable for long-term use, can facilitate collaborative work among multiple developers, and can achieve a wide adoption. In the following, I discuss future work required to address these challenges in different aspects.

Improving suggestions with more advanced program analysis techniques

My proof-of-concept prototype, built with simple code analysis techniques, already achieved substantial improvement. However, more precise suggestions would be helpful for addressing the redundancy issues mentioned by participants (Section 6.4.4). One idea is to enhance the keyword search with large programming models such as Codex [38]. In addition, future research can also study how to combine dynamic program analysis techniques to provide post-hoc data transmission monitoring and feedback.

Platforms verifying label information to increase incentives for improving accuracy

Developers using Matcha can fix many under-reporting errors, but it may also make their app appear to have more privacy issues than apps that do not use it. To solve this problem, the platform should take actions to motivate developers to improve accuracy. For example, if Google provides some level of verification of the data practices and makes the results visible to both the developer and the end users, it can reward developers who honestly disclose their data practices in the privacy label.

Designing and evaluating for other real-world challenges

My study focused on creating a data safety label for a completed app, while future research should also explore other use scenarios. For example, future research may examine how developers add annotations while coding, as discussed in the methodological limitations (Section 6.3.5) and by my participants (Section 6.4.4). Furthermore, future work should study how to support multiple developers or even people in other roles to coordinate changes in data practices and properly encode them in the annotations.

Table 6.3: The `collectionAttribute` field of the `@DataTransmission` annotation encodes the data collection information as a list of predefined attribute values. This table shows the groups of attributes that need to be completed in this field, as well as the corresponding collection questions and the exempt conditions of collection defined by Google.

Attribute name	Values	Original questions / Exempt conditions
`TransmittedOffDevice`	True or False	Is this data collected, shared, or both?
`NotStoredInBackend`	True or False	Is this data processed ephemerally?
`EncryptedInTransit`	True or False	Is all of the user data collected by your app encrypted in transit?
`OptionalCollection`	True or False	Is this data required for your app, or can users choose whether it's collected?
`UserToUserEncryption`	True or False	User data that is sent off device, but that is unreadable by you or anyone other than the sender and recipient as a result of end-to-end encryption does not need to be disclosed.
`CollectedFor`	Seven options: App functionality; Analytics; Developer communications; Advertising or marketing; Fraud prevention, Security and compliance; Personalization; Account Management	Why is this user data collected? Select all that apply.

6.6 Appendix: Annotation Design Details

Table 6.3 and Table 6.4 summarize the design details of the `@DataTransmission` annotation.

6.7 Appendix: Participant Overview

Table 6.5 provides a detailed overview of the background of each participant and their app selected for the study.

Table 6.4: The `sharingAttribute` field of the `@DataTransmission` annotation encodes the data sharing information as a list of predefined attribute values. This table shows the groups of attributes that need to be completed in this field, as well as the corresponding sharing questions and the exempt conditions of sharing defined by Google.

Attribute name	Values	Original questions / Exempt conditions
`SharedWithThirdParty`	True or False	Is this data collected, shared, or both?
`OnlySharedWithServiceProviders`	True or False	Sharing is exempt if transferring user data to a "service provider" that processes it on behalf of the developer.
`OnlySharedForLegalPurposes`	True or False	Sharing is exempt if transferring user data for specific legal purposes, such as in response to a legal obligation or government requests.
`OnlyInitiatedByUser`	True or False	Sharing is exempt if transferring user data to a third party based on a specific user-initiated action, where the user reasonably expects the data to be shared.
`OnlyAfterGettingUserConsent`	True or False	Sharing is exempt if transferring user data to a third party based on a prominent in-app disclosure and consent that meets the requirements described in our User Data policy.
`OnlyTransferringAnonymousData`	True or False	Sharing is exempt if transferring user data that has been fully anonymized so that it can no longer be associated with an individual user.
`SharedFor`	Seven options: App functionality; Analytics; Developer communications; Advertising or marketing; Fraud prevention, Security and compliance; Personalization; Account Management	Why is this user data shared? Select all that apply.

6.8 Appendix: Additional Study Results

We attach more detailed information about our study results. Table 6.6 presents information of the generated labels for each participant. Table 6.7 shows the errors of data types and purposes that were fixed by Matcha for each participant, broken down by different data practices (collection vs. sharing) and the source of the error (first-party vs. third-party).

Table 6.7: Analysis of fixed errors: Errors about data types and purposes

		Data Type			Data Purpose		
		Baseline	Added	Removed	Baseline	Added	Removed
1st-party collect	F1	0	1	0	0	2	0
	F2	6	1	0	12	7	0
	F3	0	4	0	0	4	0
	F4	0	0	0	0	0	0
	F5	1	2	0	1	2	0
	F6	0	0	0	0	0	0
	F7	0	0	0	0	0	0
	F8	0	0	0	0	0	0
	F9	0	0	0	0	0	0
	F10	1	0	1	2	0	2
	F11	0	0	0	0	0	0
	F12	13	0	12	19	0	18
3rd-party collect	F1	4	5	0	5	16	1
	F2	0	3	0	0	3	0
	F3	0	2	0	0	7	0
	F4	0	0	0	0	0	0
	F5	0	6	0	0	19	0
	F6	0	0	0	0	0	0
	F7	0	5	0	0	13	0
	F8	1	4	1	1	12	1
	F9	0	4	0	0	12	0
	F10	0	9	0	0	17	0
	F11	12	2	9	12	4	9
	F12	1	4	1	4	4	4
1st-party share	F1	0	0	0	0	0	0
	F2	0	0	0	0	0	0
	F3	0	0	0	0	0	0
	F4	0	2	0	0	2	0
	F5	0	0	0	0	0	0
	F6	0	0	0	0	0	0
	F7	0	0	0	0	0	0
	F8	0	0	0	0	0	0

	F9	0	0	0	0	0	0
	F10	1	0	1	2	0	2
	F11	0	0	0	0	0	0
	F12	0	0	0	0	0	0
3rd-party share	F1	0	4	0	0	12	0
	F2	0	0	0	0	0	0
	F3	0	0	0	0	0	0
	F4	0	0	0	0	0	0
	F5	0	4	0	0	16	0
	F6	0	0	0	0	0	0
	F7	0	4	0	0	12	0
	F8	0	4	0	0	12	0
	F9	0	4	0	0	12	0
	F10	0	0	0	0	0	0
	F11	11	0	8	11	0	8
	F12	1	0	1	1	0	1

6.9 Appendix: Qualitative Analysis Code book

We present the final code book of our qualitative analysis in Table 6.8.

Table 6.5: Participant Overview. Our sample features a good sample of developers and apps across several dimensions, including participant's geographic location (*Location*), app development purpose (*Purpose*), app development team size (*Team Size*), app downloads (*Downloads*), the current data safety label on Google Play (Current label), and participant's role(s) in the development team (*Participant's Role(s) In Team*). Nine out of the 12 participants had prior experience in publishing apps on the Google Play store (Play Exp.). The app development purposes involve four options, covering situations when the participant developed the app as part of their job (*Job*), as part of their hobby (*Hobby*), for a course project (*Course*), and for a research project (*Research*).

ID	Play Exp.	Location	Purpose	Team Size	Downloads	Current label	Participant's Role(s) in Team
F1	yes	Pakistan	Job	2-5	1M+	No data shared, 4 data types in 3 categories collected (App activity, App info and performance, and Device or other IDs)	Mobile App Developer, Designer
F2	no	U.S.	Course	2-5	Not Play	N/A	Mobile App & Backend Developer, Designer
F3	no	Nigeria	Hobby	1	Not Play	N/A	Mobile App & Backend Developer, Designer, Project Manager
F4	yes	Ukraine	Hobby	1	Not Play	N/A	Mobile App Developer
F5	yes	Georgia	Job	2-5	50K+	No data shared, 6 data types in 4 categories collected (Personal info, Photos and videos, App activity, and Device or other IDs)	Mobile App Developer
F6	no	U.S.	Course	2-5	Not Play	N/A	Mobile App & Backend Developer
F7	yes	Pakistan	Job	1	Not Play	N/A	Mobile App Developer
F8	yes	Pakistan	Job	1	100+	No data shared, no data collected	Mobile App Developer
F9	yes	Bangladesh	Hobby	1	500+	Not provided	Mobile App Developer
F10	yes	Egypt	Course	1	Not Play	N/A	Mobile App Developer
F11	yes	Pakistan	Job	1	100K+	Not provided	Mobile App Developer
F12	yes	India	Job	1	100K	Not provided	Mobile App Developer

Table 6.6: Comparing two labels: Data types and purposes. Using Matcha helped developers report more data types that are collected and shared and the purposes for collecting and sharing the data than using the developer console. Note that the only error made in the Matcha labels was the data type misclassification error for F2, which does not affect the resulting number of data types and purposes, so we do not consider this error separately in the quantitative part.

	Data Type - Collect		Data Type - Share		Data Purpose - Collect		Data Purpose - Share	
	Baseline	Matcha	Baseline	Matcha	Baseline	Matcha	Baseline	Matcha
F1	4	10	0	4	5	22	0	12
F2	6	10	0	0	12	24	0	0
F3	0	6	0	0	0	11	0	0
F4	0	0	0	2	0	0	0	2
F5	1	9	0	4	1	22	0	16
F6	0	0	0	0	0	0	0	0
F7	0	5	0	4	0	13	0	12
F8	1	4	0	4	1	12	0	12
F9	0	4	0	4	0	12	0	12
F10	1	9	1	0	2	17	2	0
F11	12	5	11	3	12	7	11	3
F12	14	5	1	0	23	5	1	0
total	39	67	13	25	56	145	14	69

Table 6.8: The complete codebook of our qualitative analysis of the interview recordings

Theme	Code	Memo	Example
Cause of error	Forgetfulness	The developer mentioned they forgot something about their apps, such as libraries integrated in the app or a feature implemented in the app.	"Sorry, I forgot about this section. It was for users to add text to their photos" (F7)
	Library	The developer mentioned misunderstanding related to third-party libraries.	"This one is more precise, because I did not know that the library was was doing it on its own behind the screen. So that's why I did not put this information." (F2)
	Misunderstanding about the task	The developer mentioned they did not understand something related to the data safety label creation task.	"I did not know that I need to report out other user generated content." (F2)
	lack technical knowledge	The developer mentioned something that demonstrated their misunderstanding about technical concepts.	"I'm not sure if that is if I'm actually using the precise location are like an approximate location. That's where I'm confused." (F2)
Comment on Matcha	prefer Matcha - informative	The developer mentioned Matcha helped them learn useful information.	"I never care about data collection and I don't even look at what we do. So I think I learned a lot from this" (F5)
	prefer Matcha - better flexibility	The developer mentioned Matcha gave them better flexibility in the label creation process.	"I want it because it gives me the flexibility and it give me the feeling of a developer's mindset." (F3)
	prefer Matcha - better engagement	The developer mentioned Matcha better involved them in the task.	"It's a lot more involved process when you're using Matcha than Google Play console." (F2)
	prefer Matcha - accuracy	The developer mentioned Matcha improved the label accuracy.	"I prefer the plugin because it can search the privacy leak for developers." (F4)
	prefer Matcha - easy to use	The developer mentioned Matcha was easy to learn and use.	"I think the quickfix for the annotation is pretty convenient" (F6)
	issue - redundancy	The developer complained that several tasks felt repetitive and unnecessary to them.	"Like for some strange reason I think, it looks for the word search, but isn't search really common?" (F6)

Chapter 7

Conclusion

With the proliferation of computing systems that rely on personal data, developers are facing increasing responsibilities to protect user privacy throughout the software development life cycle. While we often take it for granted that developers should handle all the tasks well, most developers are not privacy experts and are already overloaded with other more salient requirements such as functionality and performance. Unfortunately, existing developer support for privacy is limited, fragmented and ad-hoc, which increases the implementation and maintenance cost for privacy and deepens the awareness and knowledge barriers.

In this thesis, I take the initiative to explore an important yet understudied space, which is to mitigate the privacy issues in software systems by providing better developer support. By reading prior literature and conducting in-depth developer studies, I establish a framework that summarizes the fundamental challenges for achieving privacy by design and the corresponding design needs for better developer tooling for privacy. Then I propose the concept of *privacy annotation* and present three IDE plugins that work hand in hand with privacy annotations to tackle the burning privacy needs for mobile app development.

In this final chapter, I first summarize my main contributions, and then provide an outlook on how to expand the methods proposed in this thesis to tackle further challenges in privacy and other related issues in software development.

7.1 Summary of Contributions

7.1.1 Design Needs for Privacy-Enhancing Developer Support

In my thesis research, I employed a human-centered approach to uncover the key design needs for privacy-enhancing developer support (Table 7.1). They are both the design principles of my IDE plugins and standalone contributions for guiding future research to design better developer support for privacy.

Theme	Key Message
Codify Accountability	Set clear scope of responsibilities for developers and hold developers accountable for privacy
Reduce Burden	Reduce the burden of privacy for developers
Leverage Agency	Leverage developers' knowledge and skills to improve privacy

Table 7.1: The three key design needs identified by my work for helping developers with privacy, which have served as the main design principles of my three IDE plugins presented in this dissertation, and also inform the directions for future research in developer support for privacy.

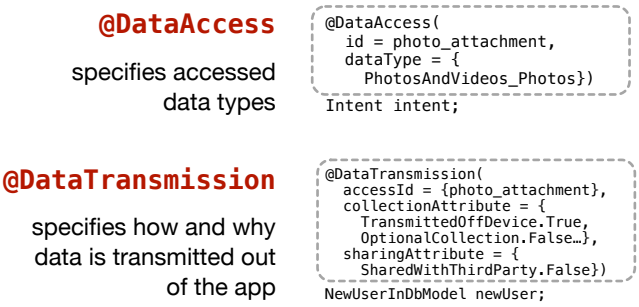


Figure 7.1: Privacy annotations are key elements that support my IDE plugins for privacy. By asking developers to provide one set of privacy annotations, my plugins help them increase knowledge about privacy and fulfill privacy requirements with less work.

7.1.2 Privacy Annotation

A key idea I propose for my dissertation work is *privacy annotation*, a type of structured metadata that embeds privacy information such as data use purposes directly in code (Figure 7.1). Unlike a separate specification file such as the manifest file for Android apps, privacy annotations are attached where personal data is first accessed by the app or any data is transmitted off the device. This property makes some context information naturally embodied in the locations of privacy annotations, which allows privacy annotations to carry rich information about privacy practices while only requiring developers to do a small amount of annotating work.

In addition to documenting key privacy information, privacy annotations also serve as a novel interface for privacy with multiple good properties to address the three key needs summarized in Table 7.1. Its structured format offers a basis to codify developers' responsibilities for privacy. Being part of the code encourages reflections on the privacy aspects of their code and offers opportunities for on-demand learning about privacy knowledge. The lightweight format makes it easy to add annotations and the rich information it contains can be used to automate and streamline many privacy tasks. These benefits are not just theoretical. These benefits of privacy annotations have been evidenced by the three IDE plugins developed to solve real-world privacy problems in mobile apps.

7.1.3 IDE plugins for Privacy

Driven by developers' general needs for privacy-enhancing support, This thesis has explored a specific type of solution: privacy-enhancing development environment.

My first attempt Coconut [95] is an Android Studio plugin that offer developer real-time feedback on privacy issues (**Leverage Agency - Need #7 Increase Awareness**) and quick-fixes (**Lessen Burden - Need #5 Scaffold and Streamline Privacy Tasks**). Coconut also offers an interactive overview panel that summarizes all the data collection and transmission activities based on the annotations (**Promote Accountability - Need #3 Support Auditing**).

Then I designed Honeysuckle [100] on top of Coconut, which supports annotation-guided generation of in-app privacy notices (**Lessen Burden - Need #4 Offload Privacy Tasks**).

Finally I designed Matcha, which helps Android developers generate a privacy nutrition label based on their annotations. Matcha uses simple code analysis to provide developers suggestions for code that potentially handles sensitive user data (**Lessen Burden - Need #5 Scaffold and Streamline Privacy Tasks**) and then allows developers to confirm, refine, or dismiss the suggestions by adding different types of annotations (**Leverage Agency - Need #7 Increase Awareness**). This method greatly reduces developers' burden while still keeping them in the loop. Since the two major app stores now require developers to create privacy nutrition labels, there are natural incentives for developers to install the tool (**Leverage Agency - Need #8 Incentivize Adoption**). To this end, I have released the Matcha plugin on the plugin store and many real-world developers have installed it on their IDE.

It is worth noting that, although the three IDE plugins have been implemented as separate tools, they are built on top of a unified design of privacy annotation, which means the developers can benefit from all their features by adding one set of annotations.

7.2 Future Work

I believe the privacy annotation idea has much greater potential for improving privacy handling in software development than what I have accomplished in my thesis. In this regard, I am enthusiastic to further investigate several directions, which I will outline below.

7.2.1 Annotations for Privacy Control and Audit

Privacy in mobile app development still faces many challenges, including the needs for better privacy controls and support for auditing. *How can we engage and support developers to tackle these challenges?* For my short-term goal, I plan to use privacy annotations and design better tooling to help developers build mobile apps that natively support privacy notices and controls, and to make it easier to audit the apps' data use. I seek to expand annotation-based privacy notice generation to privacy control generation; build systems to support end-to-end auditing with both privacy annotations for frontend data collection and backend data storage; and use static and dynamic program analysis to help different entities audit annotated apps.

7.2.2 Annotations for Privacy in Collaborative Software Development

Despite the potential for helping coordinate privacy-related design choices in software development across a team, my thesis research has mainly studied the use scenario of individual developers. Hence, another short-term goal is to investigate *how to apply annotations to improve privacy in collaborative software development*. Following the same human-centered approach, I plan to first conduct in-depth studies to gain understanding the privacy-related in software development teams. I plan to start by studying the open source community as an example. By inspecting the public commit histories, issues, and code reviews, I seek to understand developers' attitudes, practices, and challenges with respect to privacy. Then I seek to draw on the privacy annotation idea and design developer tooling to tackle the identified challenges.

7.2.3 Annotations for Privacy in Emerging Technologies

Emerging technologies, such as AR/VR, IoT, and blockchain, present new challenges for developing privacy-preserving apps. *How can we help developers easily build compelling and privacy-preserving apps with these technologies?* To this end, I plan to categorize threats into general threats that are consistent with mobile apps and unique threats about specific technologies. Then, I aim to design developer support to tackle the corresponding issues, drawing on my prior research on mobile app developer support. For example, I plan to transfer annotation-based privacy notice generation from mobile apps to VR apps. In addition, I intend to study the re-identification risks of collecting body motion data that do not apply to mobile apps, and designing tooling that supports privacy-preserving body motion data collection for VR app developers.

7.2.4 Annotations for Other Problems

Nowadays, developers are tasked with meeting crucial requirements that they may have limited knowledge of. Privacy is one such example. As a long-term goal, I aim to explore this question: *How can we further support developers to deal with other important requirements that are outside of their primary goal and domain expertise?* For instance, accessibility is also a pressing issue in mobile apps that developers face similar challenges for. Developers are not accessibility experts, they have limited knowledge about accessibility best practices, and they have limited time to tackle the various accessibility needs from different users. Hence, I seek to apply similar ideas, such as giving developers real-time accessibility checks on their user interface designs within the IDE and generating interfaces based on annotations that can adapt to different input/output modality preferences.

Bibliography

[1] Americans and privacy: Concerned, confused and feeling lack of control over their personal information — pew research center. https://web.archive.org/web/20220214011554/https://www.pewresearch.org/internet/2019/11/15/americans-and-privacy-concerned-confused-and-feeling-lack-of-control-over-their-personal-information/. (Accessed on 02/15/2022). 2.1

[2] Apple contractors were allegedly listening to 1,000 siri recordings a day — each - the verge. https://web.archive.org/web/20220101115259/https://www.theverge.com/2019/8/23/20830120/apple-contractors-siri-recordings-listening-1000-a-day-globetech-microsoft-cortana. (Accessed on 02/16/2022). 2.1.1

[3] Ccpa opt-out icon — state of california - department of justice - office of the attorney general. http://web.archive.org/web/20210902015740/https://oag.ca.gov/privacy/ccpa/icons-download. (Accessed on 02/17/2022). 2.3.2

[4] Github - ghostwords/chameleon: Browser fingerprinting protection for everybody. https://github.com/ghostwords/chameleon. (Accessed on 02/17/2022). 2.2.2

[5] Google workers can listen to what people say to its ai home devices — google assistant — the guardian. https://web.archive.org/web/20220201125102/https://www.theguardian.com/technology/2019/jul/11/google-home-assistant-listen-recordings-users-privacy. (Accessed on 02/16/2022). 2.1.2

[6] Privacy badger. http://web.archive.org/web/20220216040654/https://privacybadger.org/. (Accessed on 02/17/2022). 2.2.2

[7] The facebook and cambridge analytica scandal, explained with a simple diagram - vox. https://web.archive.org/web/20220207161110/https://www.vox.com/policy-and-politics/2018/3/23/17151916/facebook-cambridge-analytica-trump-diagram. (Accessed on 02/16/2022). 2.1.1

[8] Yep, human workers are listening to recordings from google assistant, too - the verge. https://web.archive.org/web/20220122175849/https://www.theverge.com/2019/7/11/20690020/google-assistant-home-

human-contractors-listening-recordings-vrt-nws. (Accessed on 02/16/2022). 2.1.1

[9] Improve your code with lint. Available at https://developer.android.com/studio/write/lint.html (2017/05/14), 2017. 2.4.2

[10] Chapter 20. california consumer privacy act regulations (feb 2020 revision). https://web.archive.org/web/20200619170238/https://www.oag.ca.gov/sites/all/files/agweb/pdfs/privacy/ccpa-text-of-mod-redline-020720.pdf, 2 2020. (Accessed on 09/13/2020). 5.1.1

[11] Github - permissions-dispatcher/permissionsdispatcher: A declarative and comprehensive api to handle android runtime permissions. https://web.archive.org/web/20201205195815/https://github.com/permissions-dispatcher/PermissionsDispatcher, 2021. (Accessed on 02/13/2021). 5.1.2

[12] Privacy - features - apple. https://web.archive.org/web/20210126192113/https://www.apple.com/privacy/features/, 2021. (Accessed on 01/29/2021). 2.1.3, 5.1

[13] Privacy in android 11 — android developers. https://web.archive.org/web/20210129195758/https://developer.android.com/about/versions/11/privacy, 2021. (Accessed on 01/29/2021). 5.1

[14] Requesting consent from european users — android — google developers. https://web.archive.org/web/20201109005634/https://developers.google.com/admob/android/eu-consent, 2021. (Accessed on 02/13/2021). 5.1.2

[15] Noura Abdi, Kopo M Ramokapane, and Jose M Such. More than smart speakers: security and privacy perceptions of smart home personal assistants. In *Fifteenth Symposium on Usable Privacy and Security ({SOUPS} 2019)*, pages 451–466, 2019. 1

[16] Yasemin Acar, Michael Backes, Sascha Fahl, Doowon Kim, Michelle L. Mazurek, and Christian Stransky. You get where you're looking for: The impact of information sources on code security. In *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, may 2016. doi: 10.1109/sp.2016.25. URL https://doi.org/10.1109/sp.2016.25. 2.4.1, 2.4.2, 4.1, 4.4.6

[17] Yasemin Acar, Sascha Fahl, and Michelle L. Mazurek. You are not your developer, either: A research agenda for usable security and privacy research beyond end users. In *2016 IEEE Cybersecurity Development (SecDev)*. IEEE, nov 2016. doi: 10.1109/secdev.2016.013. URL https://doi.org/10.1109/secdev.2016.013. 2.4.1, 4.1, 4.5.4

[18] Yasemin Acar, Michael Backes, Sascha Fahl, Simson Garfinkel, Doowon Kim, Michelle L. Mazurek, and Christian Stransky. Comparing the usability of cryptographic APIs. In *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, may 2017. doi: 10.1109/sp.2017.52. URL https://doi.org/10.1109/sp.2017.52. 2.4.1, 4.1

[19] Yuvraj Agarwal and Malcolm Hall. ProtectMyPrivacy: detecting and mitigating privacy

leaks on ios devices using crowdsourcing. In *Proceeding of the 11th annual international conference on Mobile systems, applications, and services - MobiSys '13*. ACM Press, 2013. doi: 10.1145/2462456.2464460. URL https://doi.org/10.1145/2462456.2464460. 2.2.2, 4.1

[20] Hazim Almuhimedi, Florian Schaub, Norman Sadeh, Idris Adjerid, Alessandro Acquisti, Joshua Gluck, Lorrie Faith Cranor, and Yuvraj Agarwal. Your location has been shared 5,398 times!: A field study on mobile app privacy nudging. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. ACM, apr 2015. doi: 10.1145/2702123.2702210. URL https://doi.org/10.1145/2702123.2702210. 2.2.1, 5.1, 5.2.1

[21] Saleema Amershi, Dan Weld, Mihaela Vorvoreanu, Adam Fourney, Besmira Nushi, Penny Collisson, Jina Suh, Shamsi Iqbal, Paul N Bennett, Kori Inkpen, et al. Guidelines for human-ai interaction. In *Proceedings of the 2019 chi conference on human factors in computing systems*, pages 1–13, 2019. 6.2.3

[22] Benjamin Andow, Samin Yaseer Mahmud, Wenyu Wang, Justin Whitaker, William Enck, Bradley Reaves, Kapil Singh, and Tao Xie. PolicyLint: Investigating internal privacy policy contradictions on google play. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 585–602, Santa Clara, CA, August 2019. USENIX Association. ISBN 978-1-939133-06-9. URL https://www.usenix.org/conference/usenixsecurity19/presentation/andow. 2.1.3

[23] Benjamin Andow, Samin Yaseer Mahmud, Justin Whitaker, William Enck, Bradley Reaves, Kapil Singh, and Serge Egelman. Actions speak louder than words: Entity-Sensitive privacy policy and data flow analysis with PoliCheck. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 985–1002. USENIX Association, August 2020. ISBN 978-1-939133-17-5. URL https://www.usenix.org/conference/usenixsecurity20/presentation/andow. 2.1.3

[24] Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves Le Traon, Damien Octeau, and Patrick McDaniel. FlowDroid: precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. *ACM SIGPLAN Notices*, 49(6):259–269, jun 2014. doi: 10.1145/2666356.2594299. URL https://doi.org/10.1145/2666356.2594299. 2.4.2, 5.2.3, 5.4.3, 6.1

[25] GSM Association. Mobile privacy principles, promoting consumer privacy in the mobile ecosystem. Available at http://www.gsma.com/publicpolicy/wp-content/uploads/2016/10/GSMA-Privacy-Principles.pdf (2017/05/14), 2012. 2.4.2

[26] Kathy Wain Yee Au, Yi Fan Zhou, Zhen Huang, and David Lie. PScout: analyzing the android permission specification. In *Proceedings of the 2012 ACM conference on Computer and communications security - CCS '12*. ACM Press, 2012. doi: 10.1145/2382196.2382222. URL https://doi.org/10.1145/2382196.2382222. 5.4.3

[27] David G Balash, Mir Masood Ali, Xiaoyuan Wu, Chris Kanich, and Adam J Aviv. Longitudinal analysis of privacy labels in the apple app store. *arXiv preprint arXiv:2206.02658*,

2022. 6.1, 6.1.2

[28] Rebecca Balebako and Lorrie Cranor. Improving app privacy: Nudging app developers to protect user privacy. *IEEE Security & Privacy*, 12(4):55–58, jul 2014. doi: 10.1109/msp.2014.70. URL https://doi.org/10.1109/msp.2014.70. 2.4.1, 4.1, 5.2, 6.1

[29] Rebecca Balebako, Jaeyeon Jung, Wei Lu, Lorrie Faith Cranor, and Carolyn Nguyen. "little brothers watching you": raising awareness of data leaks on smartphones. In *Proceedings of the Ninth Symposium on Usable Privacy and Security - SOUPS '13*. ACM Press, 2013. doi: 10.1145/2501604.2501616. URL https://doi.org/10.1145/2501604.2501616. 2.2.1, 5.2.1

[30] Rebecca Balebako, Abigail Marsh, Jialiu Lin, Jason Hong, and Lorrie Faith Cranor. The privacy and security behaviors of smartphone app developers. In *Proceedings 2014 Workshop on Usable Security*. Internet Society, 2014. doi: 10.14722/usec.2014.23006. URL https://doi.org/10.14722/usec.2014.23006. 2.4.1, 2.4.2, 3.1, 3.6.3, 4.1, 4.3.2, 5.1, 5.1.2, 5.2.3, 6.1, 6.2.1, 6.4.3

[31] Rebecca Balebako, Richard Shay, and Lorrie Faith Cranor. Is your inseam a biometric? a case study on the role of usability studies in developing public policy. In *Proceedings 2014 Workshop on Usable Security*. Internet Society, 2014. doi: 10.14722/usec.2014.23039. URL https://doi.org/10.14722/usec.2014.23039. 6.2.1

[32] Rebecca Balebako, Florian Schaub, Idris Adjerid, Alessandro Acquisti, and Lorrie Cranor. The impact of timing on the salience of smartphone app privacy notices. In *Proceedings of the 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices*. ACM, oct 2015. doi: 10.1145/2808117.2808119. URL https://doi.org/10.1145/2808117.2808119. 2.2.1

[33] Carlos Bermejo Fernandez, Dimitris Chatzopoulos, Dimitrios Papadopoulos, and Pan Hui. This website uses nudging: Mturk workers' behaviour on cookie consent notices. *Proceedings of the ACM on Human-Computer Interaction*, 5(CSCW2):1–22, 2021. 2.3.1

[34] Bram Bonné, Sai Teja Peddinti, Igor Bilogrevic, and Nina Taft. Exploring decision making with android's runtime permission dialogs using in-context surveys. In *Thirteenth Symposium on Usable Privacy and Security (SOUPS 2017)*, pages 195–210, 2017. 2.1.1, 2.1.2, 5.4.2

[35] Joel Brandt, Mira Dontcheva, Marcos Weskamp, and Scott R. Klemmer. Example-centric programming. In *Proceedings of the 28th international conference on Human factors in computing systems - CHI '10*. ACM Press, 2010. doi: 10.1145/1753326.1753402. URL https://doi.org/10.1145/1753326.1753402. 4.5.4

[36] Virginia Braun and Victoria Clarke. Using thematic analysis in psychology. *Qualitative Research in Psychology*, 3(2):77–101, jan 2006. doi: 10.1191/1478088706qp063oa. URL https://doi.org/10.1191/1478088706qp063oa. 5.3.6

[37] Giovanni Campagna, Rakesh Ramesh, Silei Xu, Michael Fischer, and Monica S Lam. Almond: The architecture of an open, crowdsourced, privacy-preserving, programmable virtual assistant. In *Proceedings of the 26th International Conference on World Wide Web*, pages 341–350, 2017. 1

[38] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021. 6.5.4

[39] Yi Chen, Mingming Zha, Nan Zhang, Dandan Xu, Qianqian Zhao, Xuan Feng, Kan Yuan, Fnu Suya, Yuan Tian, Kai Chen, et al. Demystifying hidden privacy settings in mobile apps. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 570–586. IEEE, 2019. 2.1.3

[40] W. Cheng, Qin Zhao, Bei Yu, and S. Hiroshige. TaintTrace: Efficient flow tracing with dynamic binary rewriting. In *11th IEEE Symposium on Computers and Communications (ISCC'06)*. IEEE, 2006. doi: 10.1109/iscc.2006.158. URL https://doi.org/10.1109/iscc.2006.158. 2.4.2

[41] Erika Chin, Adrienne Porter Felt, Kate Greenwood, and David Wagner. Analyzing inter-application communication in android. In *Proceedings of the 9th international conference on Mobile systems, applications, and services - MobiSys '11*. ACM Press, 2011. doi: 10.1145/1999995.2000018. URL https://doi.org/10.1145/1999995.2000018. 4.1

[42] Saksham Chitkara, Nishad Gothoskar, Suhas Harish, Jason I. Hong, and Yuvraj Agarwal. Does this app really need my location?: Context-aware privacy management for smartphones. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 1(3):1–22, sep 2017. doi: 10.1145/3132029. URL https://doi.org/10.1145/3132029. 2.2.2, 4.2.3, 5.2.1, 5.2.4, 5.4.3, 6.1.1, 6.4.2

[43] Camille Cobb, Lucy Simko, Tadayoshi Kohno, and Alexis Hiniker. User experiences with online status indicators. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, pages 1–12, 2020. 1, 2.1.2

[44] Federal Trade Commission. Mobile privacy disclosures: Building trust through transparency: a federal trade commission staff report. https://www.ftc.gov/sites/default/files/documents/reports/mobile-privacy-disclosures-building-trust-through-transparency-federal-trade-commission-staff-report/130201mobileprivacyreport.pdf, 2013. (Accessed on 08/02/2021). 2.2.1, 2.4.2

[45] Lorrie Faith Cranor. P3p: Making privacy policies more useful. *IEEE Security & Privacy*, 1(6):50–55, 2003. 2.2.1

[46] Lorrie Faith Cranor. Necessary but not sufficient: Standardized mechanisms for privacy notice and choice. *J. on Telecomm. & High Tech. L.*, 10:273, 2012. 2.2.1

[47] Lorrie Faith Cranor, Hana Habib, Yixin Zou, Alessandro Acquisti, Joel Reidenberg, Norman Sadeh, and Florian Schaub. User testing of the proposed ccpa do-not-sell icon. *Submitted to California Office of Attorney General*, 2020. 2.3.2

[48] Sauvik Das, Joanne Lo, Laura Dabbish, and Jason I Hong. Breaking! a typology of security and privacy news and how it's shared. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, pages 1–12, 2018. 1

[49] Fred D. Davis. Perceived usefulness, perceived ease of use, and user acceptance of information technology. *MIS Quarterly*, 13(3):319, sep 1989. doi: 10.2307/249008. URL https://doi.org/10.2307/249008. 5.3.3, 5.3.5

[50] Martin Degeling, Christine Utz, Christopher Lentzsch, Henry Hosseini, Florian Schaub, and Thorsten Holz. We value your privacy... now take some cookies: Measuring the gdpr's impact on web privacy. *arXiv preprint arXiv:1808.05096*, 2018. 2.3.1

[51] Android Developer Document. App permissions best practices — android developers. https://developer.android.com/training/permissions/usage-notes, 2020. (Accessed on 09/17/2020). 5.2.1

[52] Android Official Documentation. Best practices for permissions and identifiers. Available at https://developer.android.com/training/best-permissions-ids.html (2017/05/14), 2017. 2.4.2

[53] Android Official Documentation. Best practices for unique identifiers. Available at https://developer.android.com/training/articles/user-data-ids.html (2017/05/14), 2017. (document), 4.3

[54] Ian Drosos, Titus Barik, Philip J Guo, Robert DeLine, and Sumit Gulwani. Wrex: A unified programming-by-example interaction for synthesizing readable code for data scientists. In *Proceedings of the 2020 CHI conference on human factors in computing systems*, pages 1–12, 2020. 6.3.1, 6.3.2

[55] Julia C Dunbar, Emily Bascom, Ashley Boone, and Alexis Hiniker. Is someone listening? audio-related privacy perceptions and design recommendations from guardians, pragmatists, and cynics. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 5(3):1–23, 2021. 1

[56] Pardis Emami-Naeini, Yuvraj Agarwal, Lorrie Faith Cranor, and Hanan Hibshi. Ask the experts: What should be on an IoT privacy and security label? In *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, may 2020. doi: 10.1109/sp40000.2020.00043. URL https://doi.org/10.1109/sp40000.2020.00043. 2.2.1, 6.1, 6.1.2

[57] William Enck, Peter Gilbert, Seungyeop Han, Vasant Tendulkar, Byung-Gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N. Sheth. TaintDroid: An information-flow tracking system for realtime privacy monitoring on smartphones. *ACM Transactions on Computer Systems*, 32(2):1–29, jun 2014. doi: 10.1145/2619091. URL https://doi.org/10.1145/2619091. 2.4.2, 4.1, 6.1

[58] Steven Englehardt and Arvind Narayanan. Online tracking: A 1-million-site measurement and analysis. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 1388–1401, 2016. 2.2.2

[59] Benjamin Fabian, Tatiana Ermakova, and Tino Lentz. Large-scale readability analysis of privacy policies. In *Proceedings of the International Conference on Web Intelligence*. ACM, aug 2017. doi: 10.1145/3106426.3106427. URL https://doi.org/10.1145/3106426.3106427. 2.2.1

[60] Adrienne Porter Felt, Elizabeth Ha, Serge Egelman, Ariel Haney, Erika Chin, and David

Wagner. Android permissions: User attention, comprehension, and behavior. In *Proceedings of the Eighth Symposium on Usable Privacy and Security - SOUPS '12*. ACM Press, 2012. doi: 10.1145/2335356.2335360. URL https://doi.org/10.1145/2335356.2335360. 2.1.3

[61] Felix Fischer, Konstantin Bottinger, Huang Xiao, Christian Stransky, Yasemin Acar, Michael Backes, and Sascha Fahl. Stack overflow considered harmful? the impact of copy&paste on android application security. In *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, may 2017. doi: 10.1109/sp.2017.31. URL https://doi.org/10.1109/sp.2017.31. 2.4.1, 4.1

[62] Geoffrey A. Fowler. iphone app privacy labels are a great idea, except when apple lets them deceive - the washington post. https://web.archive.org/web/20220630055538/https://www.washingtonpost.com/technology/2021/01/29/apple-privacy-nutrition-label/, 1 2021. (Accessed on 08/27/2022). 6.1

[63] Jack Gardner, Yuanyuan Feng, Kayla Reiman, Zhi Lin, Akshath Jain, and Norman Sadeh. Helping mobile application developers create accurate privacy labels. In *2022 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 212–230. IEEE, 2022. 6.1.2

[64] Clint Gibler, Jonathan Crussell, Jeremy Erickson, and Hao Chen. AndroidLeaks: Automatically detecting potential privacy leaks in android applications on a large scale. In *Trust and Trustworthy Computing*, pages 291–307. Springer Berlin Heidelberg, 2012. doi: 10.1007/978-3-642-30921-2_17. URL https://doi.org/10.1007/978-3-642-30921-2_17. 5.2.1

[65] Alyssa Glass, Deborah L. McGuinness, and Michael Wolverton. Toward establishing trust in adaptive agents. In *Proceedings of the 13th international conference on Intelligent user interfaces - IUI '08*. ACM Press, 2008. doi: 10.1145/1378773.1378804. URL https://doi.org/10.1145/1378773.1378804. 4.5.2

[66] Joshua Gluck, Florian Schaub, Amy Friedman, Hana Habib, Norman Sadeh, Lorrie Faith Cranor, and Yuvraj Agarwal. How short is too short? implications of length and framing on the effectiveness of privacy notices. In *Twelfth Symposium on Usable Privacy and Security (SOUPS 2016)*, pages 321–340, 2016. 2.2.1

[67] Michael I. Gordon, Deokhwan Kim, Jeff Perkins, Limei Gilham, Nguyen Nguyen, and Martin Rinard. Information-flow analysis of android applications in DroidSafe. In *Proceedings 2015 Network and Distributed System Security Symposium*. Internet Society, 2015. doi: 10.14722/ndss.2015.23089. URL https://doi.org/10.14722/ndss.2015.23089. 2.4.2

[68] Philip Guo. Ten million users and ten years later: Python tutor's design guidelines for building scalable and sustainable research software in academia. In *The 34th Annual ACM Symposium on User Interface Software and Technology*, pages 1235–1251, 2021. 6.3.1

[69] Hana Habib, Yixin Zou, Aditi Jannu, Neha Sridhar, Chelse Swoopes, Alessandro Acquisti,

Lorrie Faith Cranor, Norman Sadeh, and Florian Schaub. An empirical analysis of data deletion and {Opt-Out} choices on 150 websites. In *Fifteenth Symposium on Usable Privacy and Security (SOUPS 2019)*, pages 387–406, 2019. 2.1.3

[70] Hana Habib, Yixin Zou, Yaxing Yao, Alessandro Acquisti, Lorrie Cranor, Joel Reidenberg, Norman Sadeh, and Florian Schaub. Toggles, dollar signs, and triangles: How to (in) effectively convey privacy choices with icons and link texts. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, pages 1–25, 2021. 2.3.2

[71] Irit Hadar, Tomer Hasson, Oshrat Ayalon, Eran Toch, Michael Birnhack, Sofia Sherman, and Arod Balissa. Privacy by designers: software developers' privacy mindset. *Empirical Software Engineering*, 23(1):259–289, apr 2017. doi: 10.1007/s10664-017-9517-1. URL https://doi.org/10.1007/s10664-017-9517-1. 2.4.1, 3.6.1, 4.1, 5.1, 5.1.2

[72] Marian Harbach, Markus Hettig, Susanne Weber, and Matthew Smith. Using personal examples to improve risk communication for security & privacy decisions. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, apr 2014. doi: 10.1145/2556288.2556978. URL https://doi.org/10.1145/2556288.2556978. 5.4.4

[73] Kamala D. Harris. Privacy on the go, recommendations for the mobile ecosystem. Available at https://oag.ca.gov/sites/all/files/agweb/pdfs/privacy/privacy_on_the_go.pdf (2017/05/14), 2013. 2.4.2

[74] Sandra G. Hart and Lowell E. Staveland. Development of NASA-TLX (task load index): Results of empirical and theoretical research. In *Advances in Psychology*, pages 139–183. Elsevier, 1988. doi: 10.1016/s0166-4115(08)62386-9. URL https://doi.org/10.1016/s0166-4115(08)62386-9. 5.3.3, 5.3.5

[75] Monique M Hennink, Bonnie N Kaiser, and Vincent C Marconi. Code saturation versus meaning saturation: how many interviews are enough? *Qualitative health research*, 27 (4):591–608, 2017. 6.3.2

[76] David Hovemeyer and William Pugh. Finding bugs is easy. *ACM SIGPLAN Notices*, 39(12):92, dec 2004. doi: 10.1145/1052883.1052895. URL https://doi.org/10.1145/1052883.1052895. 2.4.2

[77] Jianjun Huang, Zhichun Li, Xusheng Xiao, Zhenyu Wu, Kangjie Lu, Xiangyu Zhang, and Guofei Jiang. Supor: Precise and scalable sensitive user input detection for android apps. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 977–992, 2015. 5.2.3, 6.1, 6.2.1

[78] Luigi Lo Iacono and Peter Leo Gorski. I do and i understand. not yet true for security APIs. so sad. In *Proceedings 2nd European Workshop on Usable Security*. Internet Society, 2017. doi: 10.14722/eurousec.2017.23015. URL https://doi.org/10.14722/eurousec.2017.23015. 2.4.1, 4.1

[79] Shubham Jain and Janne Lindqvist. Should i protect you? understanding developers' behavior to privacy-preserving APIs. In *Proceedings 2014 Workshop on Usable Security*. Internet Society, 2014. doi: 10.14722/usec.2014.23045. URL https://doi.org/10.14722/usec.2014.23045. 4.1

[80] Carlos Jensen and Colin Potts. Privacy policies as decision-making tools: an evaluation of online privacy notices. In *Proceedings of the 2004 conference on Human factors in computing systems - CHI '04*. ACM Press, 2004. doi: 10.1145/985692.985752. URL https://doi.org/10.1145/985692.985752. 2.2.1

[81] Haojian Jin, Minyi Liu, Kevan Dodhia, Yuanchun Li, Gaurav Srivastava, Matthew Fredrikson, Yuvraj Agarwal, and Jason I Hong. Why are they collecting my data? inferring the purposes of network traffic in mobile apps. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 2(4):1–27, 2018. 2.2.2, 6.1, 6.2.1

[82] Haojian Jin, Minyi Liu, Kevan Dodhia, Yuanchun Li, Gaurav Srivastava, Matthew Fredrikson, Yuvraj Agarwal, and Jason I. Hong. "why are they collecting my data?": Inferring the purposes of network traffic in mobile apps. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 2018. 4.2.2

[83] Brittany Johnson, Yoonki Song, Emerson Murphy-Hill, and Robert Bowdidge. Why don't software developers use static analysis tools to find bugs? In *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, may 2013. doi: 10.1109/icse.2013.6606613. URL https://doi.org/10.1109/icse.2013.6606613. 2.4.1, 2.4.2

[84] Ruogu Kang, Laura Dabbish, Nathaniel Fruchter, and Sara Kiesler. {"My} data just goes {Everywhere:"} user mental models of the internet and implications for privacy and security. In *Eleventh Symposium on Usable Privacy and Security (SOUPS 2015)*, pages 39–52, 2015. 2.1.1, 2.1.2

[85] Farzaneh Karegar, John Sören Pettersson, and Simone Fischer-Hübner. The dilemma of user engagement in privacy notices: Effects of interaction modes and habituation on user attention. *ACM Transactions on Privacy and Security*, 23(1):1–38, feb 2020. doi: 10.1145/3372296. URL https://doi.org/10.1145/3372296. 2.2.1

[86] G. Karjoth and M. Schunter. A privacy policy model for enterprises. In *Proceedings 15th IEEE Computer Security Foundations Workshop. CSFW-15*. IEEE Comput. Soc. doi: 10.1109/csfw.2002.1021821. URL https://doi.org/10.1109/csfw.2002.1021821. 4.2.2

[87] Patrick Gage Kelley, Joanna Bresee, Lorrie Faith Cranor, and Robert W. Reeder. A "nutrition label" for privacy. In *Proceedings of the 5th Symposium on Usable Privacy and Security - SOUPS '09*. ACM Press, 2009. doi: 10.1145/1572532.1572538. URL https://doi.org/10.1145/1572532.1572538. 2.2.1, 2.3.3, 6.1, 6.1.2

[88] Patrick Gage Kelley, Lucian Cesca, Joanna Bresee, and Lorrie Faith Cranor. Standardizing privacy notices: an online study of the nutrition label approach. In *Proceedings of the 28th international conference on Human factors in computing systems - CHI '10*. ACM Press, 2010. doi: 10.1145/1753326.1753561. URL https://doi.org/10.1145/1753326.1753561. 2.2.1, 6.1.2

[89] Patrick Gage Kelley, Lorrie Faith Cranor, and Norman Sadeh. Privacy as part of the app decision-making process. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, apr 2013. doi: 10.1145/2470654.2466466. URL https:

//doi.org/10.1145/2470654.2466466. 2.2.1, 2.3.3, 6.1, 6.1.2

[90] Amy J. Ko and Brad A. Myers. Designing the whyline. In *Proceedings of the 2004 conference on Human factors in computing systems - CHI '04*. ACM Press, 2004. doi: 10.1145/985692.985712. URL https://doi.org/10.1145/985692.985712. 4.5.4

[91] Spyros Kokolakis. Privacy attitudes and privacy behaviour: A review of current research on the privacy paradox phenomenon. *Computers & security*, 64:122–134, 2017. 2.1.2

[92] Konrad Kollnig, Anastasia Shuba, Max Van Kleek, Reuben Binns, and Nigel Shadbolt. Goodbye tracking? impact of ios app tracking transparency and privacy labels. *arXiv preprint arXiv:2204.03556*, 2022. 6.1, 6.1.2

[93] Isadora Krsek, Kimi V Wenzel, Sauvik Das, Jason I Hong, and Laura A Dabbish. To self-persuade or be persuaded: Examining interventions for users' privacy setting selection. 2022. 2.1.3

[94] Li Li, Alexandre Bartel, Jacques Klein, Yves Le Traon, Steven Arzt, Siegfried Rasthofer, Eric Bodden, Damien Octeau, and Patrick Mcdaniel. I know what leaked in your pocket: uncovering privacy leaks on android apps with static taint analysis. *arXiv preprint arXiv:1404.7431*, 2014. 2.4.2

[95] Tianshi Li, Yuvraj Agarwal, and Jason I. Hong. Coconut: An ide plugin for developing privacy-friendly apps. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 2(4):1–35, dec 2018. doi: 10.1145/3287056. URL https://doi.org/10.1145/3287056. 1, 1.1.1, 1.1.2, 2.2.1, 2.4.2, 2.4.2, 3, 3.1, 3.6.1, 3.6.1, 3.6.1, 3.6.2, 3.6.3, 3.6.3, 5.1, 5.1.2, 5.2, 5.2.3, 5.2.4, 5.4.1, 5.4.3, 6.1, 6.1, 6.2.1, 6.2.5, 6.3.1, 6.4.3, 7.1.3

[96] Tianshi Li, Cori Faklaris, Jennifer King, Yuvraj Agarwal, Laura Dabbish, Jason I Hong, et al. Decentralized is not risk-free: Understanding public perceptions of privacy-utility trade-offs in covid-19 contact-tracing apps. *arXiv preprint arXiv:2005.11957*, 2020. 2.1.2

[97] Tianshi Li, Camille Cobb, Jackie Junrui Yang, Sagar Baviskar, Yuvraj Agarwal, Beibei Li, Lujo Bauer, and Jason I Hong. What makes people install a covid-19 contact-tracing app? understanding the influence of app design and individual difference on contact-tracing app adoption intention. *Pervasive and Mobile Computing*, 75:101439, 2021. 2.1.2

[98] Tianshi Li, Elizabeth Louie, Laura Dabbish, and Jason I. Hong. How developers talk about personal data and what it means for user privacy: A case study of a developer forum on reddit. *Proceedings of the ACM on Human-Computer Interaction*, 4(CSCW3):1–28, jan 2021. doi: 10.1145/3432919. URL https://doi.org/10.1145/3432919. 1, 1.1.1, 3, 3.6.3, 3.6.3, 3.6.3, 5.1, 5.1.1, 5.1.2, 5.4.2

[99] Tianshi Li, Elizabeth Louie, Laura Dabbish, and Jason I. Hong. How developers talk about personal data and what it means for user privacy: A case study of a developer forum on reddit. *Proceedings of the ACM on Human-Computer Interaction*, 4(CSCW3):1–28, jan 2021. doi: 10.1145/3432919. URL https://doi.org/10.1145/3432919. 6.5.1

[100] Tianshi Li, Elijah B. Neundorfer, Yuvraj Agarwal, and Jason I. Hong. Honeysuckle: Annotation-guided code generation of in-app privacy notices. *Proceedings of the ACM*

on Interactive, Mobile, Wearable and Ubiquitous Technologies, 5(3):1–27, sep 2021. doi: 10.1145/3478097. URL https://doi.org/10.1145/3478097. 1, 1.1.2, 2.1.3, 2.2.1, 2.2.1, 3.6.2, 6.1, 6.2.1, 6.3.1, 6.3.2, 6.5.2, 7.1.3

[101] Tianshi Li, Kayla Reiman, Yuvraj Agarwal, Lorrie Faith Cranor, and Jason I Hong. Understanding challenges for developers to create accurate privacy nutrition labels. *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, 2022. 1, 1.1.3, 2.2.1, 3, 3.6.1, 3.6.1, 3.6.2, 3.6.2, 3.6.2

[102] Tianshi Li, Kayla Reiman, Yuvraj Agarwal, Lorrie Faith Cranor, and Jason I Hong. Understanding challenges for developers to create accurate privacy nutrition labels. In *CHI Conference on Human Factors in Computing Systems*, pages 1–24, 2022. 6.1, 6.1.2, 6.2.1, 6.2.2, 6.3.5, 6.4.3, 6.4.3, 6.5.2, 6.5.3

[103] Yuanchun Li, Fanglin Chen, Toby Jia-Jun Li, Yao Guo, Gang Huang, Matthew Fredrikson, Yuvraj Agarwal, and Jason I. Hong. PrivacyStreams: Enabling transparency in personal data processing for mobile apps. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 1(3):1–26, sep 2017. doi: 10.1145/3130941. URL https://doi.org/10.1145/3130941. 2.4.2, 3.6.3, 4.1, 4.2.2

[104] Yucheng Li, Deyuan Chen, Tianshi Li, Yuvraj Agarwal, Lorrie Faith Cranor, and Jason I Hong. Understanding ios privacy nutrition labels: An exploratory large-scale analysis of app store data. In *CHI Conference on Human Factors in Computing Systems Extended Abstracts*, pages 1–7, 2022. 6.1, 6.1.2

[105] Yucheng Li, Deyuan Chen, Tianshi Li, Yuvraj, Lorrie Faith Cranor, and Jason I Hong. Understanding ios privacy nutrition labels: An exploratory large-scale analysis of app store data. 2022. 2.2.1, 3.6.2

[106] Jialiu Lin, Norman Sadeh, Shahriyar Amini, Janne Lindqvist, Jason I. Hong, and Joy Zhang. Expectation and purpose: understanding users' mental models of mobile app privacy through crowdsourcing. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing - UbiComp '12*. ACM Press, 2012. doi: 10.1145/2370216.2370290. URL https://doi.org/10.1145/2370216.2370290. 2.1.1, 2.1.3, 2.2.1, 5.1.1, 5.2, 5.2.1, 5.4.2

[107] Jialiu Lin, Bin Liu, Norman Sadeh, and Jason I Hong. Modeling users' mobile app privacy preferences: Restoring usability in a sea of permission settings. In *10th Symposium On Usable Privacy and Security (SOUPS 2014)*, pages 199–212, 2014. 2.1.3, 2.2.1

[108] Bin Liu, Mads Schaarup Andersen, Florian Schaub, Hazim Almuhimedi, Shikun Aerin Zhang, Norman Sadeh, Yuvraj Agarwal, and Alessandro Acquisti. Follow my recommendations: A personalized privacy assistant for mobile app permissions. In *Twelfth Symposium on Usable Privacy and Security (SOUPS 2016)*, pages 27–41, 2016. 2.2.1

[109] Michael Xieyang Liu, Jane Hsieh, Nathan Hahn, Angelina Zhou, Emily Deng, Shaun Burley, Cynthia Taylor, Aniket Kittur, and Brad A. Myers. Unakite: Scaffolding developers' decision-making using the web. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology*. ACM, oct 2019. doi: 10.1145/3332165.3347908. URL https://doi.org/10.1145/3332165.3347908. 5.1.2

[110] Michael Xieyang Liu, Andrew Kuznetsov, Yongsung Kim, Joseph Chee Chang, Aniket Kittur, and Brad A Myers. Wigglite: Low-cost information collection and triage. In *Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology*, pages 1–16, 2022. 6.3.2

[111] Xueqing Liu, Yue Leng, Wei Yang, Wenyu Wang, Chengxiang Zhai, and Tao Xie. A large-scale empirical study on android runtime-permission rationale messages. In *2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, oct 2018. doi: 10.1109/vlhcc.2018.8506574. URL https://doi.org/10.1109/vlhcc.2018.8506574. 2.1.3, 5.1, 5.1.2, 5.2.3

[112] Kai-Uwe Loser and Martin Degeling. Security and privacy as hygiene factors of developer behavior in small and agile teams. In *IFIP Advances in Information and Communication Technology*, pages 255–265. Springer Berlin Heidelberg, 2014. doi: 10.1007/978-3-662-44208-1_21. URL https://doi.org/10.1007/978-3-662-44208-1_21. 2.4.1, 4.1

[113] Kangjie Lu, Zhichun Li, Vasileios P. Kemerlis, Zhenyu Wu, Long Lu, Cong Zheng, Zhiyun Qian, Wenke Lee, and Guofei Jiang. Checking more and alerting less: Detecting privacy leakages via enhanced data-flow analysis and peer voting. In *Proceedings 2015 Network and Distributed System Security Symposium*. Internet Society, 2015. doi: 10.14722/ndss.2015.23287. URL https://doi.org/10.14722/ndss.2015.23287. 5.2.1

[114] Long Lu, Zhichun Li, Zhenyu Wu, Wenke Lee, and Guofei Jiang. CHEX. In *Proceedings of the 2012 ACM conference on Computer and communications security - CCS '12*. ACM Press, 2012. doi: 10.1145/2382196.2382223. URL https://doi.org/10.1145/2382196.2382223. 4.1

[115] Stephen McCamant and Michael D Ernst. Quantitative information-flow tracking for c and related languages. 2006. 2.4.2

[116] Aleecia M McDonald and Lorrie Faith Cranor. The cost of reading privacy policies. *I/S: A Journal of Law and Policy for the Information Society*, 4:543, 2008. 2.1.3, 2.2.1, 2.2.1, 6.1.2

[117] Aleecia M McDonald and Lorrie Faith Cranor. Americans' attitudes about internet behavioral advertising practices. In *Proceedings of the 9th annual ACM workshop on Privacy in the electronic society*, pages 63–72, 2010. 2.1.2

[118] Gerrit Meixner, Fabio Paternò, and Jean Vanderdonckt. Past, present, and future of model-based user interface development. *icom*, 10(3):2–11, nov 2011. doi: 10.1524/icom.2011.0026. URL https://doi.org/10.1524/icom.2011.0026. 5.4.1

[119] Abraham H. Mhaidli, Yixin Zou, and Florian Schaub. "we can't live without them!" app developers' adoption of ad networks and their considerations of consumer risks. SOUPS'19, page 225–244, USA, 2019. USENIX Association. ISBN 9781939133052. 3.6.1, 6.1

[120] Victor Morel and Raúl Pardo. Sok: Three facets of privacy policies. In *Proceedings of the 19th Workshop on Privacy in the Electronic Society*, pages 41–56, 2020. 2.2.1

[121] Andreas Mueller, Peter Forbrig, and Clemens Cap. Model-based user interface design using markup concepts. In *Interactive Systems: Design, Specification, and Verification*, pages 16–27. Springer Berlin Heidelberg, 2001. doi: 10.1007/3-540-45522-1_2. URL https://doi.org/10.1007/3-540-45522-1_2. 5.1.1

[122] Brad A Myers, Amy J Ko, Thomas D LaToza, and YoungSeok Yoon. Programmers are users too: Human-centered methods for improving programming tools. *Computer*, 49(7): 44–52, 2016. 1, 2.4

[123] Yuhong Nan, Min Yang, Zhemin Yang, Shunfan Zhou, Guofei Gu, and XiaoFeng Wang. Uipicker: User-input privacy identification in mobile applications. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 993–1008, 2015. 6.1, 6.2.1

[124] James Newsome and Dawn Song. Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software. 2005. 2.4.2

[125] Duc Cuong Nguyen, Dominik Wermke, Yasemin Acar, Michael Backes, Charles Weir, and Sascha Fahl. A stitch in time: Supporting android developers in writingsecure code. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, oct 2017. doi: 10.1145/3133956.3133977. URL https://doi.org/10.1145/3133956.3133977. 2.4.2, 4.1

[126] Trung Tin Nguyen, Michael Backes, Ninja Marnau, and Ben Stock. Share first, ask later (or never?) studying violations of {GDPR's} explicit consent in android apps. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 3667–3684, 2021. 2.3.1

[127] Helen Nissenbaum. Privacy as contextual integrity. *Wash. L. Rev.*, 79:119, 2004. 2.1.1

[128] Don Norman. *The design of everyday things: Revised and expanded edition*. Basic books, 2013. 6.2.4

[129] Midas Nouwens, Ilaria Liccardi, Michael Veale, David Karger, and Lalana Kagal. Dark patterns after the GDPR: Scraping consent pop-ups and demonstrating their influence. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. ACM, apr 2020. doi: 10.1145/3313831.3376321. URL https://doi.org/10.1145/3313831.3376321. 2.3.1, 5.1

[130] Damien Octeau, Patrick McDaniel, Somesh Jha, Alexandre Bartel, Eric Bodden, Jacques Klein, and Yves Le Traon. Effective inter-component communication mapping in android with epicc: An essential step towards holistic security analysis. In *Proceedings of the 22nd USENIX security symposium*, pages 543–558, 2013. 2.4.2

[131] Future of Privacy Forum and the Center for Democracy & Technology. Best practices for mobile application developers. Available at https://www.cdt.org/files/pdfs/Best-Practices-Mobile-App-Developers.pdf (2017/05/14), 2012. 2.4.2

[132] Office of the Australian Information Commissioner. Mobile privacy: a better practice guide for mobile app developers. Available at https://www.oaic.gov.au/agencies-and-organisations/guides/guide-for-mobile-app-developers (2017/05/14), 2014. 2.4.2

[133] Office of the Privacy Commissioner of Canada. Seizing opportunity: Good privacy

practices for developing mobile apps. Available at https://www.priv.gc.ca/en/privacy-topics/technology-and-privacy/mobile-devices-and-apps/gd_app_201210/ (2017/05/14), 2012. 2.4.2

[134] Information Commissioner's Office. Privacy in mobile apps, guidance for app developers. Available at https://ico.org.uk/media/for-organisations/documents/1596/privacy-in-mobile-apps-dp-guidance.pdf (2017/05/14), 2013. 2.4.2

[135] Cheul Young Park, Cori Faklaris, Siyan Zhao, Alex Sciuto, Laura Dabbish, and Jason Hong. Share and share alike? an exploration of secure behaviors in romantic relationships. In *Fourteenth Symposium on Usable Privacy and Security ({SOUPS} 2018)*, pages 83–102, 2018. 1

[136] Article 29 Data Protection Working Party. Opinion 02/2013 on apps on smart devices. Available at http://ec.europa.eu/justice/data-protection/article-29/documentation/opinion-recommendation/files/2013/wp202_en.pdf (2017/05/14), 2013. 2.4.2

[137] Feng Qin, Cheng Wang, Zhenmin Li, Ho seop Kim, Yuanyuan Zhou, and Youfeng Wu. LIFT: A low-overhead practical information flow tracking system for detecting security attacks. In *2006 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'06)*. IEEE, dec 2006. doi: 10.1109/micro.2006.29. URL https://doi.org/10.1109/micro.2006.29. 2.4.2

[138] Ashwini Rao, Florian Schaub, Norman Sadeh, Alessandro Acquisti, and Ruogu Kang. Expecting the unexpected: Understanding mismatched privacy expectations online. In *Twelfth Symposium on Usable Privacy and Security (SOUPS 2016)*, pages 77–96, 2016. 2.2.1, 5.2.1

[139] Siegfried Rasthofer, Steven Arzt, and Eric Bodden. A machine-learning approach for classifying and categorizing android sources and sinks. In *Proceedings 2014 Network and Distributed System Security Symposium*. Internet Society, 2014. doi: 10.14722/ndss.2014.23039. URL https://doi.org/10.14722/ndss.2014.23039. 5.4.3

[140] Jingjing Ren, Ashwin Rao, Martina Lindorfer, Arnaud Legout, and David Choffnes. ReCon: Revealing and controlling pii leaks in mobile network traffic. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, jun 2016. doi: 10.1145/2906388.2906392. URL https://doi.org/10.1145/2906388.2906392. 2.2.2

[141] Franziska Roesner, Tadayoshi Kohno, and David Wetherall. Detecting and defending against {Third-Party} tracking on the web. In *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pages 155–168, 2012. 2.2.2

[142] Caitlin Sadowski, Jeffrey van Gogh, Ciera Jaspan, Emma Soderberg, and Collin Winter. Tricorder: Building a program analysis ecosystem. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*. IEEE, may 2015. doi: 10.1109/icse.2015.76. URL https://doi.org/10.1109/icse.2015.76. 2.4.1, 2.4.2

[143] Johnny Saldaña. *The coding manual for qualitative researchers*. Sage, 2015. 6.3.4

[144] Shruti Sannon, Natalya N Bazarova, and Dan Cosley. Privacy lies: Understanding how, when, and why people lie to protect their privacy in multiple online contexts. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, pages 1–13, 2018. 2.1.2

[145] Florian Schaub, Rebecca Balebako, Adam L. Durity, and Lorrie Faith Cranor. A design space for effective privacy notices. In *Eleventh Symposium On Usable Privacy and Security (SOUPS 2015)*, pages 1–17, Ottawa, July 2015. USENIX Association. ISBN 978-1-931971-249. URL https://www.usenix.org/conference/soups2015/proceedings/presentation/schaub. 2.2.1, 2.2.1, 5.1, 5.1.1, 5.1.2, 5.2.1, 5.2.1, 6.1.2

[146] Florian Schaub, Rebecca Balebako, and Lorrie Faith Cranor. Designing effective privacy notices and controls. *IEEE Internet Computing*, pages 1–1, 2017. doi: 10.1109/mic.2017.265102930. URL https://doi.org/10.1109/mic.2017.265102930. 2.2.1

[147] Shayak Sen, Saikat Guha, Anupam Datta, Sriram K. Rajamani, Janice Tsai, and Jeannette M. Wing. Bootstrapping privacy compliance in big data systems. In *2014 IEEE Symposium on Security and Privacy*. IEEE, may 2014. doi: 10.1109/sp.2014.28. URL https://doi.org/10.1109/sp.2014.28. 2.4.2

[148] Swapneel Sheth, Gail Kaiser, and Walid Maalej. Us and them: a study of privacy requirements across north america, asia, and europe. In *Proceedings of the 36th International Conference on Software Engineering*. ACM, may 2014. doi: 10.1145/2568225.2568244. URL https://doi.org/10.1145/2568225.2568244. 2.4.1, 3.6.1, 5.1, 5.1.2

[149] Irina Shklovski, Scott D Mainwaring, Halla Hrund Skúladóttir, and Höskuldur Borgthorsson. Leakiness and creepiness in app space: Perceptions of privacy and mobile app use. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 2347–2356, 2014. 2.1.1

[150] Justin Smith, Brittany Johnson, Emerson Murphy-Hill, Bill Chu, and Heather Richter Lipford. Questions developers ask while diagnosing potential security vulnerabilities with static analysis. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2015*. ACM Press, 2015. doi: 10.1145/2786805.2786812. URL https://doi.org/10.1145/2786805.2786812. 2.4.1

[151] Sooel Son, Kathryn S. McKinley, and Vitaly Shmatikov. RoleCast. *ACM SIGPLAN Notices*, 46(10):1069, oct 2011. doi: 10.1145/2076021.2048146. URL https://doi.org/10.1145/2076021.2048146. 4.1

[152] Yihang Song and Urs Hengartner. PrivacyGuard: A vpn-based platform to detect information leakage on android devices. In *Proceedings of the 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices*. ACM, oct 2015. doi: 10.1145/2808117.2808120. URL https://doi.org/10.1145/2808117.2808120. 2.2.2

[153] Yunpeng Song, Cori Faklaris, Zhongmin Cai, Jason I Hong, and Laura Dabbish. Normal and easy: Account sharing practices in the workplace. *Proceedings of the ACM on Human-Computer Interaction*, 3(CSCW):1–25, 2019. 1

[154] Gaurav Srivastava, Kunal Bhuwalka, Swarup Kumar Sahoo, Saksham Chitkara, Kevin Ku, Matt Fredrikson, Jason Hong, and Yuvraj Agarwal. Privacyproxy: Leveraging crowdsourcing and in situ traffic analysis to detect and mitigate information leakage. *arXiv preprint arXiv:1708.06384*, 2017. 2.2.2, 5.1, 5.1.2

[155] FTC Staff. Protecting consumer privacy in an era of rapid change–a proposed framework for businesses and policymakers. *Journal of Privacy and Confidentiality*, jun 2011. doi: 10.29012/jpc.v3i1.596. URL https://doi.org/10.29012/jpc.v3i1.596. 2.2.1

[156] Ryan Stevens, Clint Gibler, Jon Crussell, Jeremy Erickson, and Hao Chen. Investigating user privacy in android ad libraries. In *Workshop on Mobile Security Technologies (MoST)*, volume 10. Citeseer, 2012. 5.2.1

[157] Noi Sukaviriya, Srdjan Kovacevic, James D. Foley, Brad A. Myers, Dan R. Olsen, and Matthias Schneider-Hufschmidt. Model-based user interfaces: what are they and why should we care? In *Proceedings of the 7th annual ACM symposium on User interface software and technology - UIST '94*. ACM Press, 1994. doi: 10.1145/192426.192479. URL https://doi.org/10.1145/192426.192479. 5.4.1

[158] Mohammad Tahaei, Kami Vaniea, and Naomi Saphra. Understanding privacy-related questions on stack overflow. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. ACM, apr 2020. doi: 10.1145/3313831.3376768. URL https://doi.org/10.1145/3313831.3376768. 3.6.3

[159] Mohammad Tahaei, Kami Vaniea, and Naomi Saphra. *Understanding Privacy-Related Questions on Stack Overflow*, page 1–14. Association for Computing Machinery, New York, NY, USA, 2020. ISBN 9781450367080. URL https://doi.org/10.1145/3313831.3376768. 3.6.3

[160] Mohammad Tahaei, Tianshi Li, and Kami Vaniea. Understanding privacy-related advice on stack overflow. In *Proceedings on Privacy Enhancing Technologies*, pages 1—-18, 2022. doi: 10.2478/popets-2022-0032. 1, 3, 3.6.2, 3.6.3

[161] Mohammad Tahaei, Kopo M. Ramokapane, Tianshi Li, Jason I. Hong, and Awais Rashid. Charting app developers' journey through privacy regulation features in ad networks. In *Proceedings on Privacy Enhancing Technologies*, pages 1—-24, 2022. 1, 3, 3.6.1, 6.1

[162] Joshua Tan, Khanh Nguyen, Michael Theodorides, Heidi Negrón-Arroyo, Christopher Thompson, Serge Egelman, and David Wagner. The effect of developer-specified explanations for permission requests on smartphone user behavior. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, apr 2014. doi: 10.1145/2556288.2557400. URL https://doi.org/10.1145/2556288.2557400. 5.1.1, 5.2.1

[163] National Telecommunications and Information Administration. Short form notice code of conduct to promote transparency in mobile app practices. Available at https://www.ntia.doc.gov/files/ntia/publications/july_25_code_draft.pdf (2017/05/14), 2013. 2.4.2

[164] Tyler W. Thomas, Madiha Tabassum, Bill Chu, and Heather Lipford. Security during application development. In *Proceedings of the 2018 CHI Conference on Human Factors*

in Computing Systems - CHI '18. ACM Press, 2018. doi: 10.1145/3173574.3173836. URL https://doi.org/10.1145/3173574.3173836. 2.4.2

[165] April Yi Wang, Will Epperson, Robert A DeLine, and Steven M Drucker. Diff in the loop: Supporting data comparison in exploratory data analysis. In *CHI Conference on Human Factors in Computing Systems*, pages 1–10, 2022. 6.3.1

[166] April Yi Wang, Dakuo Wang, Jaimie Drozdal, Michael Muller, Soya Park, Justin D Weisz, Xuye Liu, Lingfei Wu, and Casey Dugan. Documentation matters: Human-centered ai system to assist data science code documentation in computational notebooks. *ACM Transactions on Computer-Human Interaction*, 29(2):1–33, 2022. 6.3.1

[167] Haoyu Wang, Jason Hong, and Yao Guo. Using text mining to infer the purpose of permission use in mobile apps. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing - UbiComp '15*. ACM Press, 2015. doi: 10.1145/2750858.2805833. URL https://doi.org/10.1145/2750858.2805833. 6.1, 6.2.1

[168] Haoyu Wang, Yuanchun Li, Yao Guo, Yuvraj Agarwal, and Jason I Hong. Understanding the purpose of permission use in mobile apps. *ACM Transactions on Information Systems (TOIS)*, 35(4):1–40, 2017. 2.2.2

[169] Yang Wang, Gregory Norcie, Saranga Komanduri, Alessandro Acquisti, Pedro Giovanni Leon, and Lorrie Faith Cranor. " i regretted the minute i pressed share" a qualitative study of regrets on facebook. In *Proceedings of the seventh symposium on usable privacy and security*, pages 1–16, 2011. 1

[170] Chamila Wijayarathna, Nalin A. G. Arachchilage, and Jill Slay. A generic cognitive dimensions questionnaire to evaluate the usability of security APIs. In *Human Aspects of Information Security, Privacy and Trust*, pages 160–173. Springer International Publishing, 2017. doi: 10.1007/978-3-319-58460-7_11. URL https://doi.org/10.1007/978-3-319-58460-7_11. 2.4.1

[171] Primal Wijesekera, Joel Reardon, Irwin Reyes, Lynn Tsai, Jung-Wei Chen, Nathan Good, David Wagner, Konstantin Beznosov, and Serge Egelman. Contextualizing privacy decisions for better prediction (and protection). In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. ACM, apr 2018. doi: 10.1145/3173574.3173842. URL https://doi.org/10.1145/3173574.3173842. 5.1, 5.1.1, 5.2.1

[172] Edwin B Wilson. Probable inference, the law of succession, and statistical inference. *Journal of the American Statistical Association*, 22(158):209–212, 1927. 6.4.1

[173] Jim Witschey, Olga Zielinska, Allaire Welk, Emerson Murphy-Hill, Chris Mayhorn, and Thomas Zimmermann. Quantifying developers' adoption of security tools. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2015*. ACM Press, 2015. doi: 10.1145/2786805.2786816. URL https://doi.org/10.1145/2786805.2786816. 2.4.1, 2.4.2

[174] Shundan Xiao, Jim Witschey, and Emerson Murphy-Hill. Social influences on secure development tool adoption. In *Proceedings of the 17th ACM conference on Computer supported cooperative work & social computing - CSCW '14*. ACM Press, 2014. doi: 10.1145/

2531602.2531722. URL https://doi.org/10.1145/2531602.2531722. 2.4.1, 2.4.2

[175] Yue Xiao, Zhengyi Li, Yue Qin, Jiale Guan, Xiaolong Bai, Xiaojing Liao, and Luyi Xing. Lalaine: Measuring and characterizing non-compliance of apple privacy labels at scale. *arXiv preprint arXiv:2206.06274*, 2022. 6.1.2

[176] Jing Xie, H. R. Lipford, and Bill Chu. Why do programmers make security errors? In *2011 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, sep 2011. doi: 10.1109/vlhcc.2011.6070393. URL https://doi.org/10.1109/vlhcc.2011.6070393. 2.4.1, 2.4.2

[177] Jing Xie, Heather Lipford, and Bei-Tseng Chu. Evaluating interactive support for secure programming. In *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems - CHI '12*. ACM Press, 2012. doi: 10.1145/2207676.2208665. URL https://doi.org/10.1145/2207676.2208665. 2.4.2, 4.1

[178] Jean Yang, Kuat Yessenov, and Armando Solar-Lezama. A language for automatically enforcing privacy policies. *ACM SIGPLAN Notices*, 47(1):85, jan 2012. doi: 10.1145/2103621.2103669. URL https://doi.org/10.1145/2103621.2103669. 2.4.2

[179] Jean Yang, Travis Hance, Thomas H. Austin, Armando Solar-Lezama, Cormac Flanagan, and Stephen Chong. Precise, dynamic information flow for database-backed applications. *ACM SIGPLAN Notices*, 51(6):631–647, jun 2016. doi: 10.1145/2980983.2908098. URL https://doi.org/10.1145/2980983.2908098. 2.4.2

[180] Jinyan Zang, Krysta Dummit, James Graves, Paul Lisker, and Latanya Sweeney. Who knows what about me? a survey of behind the scenes personal data sharing to third parties by mobile apps. *Technology Science*, 30, 2015. 4.1

[181] Shikun Zhang, Yuanyuan Feng, Yaxing Yao, Lorrie Faith Cranor, and Norman Sadeh. How usable are ios app privacy labels? *UMBC Faculty Collection*, 2022. 6.1, 6.1.2

[182] Chengbo Zheng, Dakuo Wang, April Yi Wang, and Xiaojuan Ma. Telling stories from computational notebooks: Ai-assisted presentation slides creation for presenting data science work. In *CHI Conference on Human Factors in Computing Systems*, pages 1–20, 2022. 6.3.2

[183] Sebastian Zimmeck, Ziqi Wang, Lieyong Zou, Roger Iyengar, Bin Liu, Florian Schaub, Shomir Wilson, Norman Sadeh, Steven M. Bellovin, and Joel Reidenberg. Automated analysis of privacy requirements for mobile apps. In *Proceedings 2017 Network and Distributed System Security Symposium*. Internet Society, 2017. doi: 10.14722/ndss.2017.23034. URL https://doi.org/10.14722/ndss.2017.23034. 2.1.3