DASP: a framework for driving the adoption of software security practices

Larios-Vargas, E.; Elazhary, O.; Yousefi, S.; Lowlind, D.; Vliek, M.L.W.; Storey, M.A.

DASP: A Framework for Driving the Adoption of Software Security Practices

Enrique Larios-Vargas, *Member, IEEE*, Omar Elazhary, Soroush Yousefi, Derek Lowlind, Michael L. W. Vliek, and Margaret-Anne Storey, *Member, IEEE*

***Abstract*—Implementing software security practices is a critical concern in modern software development. Industry practitioners, security tool providers, and researchers have provided standard security guidelines and sophisticated security development tools to ensure a secure software development pipeline. But despite these efforts, there continues to be an increase in the number of vulnerabilities that can be exploited by malicious hackers. There is thus an urgent need to understand why developers still introduce security vulnerabilities into their applications and to understand what can be done to motivate them to write more secure code. To understand and address this problem further, we propose DASP, a framework for diagnosing and driving the adoption of software security practices among developers. DASP was conceived by combining behavioral science theories to shape a cross-sectional interview study with 28 software practitioners. Our interviews lead to a framework that consists of a comprehensive set of 33 drivers grouped into 7 higher-level categories that represent what needs to happen or change so that the adoption of software security practices occurs. Using the DASP framework, organizations can design interventions suitable for developers' specific development contexts that will motivate them to write more secure code.**

***Index Terms*—Behavior change, developer-centric security, software security, software security practices.**

I. Introduction

SOFTWARE security is undeniably one of the most critical and ongoing concerns in modern software development. In 2021, the NIST Computer Security Division[1] identified over 18,000 software vulnerabilities, and this number has been steadily increasing from 2016 [1]. Flawed applications might behave unpredictably, and these weaknesses are often abused by malicious hackers [2]. For example, recent reports show that malicious attackers use unique platforms and search engines, e.g., Shodan[2], to scan for networks that are exposed to known vulnerabilities and exploit them before a victim can apply a patch [3].

Software developers are responsible for many of these vulnerabilities. Notably, around 60% of vulnerabilities identified by Veracode[3] in a study of 130,000 active applications highlighted that a developer's lack of careful development and maintenance was a significant reason for the introduction of vulnerabilities. These vulnerabilities occur because developers face pressure to meet customer requirements and deliver features quickly. In addition, developers often treat security as a non-functional requirement that is less critical than delivering features, unless employers or application users impose security compliance [52]. The delayed consideration of security issues makes it more challenging and even more expensive to address in later stages [50].

Neglecting software security is a well-recognized problem in any industry. As a result, there are many ongoing efforts to fix it by leading cybersecurity organizations of different business types, such as the MITRE corporation[4], OWASP[5], the CERT Division[6], and NIST[7]. These organizations provide security standards and excellent resources to help practitioners ensure a secure software product. There are also hundreds of free online resources available for practitioners to learn software security practices. Moreover, there is a huge active community of security professionals and researchers behind the development of security tools and keeping up-to-date security guidelines aimed at ensuring a secure software development life cycle. On the flip side, several research studies have exposed the need to investigate the behavioral aspects of security adoption, particularly developers' motivations and attitudes towards security [44], [47], [53]. However, despite the availability of these many resources, developers continue to introduce security vulnerabilities in source code, and organizations lack proper guidelines for designing strategies to mitigate poor security. Furthermore, we still lack an understanding of what drives developers to adopt software security practices from the behavioral science perspective.

This situation pushes forward the need to properly understand developer behaviors—specifically, what drives developers to

Manuscript received 4 May 2022; revised 31 October 2022; accepted 14 December 2022. Date of publication 10 January 2023; date of current version 18 April 2023. Recommended for acceptance by F. Ferrucci. *(Corresponding author: Enrique Larios-Vargas.)*

This work involved human subjects or animals in its research. Approval of all ethical and experimental procedures and protocols was granted by the UVic Human Research Ethics Board under Application No. 21-0122, and performed in line with the Human Research Ethics Board Application for Ethics Approval for Human Participant Research.

Enrique Larios-Vargas, Omar Elazhary, Soroush Yousefi, Derek Lowlind, and Margaret-Anne Storey are with the Department of Computer Science, University of Victoria, Victoria, BC V8P 5C2, Canada (e-mail: elariosvargas@uvic.ca; omazhary@gmail.com; soroush.ysf@gmail.com; dereklowlind@gmail.com).

Michael L. W. Vliek is with the Leiden University, 2311, EZ Leiden, Netherlands (e-mail: m.l.w.vliek@fsw.leidenuniv.nl).

Digital Object Identifier 10.1109/TSE.2023.3235684

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

[2]https://www.shodan.io/

[3]https://info.veracode.com/report-state-of-software-security-volume-11

[4]https://attack.mitre.org/

[5]https://owasp.org/

[6]https://www.sei.cmu.edu/about/divisions/cert/

[7]https://www.nist.gov/

adopt software security practices. With this knowledge, organizations would be better positioned to design interventions to foster behavior change, leading developers to write more secure code. To understand developer behaviour towards the adoption of security practices, we conducted a cross-sectional interview study with a cohort of 28 software practitioners and used the COM-B Model for behavior change [11] as a diagnostic tool to understand the capabilities, opportunities, and motivations behind the adoption of software security practices. Through our study, our research provided answers to the following research question:

> RQ: What needs to **happen** or **change** so the **adoption** of **software security practices** by **developers** occurs?

The insights from our study have led to a novel actionable framework that organizations and developers can use to drive the adoption of software security practices. The framework captures 33 drivers[8] across seven categories of behavior change. In addition, we exhaustively compared our findings with current literature, noticing that previous research did not report three of our drivers. Our framework can be used by organizations to diagnose security challenges and to design strategies that influence the adoption of security practices within their specific context. This is the first study that considers behavior change in software security, opening the door for future research. Our framework can support future researchers by leveraging the power of well-known behavioral psychology theories to understand and drive improvement in secure software development.

The following section (Section II) provides some background on the behavioral theories that shaped the design of our study. Then we describe the cross-sectional interview methodology and our analysis approach in Section III. We present our findings in Section IV, and review related work in Section V. Next, we discuss how organizations can use our framework in Section VI and provide implications for developers, security specialists, and researchers. In Section VII, we detail the threats to the validity of our research and conclude the paper in Section VIII.

II. Background: Theoretical Foundations

Understanding human behavior is challenging due to the diverse number of psychological factors that may impact positive and negative change. As a result, behavioral science disciplines, such as psychology and behavioral economics, offer many theories and models that describe drivers of behavior, such as attitudes, motivations, norms, habits, and behavioral control. Many of these theories and models can be further used to predict human decision-making and behaviors (e.g.,[5], [6], [7], [8], [9], [10]). Similarly, it is a challenge to understand what drives change in human behaviors in software development activities such as software security, but doing so may help organizations understand how to improve the adoption of practices that will encourage developers to write more secure code. In the following, we describe three behavioral theories and models used to study behavior change. Later we discuss how we used these theories to design our novel study of the potential for behavior change in software security.

[8]A driver is a factor that affects the adoption of security practices by developers.

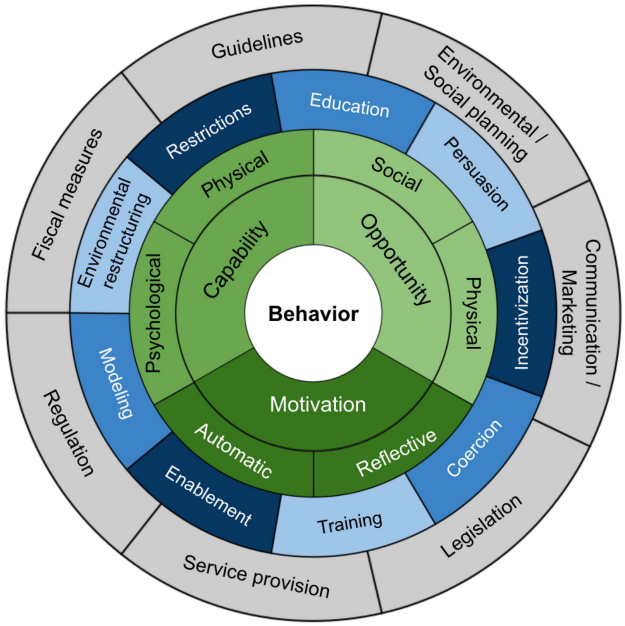

Fig. 1. The Behavior Change Wheel: The green layer refers to the COM-B model (sources of behavior), the blue layer represents the intervention functions, and the grey layer refers to the policy categories.

A. Behavioral Theories and Models

We present three behavioral theories and models: (1) the COM-B model and the Behavior Change Wheel, (2) the Self-Efficacy Theory, and (3) the Response efficacy concept.

1) The COM-B Model and the Behavior Change Wheel: Michie et al. proposed *the Behavior Change Wheel* (BCW) as a synthesis of 19 different theories and models of behavior change identified in a systematic literature review [11]. Some of these frameworks suggest that behavior is primarily driven by beliefs and perceptions, while others significantly emphasize unconscious biases or the social environment [12]. These drivers of behavior are undeniably relevant, but they were not integrated coherently. The BCW was conceived by combining the common features of these theories and linking them to a behavioral model that is sufficiently broad and applicable to any setting [12]. Fig. 1 shows the BCW that aims to incorporate the standard features of all these frameworks and link them to a model of behavior.

At the center of the BCW resides the COM-B model, shown in Fig. 2, which is composed of three vital conditions: **capability** (C), **opportunity** (O), **motivation** (M), and **behavior** (B). The COM-B model provides a clear starting point to understand behavior in a specific context, and guides the design and development of interventions. For example, according to the model, *to engage in a particular behavior, someone must be* **physically and psychologically capable**, *have the* **social and physical opportunity**, *and be* **motivated***to perform the target* **behavior***more than any other competing behaviors*. In addition, the model presents motivation from an automatic (habits) and reflective (rational intentions) perspective.

TABLE I
INTERVENTION FUNCTION DEFINITIONS FROM THE BCW

Intervention function	Definition
Education	Increasing knowledge or understanding
Persuasion	Using communication to induce positive or negative feelings or stimulate action
Incentivization	Creating an expectation of reward
Coercion	Creating an expectation of punishment or cost
Training	Imparting skills
Restriction	Using rules to reduce the opportunity to engage in the target behaviour (or to increase the target behaviour by reducing the opportunity to engage in competing behaviours)
Environmental restructuring	Changing the physical or social context
Modeling	Providing an example for people to aspire to or imitate
Enablement	Increasing means/reducing barriers to increase capability (beyond education and training) or opportunity (beyond environmental restructuring)

TABLE II
POLICY CATEGORY DEFINITIONS FROM THE BCW

Policy categories	Definition
Communication/marketing	Using print, electronic, telephonic, or broadcast media
Guidelines	Creating documents that recommend or mandate practice. This includes all changes to service provision
Fiscal measures	Using the tax system to reduce or increase the financial cost
Regulation	Establishing rules or principles of behaviour or practice
Legislation	Making or changing laws
Environmental/social planning	Designing and/or controlling the physical or social environment
Service provision	Delivering a service

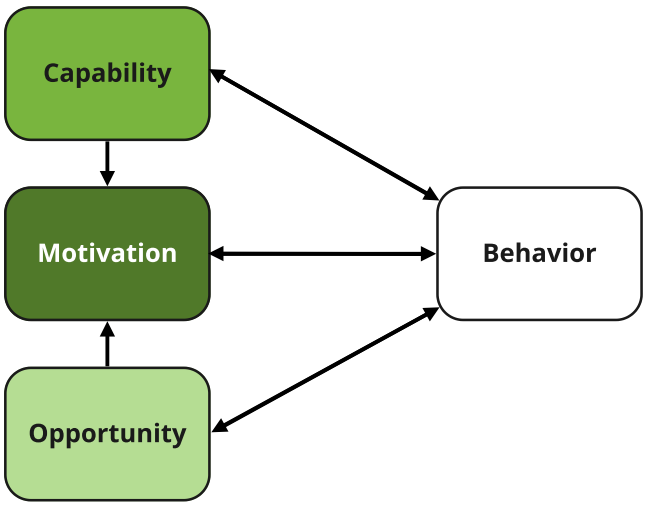

Fig. 2. The COM-B model: To engage in a particular behavior, someone must have the capabilities, have the opportunity, and feel motivated to perform the target behavior more than any other competing behaviors.

Fig. 1 depicts the COM-B model as the hub of the BCW; it identifies and explains the sources of the behavior, in other words, *what needs to happen or change so the target behavior occurs*. Surrounding the COM-B model, two extra layers represent 9 intervention functions that will help address any issue identified in any of the COM-B model components[9] (capability, opportunity, or motivation). Subsequently, the external layer comprises 7 policy categories that organizations can use to deliver the intervention functions. For more details, Table I and Table II provide the landscape of intervention functions and policy categories, including their respective definitions.

The COM-B model and the BCW have been applied in several settings, from understanding behavior change by individuals, to groups, sub-populations, and populations, and within different organizations and systems [35]. For instance, Barker et al. propose a successful application of the COM-B model and the BCW to develop an intervention to promote regular, long-term use of hearing aids by adults with acquired hearing loss [13]. When applying the model, the investigation exposes that behavioral planning for hearing-aid use on the side of the audiologists should be part of the routine audiological practice, which requires a complex intervention that addresses psychological, capability, physical, and social opportunity, and reflective and automatic motivation.

2) Self-Efficacy Theory: The Self-Efficacy Theory (SET) is an essential contribution from social cognitive theory to understand individuals' behaviors based on a self-evaluation of their abilities [38]. Bandura proposed the SET and defined it as people's beliefs about their capabilities to produce designated levels of performance that exercise influence over events that affect their lives [37]. Therefore, high levels of self-efficacy reinforce people's convictions about their abilities to perform a task successfully [39].

The SET introduces four significant sources of efficacy beliefs: *mastery experiences*, *vicarious experiences*, *verbal*

[9]Components refer to the main elements of the COM-B model: Capability, Opportunity, and Motivation.

persuasion, and *emotional and physiological states* [37]. First, individual self-efficacy is boosted by having success or direct mastery experience. Additionally, observing people around us having successful experiences, especially individuals sharing similar characteristics or backgrounds, increases our beliefs that we can also achieve success by mastering the required activities. Moreover, influential people around us encourage us and raise our beliefs that we can succeed by mastering certain activities. Finally, individuals holding positive emotions are more likely to have confidence in their skills to successfully perform particular activities.

3) Response Efficacy: The Response-Efficacy Concept (REC) has its origins in Bandura's social cognitive theory [38]. Bandura used the term *outcome expectancies* to refer to beliefs about the consequences of performing a behavior, which is the foundation of REC. Response-efficacy is defined as one's belief that acting in a specific manner is likely to mitigate threats, which is why it is generally adopted in research on fear and fear appeals [40], [41]. *Outcome expectancies* are somewhat broader and form an essential part of people's beliefs about an attitude-object (e.g., a product, event, or behavior). The construct has its origin in Fishbein and Ajzen's expectancy-value theory [6] and is captured under behavioral beliefs (leading to attitudes) in the theory of planned behavior [42]. People generally develop favorable attitudes toward behaviors they believe lead to desirable consequences and form unfavorable attitudes toward behaviors they believe lead to undesirable consequences. *Generalized outcome expectancies* are employed under traits such as optimism, where people generally hold that the future will turn out positively, which does not necessarily include actual behavior [43].

B. Combining the Three Behavioral Models in the Context of Software Security

To study why developers fail to adopt software security practices, we used the COM-B model as a practical diagnosis tool to highlight the capabilities, opportunities, and motivations that potentially influence developers to adopt software security practices. Additionally, we used the Self-Efficacy Theory (SET) and Response-Efficacy Concept (REC) to complement and enrich the diagnosis: SET assisted with understanding developers' beliefs regarding their capabilities and their confidence for performing software security practices (and what can influence that confidence), and REC aided in understanding how developers' perceived success in adopting security practices affects their security adoption behaviors.

The COM-B model was designed as a generic diagnosis tool. To use it in software security, we needed to adapt some of its components. The COM-B model highlights the *capabilities* in terms of *psychological* and *physical* features. *Psychological capabilities* refer to being aware of the knowledge required to perform the behavior. *Physical capabilities* represent having the physical skills to conduct the behavior, for instance, having more physical strength and overcoming physical limitations. In the context of software security, since physical capabilities are not an essential component, they turned into having the *technical* and *non-technical* skills to adopt software security practices.

One aspect of the COM-B model emphasizes *social* and *physical* opportunities. On the one side, *social opportunities* refer to having the opportunity afforded by interpersonal influences or social and cultural norms within the organization. On the other side, *physical opportunity* implies having the resources required to perform the behavior. The term *physical* turned into *technical* opportunities to make it more explicit to the software security context. But, keeping the exact definition and including resources such as tools, time, and money. Finally, concerning *motivations*, we used the same terminology and definitions proposed by the COM-B model. Table III [10] provides examples and more detail of each component definition.

C. Using the BCW Approach

In the following section, we describe the three stages of the Behavior Change Wheel as illustrated in Fig. 3.

Stage 1: Behavioral scientists at this stage aim to identify the behavioral problem and select the target behavior. In other words, they identify the motivations for designing a strategy or intervention and what is needed to address the problem. The COM-B model provides a framework to understand what needs to happen or change so that a desired behavior occurs. In the end, the output of this stage is to identify a set of drivers that influence a target behavior.

Stage 2: The goal of this stage is to identify the most appropriate intervention functions and then select their respective policy categories, which will serve as the foundation to design and deliver an intervention.

To select the most relevant intervention functions, organizations should use the set of drivers identified in *Stage 1* to shift from diagnosis to intervention. Organizations need to build a matrix using the COM-B model components and the nine intervention functions proposed by the BCW. As detailed in Section II, the COM-B model components to take into account are *Technical Capabilities, Non-technical Capabilities, Psychological Capabilities, Technical Opportunities, Social Opportunities, Reflective Motivations, and Automatic Motivations*, and the intervention functions to consider are *Education, Persuasion, Incentivisation, Coercion, Training, Restriction, Environmental restructuring, Modeling, and Enablement*. By using this matrix, organizations should be able to identify the intervention functions that is relevant to them. For example, our study identified a strong relationship between *Training* and *Psychological capabilities*. Developers perceived that having tailored security training is crucial for building confidence in implementing software security practices.

The next step in developing an intervention strategy is to select the policy categories that can help deliver and implement the intervention. The BCW indicates which policy categories are the most suitable in supporting each intervention function. Following these indications, organizations should select the most

[10]To clarify the definitions of the COM-B model components in the context of software security, the descriptions and examples in Table III emerged from our findings.

TABLE III
COM-B MODEL COMPONENTS AND EXAMPLES IN THE CONTEXT OF SOFTWARE SECURITY

COM-B model component	Description	Examples
Technical capability	Having the technical skills to perform security practices	Having the skill to understand technical aspects of security exploits and apply security patches
Non-Technical capability	Having the non-technical skills to perform security practices	Having the skill to communicate and discuss security issues with all stakeholders compromised in a security incident
Psychological capability	Being aware of the negative consequences of not adopting security practices Having the knowledge or confidence to apply security practices	Awareness of the compliance standards that regulate data protection and privacy, such as GDPR, PCI, and HIPPA
Technical opportunity	Having the opportunity afforded by the environment involving time, resources, tools, and locations	Having tools that can be easily integrated into the development workflow and alert developers of potential risks
Social opportunity	Having the opportunity afforded by interpersonal influences, social and cultural norms that influence the way developers think about things	Having people around implementing security practices reminds developers why to invest extra effort in security
Reflective motivation	Reflective processes involving plans (self-conscious intentions) and evaluations (beliefs about what is good and bad)	Intending to follow security guidelines after understanding their value and the rationale behind them
Automatic motivation	Automatic processes involving emotional reactions, desires, (wants and needs), impulses, inhibitions, drive states, and reflex responses	Developers feeling frustrated due to the lack of support from management when prioritizing security over other tasks

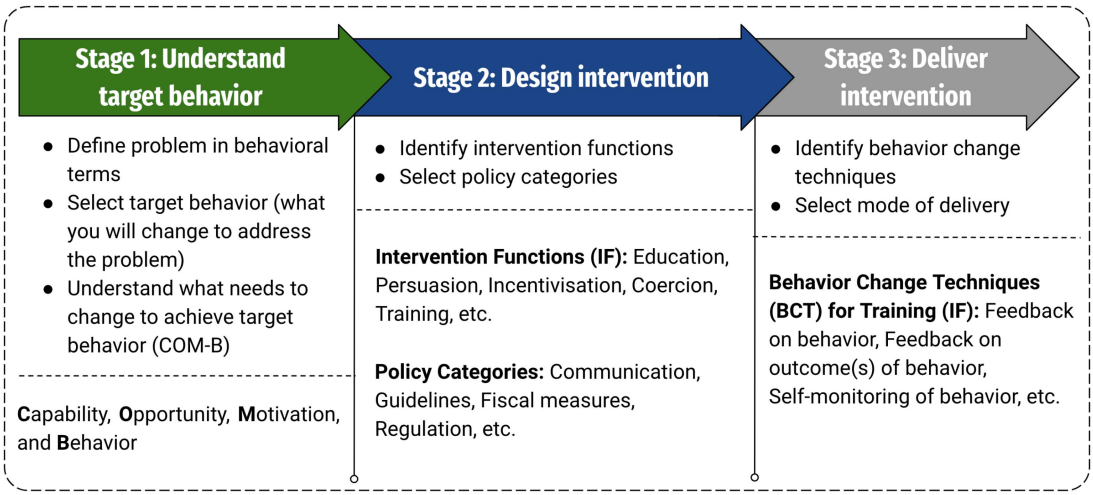

Fig. 3. The Three Stages of the Behavior Change Wheel including some intervention functions, policy categories, and behavior change techniques, as indicated in Fig. 1.

appropriate policy categories associated with their intervention functions. For example, concerning *Training*, some essential policy categories are *Guidelines, Fiscal measures, Regulation, Legislation, and Service provision*. A complete list of the seven policy categories and their definitions is detailed in Table II.

Stage 3: The goal of this stage is to determine which Behavior Change Techniques (BCTs) can support the delivery of the identified intervention functions under the relevant policy categories. Michie et al. [12] defines a BCT as an active component of an intervention designed to change behavior. BCTs are also observable, replicable, and irreducible components of an intervention. Michie et al. [36] introduced a taxonomy of 93 BCTs as a method for specifying interventions. With this knowledge, organizations may select the most frequently used BCTs for their particular intervention functions. For example, in the case of the *Training* intervention function, some of the most frequently used BCTs are *Feedback on behavior, Feedback on outcome(s) of behavior, Monitoring of behavior by others without evidence of feedback, Monitoring of outcome of behavior by others without evidence of feedback*, and *Self-monitoring of behavior.* For example, a description of the content of an intervention related to our study would be "Organizations should set the goal of *training developers* to identify potential risks in their source code, *enabling* them to understand security standards, associate those standards with threats and compliant code examples."

D. The Usefulness of the BCW

The COM-B framework helps one apply behavior change science in a systematic way that optimizes the effectiveness of behavior change interventions [19]. Since its inception in 2011, COM-B and the Behaviour Change Wheel (BCW) have been widely used to explore barriers and facilitators of behavior, identify relevant behavior change techniques, and systematically design and develop behavioral interventions, most notably in the health domain. For instance, it has been used to develop

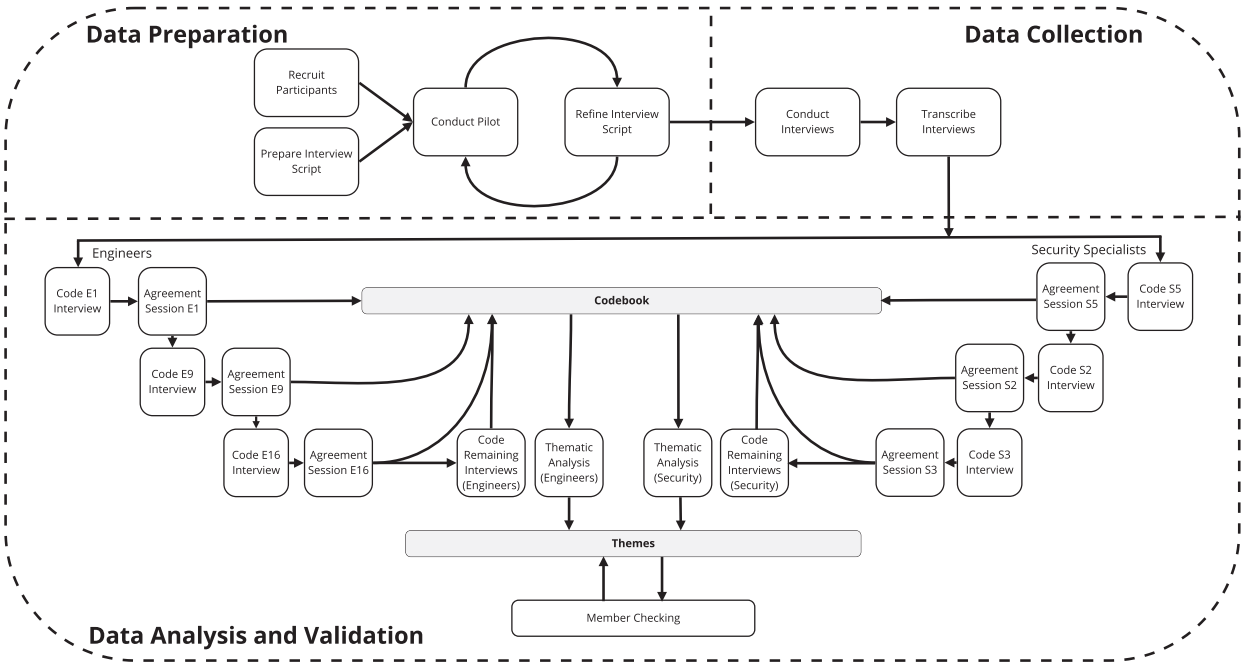

Fig. 4. Research Method: Three main stages, Data Preparation, Data Collection, and Data Analysis and Validation.

effective interventions aimed at the prevention of work-related diseases [17], enhance medical attendance in the context of smoking cessation [16], and increase the frequency of physical activity [20]. It is also a valuable aid for the systematic implementation of interventions, for example, for encouraging sustainable energy consumption [21], helping improve Open Science behaviors [18], and assisting teams in health organizations to implement new models of care [15]. Recently, Alshaikh and colleagues [14] demonstrated the usefulness of the BCW in the analysis and selection of suitable strategies to develop effective information Security Education, Training and Awareness (SETA) programs.

III. Research Method

Our study goal is to understand *what needs to happen or change so the adoption of software security practices by developers occurs*. To that aim, our research methodology consists of a cross-sectional interview study [22]. Interview-based research is suitable for gathering thoroughly detailed participant experiences and stories [22]. Our study design includes the following three stages: (a) Study preparation, (b) data collection, and (c) data analysis and results validation. Our methodology is depicted in Fig. 4. In the following sections, we explain our methodology in detail. Private information from participants and companies has been anonymized. The authors do not have the participants' authorization to make the raw interview scripts available as they contain confidential information.

A. Study Preparation

Our first step was to identify potential participants for the study and define the selection criteria at this stage. Subsequently, we designed the interview script considering demographics and the different behavioral science theories we wanted to explore to understand developers' behaviors. Finally, we conducted pilots to ensure questions' comprehensibility, ensuring that sufficient time is allowed for the interviewer to conduct the interview, and reducing interviewees' cognitive load.

Participants recruitment. Our selection criteria was to recruit participants who had at least two years of professional work experience. To thoroughly understand why software security is still neglected by developers, it is relevant to take into account the perspectives of the main stakeholders involved in the adoption of security practices within an organization. Consequently, our pool of participants included two different roles, engineers and security specialists. The engineers' group consisted of software developers, tech leads, DevOps engineers, and CTOs. The security specialists' group consisted of application security specialists, pen testers, security program managers, security developers, etc.

The pool of interviewees from the Engineers' group came from convenience sampling [26]. The authors of this paper invited their industry contacts to participate in the study. The first author of this paper sent 42 email invitations, receiving 24 satisfactory responses to join the study. The first 5 participants were considered for piloting the interviews and refining the interview questions. Additionally, the participants from the security specialists group came from purposive sampling [26]. The first author identified and invited software security experts using the LinkedIn platform. We sent 22 invitations, resulting in 9 of the specialists agreeing to join the study.

In the end, our final pool of interviewees included 28 practitioners, 19 engineers (identified as E1-E19 throughout this paper), and 9 security professionals (identified as S1-S9). Our participants came from 25 different companies that work in 16 diverse industry types. The professional experience of our

TABLE IV
PROFILE OF OUR PARTICIPANTS (N=28)

MC	Interviewee	Company	Business	Company size	Region	Role/Function	Years of Exp.
	E1	C1	E-commerce	50+	North America	Software Developer	6
	E2	C2	E-commerce	1000+	Europe	Software Developer	2
*	E3	C3	Telecommunications	5000+	South America	Software Developer	7
	E4	C4	ERP Systems	10000+	North America	Senior Software Developer	9
	E5	C5	Beauty and Personal Care	10+	North America	Development Team Lead	20
*	E6	C6	Embedded Systems	10000+	Europe	Scientific Software Developer	2
*	E7	C7	Education	5000+	Europe	Lecturer/Software Engineer	2
*	E8	C8	Health Services	10000+	North America	Senior Software Developer	20
*	E9	C9	CRM Systems	5000+	Europe	Principal Software Engineer	7
	E10	C10	Booking and Rental Services	100+	North America	Development Team Lead	16
*	E11	C11	Professional Design	1000+	Oceania	Software Developer	9
	E12	C4	ERP Systems	10000+	North America	Software Developer	5
*	E13	C7	Research	5000+	Europe	Scientific Software Engineer	2
	E14	C12	CRM Systems	100+	South America	Software Developer	3
	E15	C13	Health Services	1000+	Europe	Software Developer	7
	E16	C14	Video Games	100+	North America	Senior DevOps Engineer	18
	E17	C15	Embedded Systems	1000+	Europe	Software Engineer	4
	E18	C16	Telecommunications	50+	North America	VP Systems Enginnering	10
*	E19	C17	Digital Publishing	100+	North America	CTO	32
*	S1	C18	Health Care Services	10000+	North America	Technical Product Manager	15
	S2	C19	Security	50+	North America	Application Security Engineer	20
	S3	C20	Financial Services	1000+	Europe	Application Security Engineer	4
	S4	C21	Security	1000+	Oceania	Senior Security Developer Advocate	15
*	S5	C22	Security	10+	North America	CTO/Pen tester	15
*	S6	C23	Security	5000+	North America	Information Security Specialist	10
	S7	C24	Telecommunications	10000+	North America	Application Security Specialist	20
*	S8	C11	Professional Design	1000+	Oceania	Software and Security Engineer	12
	S9	C25	IT and Software	10000+	North America	Security Program Manager	6

Companies are anonymized. MC column denotes participants who joined a member-checking session. (E1-E19) Engineering (Developer, Tech lead, Dev Ops engineer), (S1-S9) Security Specialist.

interviewees ranged from 2 years to 32 years, having a median of 9 years of work experience. In Table IV, we provide more details about our participants.

Interview Script Preparation. We designed semi-structured interviews to collect our participants' experiences, stories, and challenges. Semi-structured interviews foster interviewees to freely share their experiences, enabling interviewers to explore new ideas based on the participant's answers [27]. Our interview script design was guided by the COM-B model and supplemented with insights from SET and RET. A complete list of the interview questions is available in our online appendix [58]. The overall structure of our interview script included the following topics:

a) the practitioners' demographics and context to understand essential aspects of the environment where software security practices' adoption (or not adoption) occurs.
b) how confident participants feel about their ability to perform specific software security tasks (self-efficacy).
c) to what extent do participants feel their adoption of software security practices impact the overall adoption in their organizations or professional network (Response efficacy Theory), and
d) participants' capabilities, opportunities, and motivations for adopting software security practices (COM-B model).

Pilot Interviews and Interview Script Refinement. Since our study goal is focused on understanding the adoption of software security practices by developers, we conducted five pilot interviews with developers aiming to increase the comprehensibility of the interview questions and reduce participants' cognitive load during the interview. As a result, the 115 min initial interview duration was considerably reduced in each iteration, resulting in 60 minutes approximately. Two researchers were always involved during the pilots, the first author collaborating with one of the other researchers. The interviewers also asked participants their feedback regarding the questions' understandability and suggestions regarding the overall study. After each pilot interview, both researchers discussed the feedback collected and introduced the respective adjustments. After the fifth interview, the researchers agreed that the questions were mature enough. During the pilot, our interview participants pointed out that we should provide a standard definition of software security practices to avoid any potential misunderstanding at the beginning of the interviews. Additionally, our pilot participants highlighted that the interview duration should be reduced considerably, forcing us to select the most relevant questions for understanding the adoption of security practices.

B. Data Collection

The authors conducted 28 semi-structured interviews over the Zoom platform from May 31st to July 16th in 2021. Two researchers were involved during the interviews, one researcher leading the interview, actively asking questions and interacting with the interviewee, and the other researcher noting down relevant aspects of the participant's story and experience. We started each interview by going through our base set of questions and slightly adapting them based on the participant's role and context. For instance, we focused our questions for security professionals on their last interaction or collaboration with the engineering team instead of their personal experience

as developers, which was not applicable in most cases. In addition, immediately after each interview, both researchers discussed any potential misinterpretation based on the notes taken. We conducted interviews until the researchers agreed that we had achieved theoretical saturation. According to Strauss and Corbin [32], sampling should be discontinued once the data gathered no longer provides new information. This situation occurred after conducting 19 interviews with engineers from the engineering group and 9 interviews with participants from the security specialists group. Each interview lasted between 55 min and 75 min, and with the participant's permission, it was recorded, producing around 32 hours of recorded audio. Subsequently, recorded audios were transcribed, anonymized, and prepared for analysis.

C. Data Analysis and Findings Validation

The next stage was conducting thematic analysis to identify themes and patterns in our qualitative data [28]. Our data analysis consisted of inductively developing codes[11] from the transcripts and identifying themes associated with participants' adoption (or not adoption) of software security practices. We divided our data analysis into two steps. First, we analyzed data coming from the engineers' group; then, we analyzed the data from the security specialists' group. In this way, coders are not switching criteria and perspectives while analyzing the data. In addition, we followed an open coding approach [30]; during the open coding process, codes emerged and were removed or merged depending on the researchers' discussions. The discussions helped reduce any potential bias introduced by the coders.

At the beginning of the coding phase, the first two authors coded every excerpt of the same transcript. An excerpt represents a "dialog segment" used as a unit of analysis associated with an interviewees' response to a question. The coding phase started with the engineers' group transcripts (E1). Both researchers coded transcript E1 independently and then calibrated their understandings of the codes in an agreement session. We defined coder agreement as an excerpt where both researchers had at least one code for the excerpt in common. Subsequently, both researchers selected another participant and continued this process iteratively. Following E1, the subsequent transcripts coded were E9 and E16, obtaining an inter-rater agreement level of 54%, 74%, and 80% respectively in each iteration. Our agreement sessions involved extensive discussions on the meaning and use of the codes in our codebook, resulting in a first consensual version of our codebook. After both researchers understood each code, they started coding the remaining transcripts independently.

To reach our study goal, we developed themes based on thematic synthesis [28] of our coded data. We conducted 5 thematic analysis sessions with the engineers' group data, each including a different subset of participants. During the thematic analysis sessions, similarities and differences between codes were discussed and then grouped in higher-level themes. Subsequently, each theme was associated with one or more of the three components of the COM-B model (Capabilities, Opportunities, and Motivations). To validate the correct interpretation of the codes and themes creation, we counted with the feedback of an expert reviewer, the 6th author of this paper, who helped us reduce any bias the researchers might introduce in the analysis. Additionally, to validate the association between themes and COM-B model components, we had the assistance of a domain expert in behavioral psychology, the 5th author of this paper.

Once the thematic analysis of the engineers' group finished, we started the open coding process for the security specialists' group. We followed a similar approach to the engineers' group. The coding phase in this group started with the first two authors coding transcript S5, then calibrating their understanding of the codes in an agreement session. Following S5, the subsequent transcripts coded were S2 and S3, reaching an average agreement of 50%, 74%, and 77% respectively in each iteration. During the agreement sessions conducted in each iteration, the researchers noted that there was no need to add new codes to the initial codebook obtained from coding the engineers' group. Consequently, the researchers conducted two thematic analysis sessions. As a result, we added three new themes to the initial set of themes coming from the engineers' group. Similarly, in this stage, we counted with the support of an expert reviewer and a behavioral psychology expert to ensure the correct codes interpretation, themes creation, and association with the COM-B model components.

After thematic analysis, we performed member-checking sessions [31] with the study participants. We sent email invitations to all participants to join a member checking meeting to verify whether our findings are significant to them and the context of their organizations. 12 participants accepted to join, 8 engineers and 4 security specialists. The member checking sessions were conducted individually, each one lasting around 45 min, and consisted of the following five steps: (1) the first author presenting the motivation and study participants' demographic data, (2) explicitly indicating what we want to collect in the session, (3) presenting the findings (drivers), (4) for each driver presented, discussing to what extent the driver is essential to them and their organizations, and (5) asking for any potential driver that we overlooked in our findings. We used all participants' feedback to validate our findings and ensure the interpretation of qualitative data collected from the interviews was correct. The member-checking presentation slides and a table summarizing the feedback collected from our participants are available in our online appendix [58].

IV. Findings

We used the COM-B model as a diagnosis tool to understand **what needs to happen or change so that developers adopt software security practices**. Based on the codes that emerged from the analysis of the interview transcripts, we associated all the excerpts from our participants' stories with the different components of the COM-B-model, *capabilities*, *opportunities*, and *motivations*. This resulted in 33 drivers that we then grouped into seven categories based on similarity: Fig. 5 shows the list of the seven categories that emerged from our study.

[11] The terms "codes", "themes", and "categories" are used in this manuscript following the definitions provided by G. Gibbs [29].

Fig. 5. *Findings*: The *seven considerations* (categories) that *organizations* and *stakeholders* should pay attention to foster the adoption of software security practices by *developers*.

TABLE V
THE 33 DRIVERS THAT EMERGED FROM OUR STUDY, (E1-E19) ENGINEERING (DEVELOPER, TECH LEAD, DEV OPS ENGINEER), (S1-S9) SECURITY SPECIALIST

		E1	E2	E3	E4	E5	E6	E7	E8	E9	E10	E11	E12	E13	E14	E15	E16	E17	E18	E19	S1	S2	S3	S4	S5	S6	S7	S8	S9	E	S	Totals
Building an organizational security culture																																
D1	Organization promoting/mandating security	•	•	•	•	•		•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	18	9	27
D2	Prioritizing security practices		•	•		•		•	•	•		•	•	•	•	•	•	•	•		•	•	•	•	•	•	•	•	•	14	9	23
D3	Having a security-specific role filled	•	•						•	•	•	•	•		•	•	•		•		•	•	•		•	•	•			11	7	18
D4	Overcoming the resistance to change				•						•								•	•		•			•					4	2	6
D5	Fostering collaboration between engineering and security teams	•	•		•			•	•	•	•	•	•		•	•	•	•			•		•	•		•	•	•	•	13	7	20
D6	Awareness of the social perception of security adoption in one's own organization and professional network	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	19	9	28
D7	Providing awareness of external incentives and compliance		•			•	•									•								•	•	•	•	•	4	5	9
Facilitating the adoption of software security by developers																																
D8	Shaping developer's attitude towards security	•	•	•	•	•					•	•	•				•	•		•	•	•		•	•	•	•	•	•	11	7	18
D9	Tool awareness	•	•	•		•		•		•	•			•						•	•	•	•	•	•			•	•	9	7	16
D10	Standard guidelines geared at developers			•			•		•	•	•		•			•	•				•	•	•	•	•					8	5	13
D11	Reduction of system complexity	•	•			•	•					•			•	•			•			•								8	1	9
Understanding risks, benefits, and trade-offs																																
D12	Awareness of potential risks and security incidents	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•		•	•		•	•	•	•	19	7	26
D13	Learning from actual incidents							•			•	•		•	•			•			•			•						6	3	9
D14	Fear of non-adoption consequences					•	•	•			•	•	•	•	•	•			•	•	•	•		•	•	•	•	•	•	12	7	19
D15	Knowledge of benefits			•	•	•	•	•	•		•	•	•	•	•	•			•	•	•	•	•	•	•	•	•	•	•	15	9	24
Providing contextual information to motivate developers to write secure code																																
D16	Promoting a customer satisfaction/protection mindset									•			•			•		•			•			•	•					4	3	7
D17	Awareness of the influential role of the industry type in developers' disposition towards security compliance	•			•					•	•		•	•	•	•	•			•				•	•		•	•	10	4	14
D18	Awareness of developers' perceptions of the need for software security based on application characteristics			•	•		•	•	•		•	•	•		•	•	•	•	•					•	•		•	•	•	13	6	19
D19	Aligning the perspective of what "good enough" security means		•	•		•	•	•	•	•	•	•	•	•		•	•	•	•	•				•	•	•	•	•	•	16	7	23
Providing justification for necessary tools and process constraints																																
D20	Consideration of tool constraints on developers' autonomy	•	•		•					•			•	•	•	•		•						•				•	9	2	11
D21	Awareness of developers' perception of security-imposed restrictions	•		•	•	•	•			•		•	•	•	•	•	•		•					•	•	•	•	•	•	13	6	19
Providing (cognitive) support to developers for writing secure code																																
D22	Availability of reminders, i.e., checklists, dashboards, etc.		•	•	•	•	•	•			•	•		•	•		•		•	•	•				•	•	•	•	13	5	18
D23	Improving the usability (complexity reduction) and accuracy of security tools	•	•							•		•	•	•																6	0	6
D24	Reducing the effort required to learn or apply security	•	•	•	•	•	•	•		•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	18	9	27
D25	Integrating tools into the development workflow											•		•	•	•			•		•		•	•			•	•	•	5	6	11
Facilitating developers' acquisition of security-specific skills																																
D26	Accessibility to learning resources	•	•	•	•	•	•			•	•	•	•		•		•		•			•	•	•	•	•			•	13	6	19
D27	Using security practices as learning tools	•		•						•	•		•		•							•	•	•						6	3	9
D28	Providing security education	•	•		•			•	•	•	•	•	•	•		•			•	•	•	•	•		•	•	•	•	•	15	8	23
D29	Fostering hands-on learning/self-learning/osmosis	•	•	•		•		•			•	•	•		•				•			•	•		•	•	•	•	•	10	7	17
D30	Creating and participating in communities of practice	•	•	•	•		•		•	•	•	•	•						•	•	•	•	•	•	•	•	•	•	•	11	9	20
D31	Having non-technical skills	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	19	9	28
D32	Confidence in their technical abilities	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•		•			•	•	•	•	19	6	25
D33	Awareness of necessary security skills	•					•	•		•		•	•	•	•	•		•	•		•	•	•	•	•	•	•	•	•	11	9	20

In Table V, we detail the entire landscape of drivers from our analysis. In the following sections, we describe the seven categories in detail using the components of the COM-B model associated with them, and the role of these drivers in the overall adoption of software security practices. To facilitate the description of each driver, we use the term practitioner when referring to both engineers and security specialists. As described in Section III, the term *engineers* includes *developers, tech lead, DevOps engineers, and CTOs*, and *security specialists* consists of *application security engineers, pentesters, security developers*.

A. Building an Organizational Security Culture

Practitioners consider culture an essential driver to boost the adoption of security practices in an organization. Organizations that promote or mandate security usually exhibit a good security posture by having a role such as a security champion or a security team in the company. Fig. 6 shows the relationships between the COM-B model components and drivers.

D1: Organization Promoting/Mandating Security. Organizations should carefully pay attention to their influential role in developers' adoption of software security practices. We used

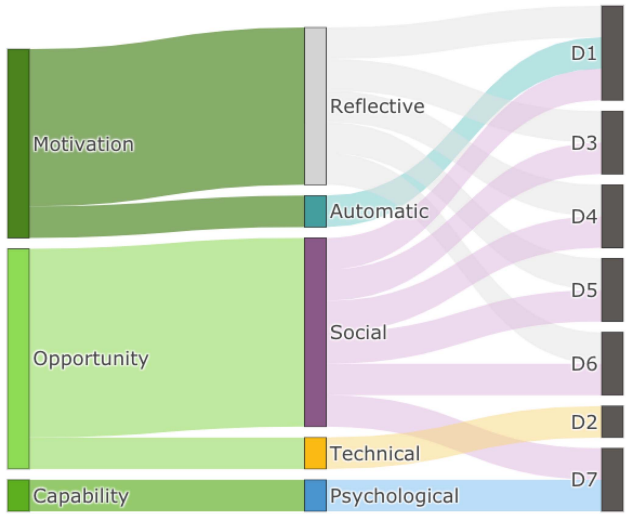

Fig. 6. *Building an organizational security culture:* **D1:** Organization promoting/mandating security, **D2:** Prioritizing security practices, **D3:** Having a security-specific role filled, **D4:** Overcoming the resistance to change, **D5:** Fostering collaboration between engineering and security teams, **D6:** Awareness of the social perception of security adoption in one's own organization and professional network, and **D7:** Providing awareness of external incentives and compliance.

the COM-B model as a diagnosis tool to highlight the *social opportunities*, *reflective motivations*, and *automatic motivations* of the behaviors related to the adoption of software security practices. *Social opportunities* represent the organization's norms and values influencing developers' attitudes towards security. *Reflective motivations* indicate how those norms and values encourage developers to reflect on their practices and adopt new habits, and *automatic motivations* indicate how developers' emotions react to those norms and values.

Social Opportunities: 23 practitioners (E1, E2, E4, E7-E17, E19, S1-S3, S5-S9) indicated that organizational culture highly influences developers' attitudes towards security. Developers are willing to adopt security practices if organizations give them the proper time and opportunity to learn and apply security in their workflow. For instance, in the words of E9, "*Adopting security requires support from the organization by facilitating the solution of a security issue.*" Sometimes organizations might treat security as a second-class citizen. For example, E16 emphasizes: "*Making it work is more important in my organization than doing it securely.*"

Reflective Motivations: Three engineers (E2, E3, E10) and four security specialists (S2, S6, S7, S9) revealed that organizations interested in embracing security as part of their culture care about developers' motivations for adopting software security practices. Strategies to disseminate security guidelines and practices across the organization influence developers' motivations for adopting software security practices. These strategies are perceived by developers as a good sign of proactiveness towards security, and they make practitioners reflect on their practices and align them with the rest of the organization. For example, E16 said: "*The company is pushing for security. Sending in the monthly newsletter what new security rules have been put in place.*"

Automatic Motivations: Six engineers (E5, E7, E9, E10, E18, E19) and two security specialists (S2, S4) experience frustration when organizations, particularly management and other departments, do not treat security as a first-class citizen, making it difficult for developers to prioritize security over other tasks. Sometimes management pushes developers to ensure a secure product despite time pressure issues and several barriers their organizations introduce that can delay the delivery of a feature or product. These delays might be caused by waiting for feedback or approval from a security team or legal and privacy department. For instance, E7 pointed out: "*So currently, we are annoyed by the slowness of the organization, so there is a legal department and privacy department, and they need to evaluate our system in terms of security, and they're just really slow in doing so.*"

D2: Prioritizing Security Practices. Not all organizations prioritize security in the same way. Depending on how critical the data managed by the application is and the resources available, organizations, if needed, will introduce different security practices across the development pipeline to ensure a secure product. Shifting security to the left, at the design stage, or leaving security to later stages during testing or operational stages are standard practices seen in the industry.

Technical Opportunities: Most practitioners (E2, E3, E5, E7-E9, E11-E18, S1-S9) perceived that organizations that prioritize security provide developers with opportunities to adopt security practices by enabling them with the appropriate resources and processes to ensure a flawless product. Typically, organizations introduce several security inspection mechanisms during the software development pipeline to rigorously identify potential vulnerabilities in the supply chain and ensure that malicious hackers cannot compromise personally identifiable information from customers. For instance, E18 highlighted: "*Security is essential for us ... There's a considerable amount of privacy work that you have to do upfront in terms of getting the consent, ensuring that you're handling the data properly ... We do regular audits and reviews from the privacy side that covers the security of the application.*"

D3: Having a Security-Specific Role Filled. Practitioners agreed that the presence of a security role within the organization can promote, maintain, and enforce security practices. Concerning the COM-B model, we identified that organizations employing a security role provides them with *social opportunities* and *reflective motivations*. *Social opportunities* are relevant due to the influential role of a security specialist in adopting software security practices within the organization. Additionally, *reflective motivations* represent how the presence of a security-specific role pushes developers to think about the security implications of their technical decisions.

Social Opportunities: Several practitioners (E1, E2, E8-E12, E14-E16, E18, S1-S3, S5-S8) highlighted the importance of having a security role in the organization. An organization that promotes a software security culture designates a specific position for security matters. In addition, developers are willing to adopt software security practices if organizations allow them to shadow security specialists to learn and understand how security issues are handled and patched. For instance, E1 stated: "*I will learn from them, see how people fix the problem, how they*

prioritize the problem, and that's one thing the company is really good at, letting you see what happens behind the scenes so you can get a good understanding of it. And eventually, that will translate into me working on more and more security-related tickets."

Reflective Motivations: Some practitioners (E10, E14, E16, S1, S2, S5, S8) indicated that security specialists can facilitate organizational discussions about security practices, the consequences of security issues, and potential threats to the system. Hence, developers will be aware of more profound implications of security issues and, as a result, will understand the rationale behind security guidelines and feel motivated to incorporate them into their software development workflow. For example, E10 pointed out: "*Having a security role in the organization, probably fosters a better environment for adopting security practices. I think that if you have that kind of security experts scattered about your organization, I think that brings everyone's security knowledge level up.*"

D4: Overcoming the Resistance to Change. Practitioners also implied that in some particular cases, developers might be reluctant to adopt new engineering practices, specifically security, because software security requires extra effort in their regular workflow—a scenario that developers are not willing to accept and a highly compelling reason behind their lack of interest.

Social Opportunities: Some engineers (E4, E10, E19) perceived that adopting security practices require convincing senior management that security is necessary and involves a cultural change where security is promoted from top management to the rest of the organization. In addition, it requires changing a collective mindset from just delivering everything as fast as possible to building a trustworthy and secure product. For example, E19 emphasized: "*For adopting security, executive sponsorship is going to be the most important, making sure that you've got the backing. Then it's just a standard change management practice. So make sure that you've got the executive sponsorship that understands the value of security and will fight for it. That's probably the most important.*"

Reflective Motivations: Three practitioners (E18, S2, S6) perceived that engineers are reluctant to change their usual engineering practices and adopt new ones. Some developers with extensive experience in development might have a negative attitude towards adopting security practices. For instance, E18 pointed out: "*Some people just might like their workflow so much, and they just don't feel like much of adopting security. They feel that the type of stuff they're working on, or the type of code they are developing, wouldn't be better by adding security.*"

D5: Fostering Collaboration Between Engineering and Security Teams. Practitioners highlighted the need to provide effective mechanisms to enhance collaboration between engineering and security teams. Typically, when software security is mandated on organizations, developers negatively perceive a security team's involvement in technical decisions across the development pipeline. For instance, some negative effects as perceived by developers are: extra work, delays, rework, or conflicting perspectives to prioritize and solve an issue.

Social Opportunities: According to most security specialists (S1, S3, S4, S6-S9) and most engineers (E1, E2, E4, E7-E12, E14-E17), developers perceive security teams as *the carrier of bad news*—the team who will notify developers every time they make a security mistake. For instance, S4 emphasized: "*Security team is usually going to be traditionally the ones that engage with devs to try and find out how to fix something that they've identified in a particular deployment.*"

Reflective Motivations: Some security specialists (S3, S4, S7, S8) perceived it is essential to work in close collaboration with developers. Developers will be encouraged to follow security practices if they feel supported by security experts during security-related tasks, facilitating the learning process and reducing the effort to apply security in their regular workflow. For example, S3 highlighted: "*For collaborations to happen is vital to understand one another, like understanding how development works and how security works. Because when one doesn't know how the other works, they will assume things and do things on how they best understand. So the other one can be put aside by mistake, and the collaboration can fail.*"

D6: Awareness of the social perception of security adoption in one's own organization and professional network. Most engineers perceived that adopting software security practices is a collaborative effort that's strongly influenced by their professional community. For instance, developers being aware of their peers' efforts to adopt software security practices provides an opportunity to join a collective effort.

Social Opportunities: Most practitioners (E1-E19, S2-S9) indicated that having people around them adopting security practices helps them follow the security procedures and reminds them why to invest extra effort for adopting those practices. For instance, E15 pointed out: "*What helps you as an external motivation to the team is having people around you that have the procedures in place. And remind you why we have those procedures, because some people don't like that, but I think it's important, so I don't mind putting extra effort into it.*"

Reflective Motivations: Some practitioners (E2, E5, E13, E15, E16, E18, S1, S5) indicated that the overall adoption within the organization significantly influences developers' perception of software security practice adoption. Not having people within the organization adopting security will considerably diminish developers' motivations to adopt security. For example, E16 pointed out: "*if nobody else takes it seriously, I'll never take it seriously. If it's not part of the culture, if it's just one guy saying security, security, security, then people will do the bare minimum to adopt security, which might be better than nothing. Still, it needs to be a part of everyone. Everybody has to care about it for you to feel like you're making secure software.*"

D7: Providing Awareness of External Incentives and Compliance. Organizations aware of external incentives, such as potential government subsidies for security testing, particularly among start-ups, represent an excellent opportunity to adopt software security practices. Additionally, developers perceive that organizations should promote awareness among developers of external regulations and compliance from the beginning of the software project, explicitly highlighting the practices, technical considerations, and justification or rationale behind the compliance.

Psychological Capabilities: Some practitioners (E2, E5, E6, E15, S4-S8) perceived it is vital for organizations to bring awareness of the compliance standards that regulate data protection and privacy, such as GDPR [12], PCI [13], and HIPPA [14]. To achieve compliance, organizations reinforce specific procedures to ensure the privacy and security of all customers' data managed by the application. For instance, some standard security practices observed are encryption mechanisms and regular penetration testing activities. Additionally, external incentives play a crucial role in adopting software security practices. For example, government incentives such as IRAP [15] allow startup organizations in Canada to subsidize 24 hours of security testing. For example, S5 highlighted: "*We work with many startups that the government subsidizes their testing. So I think that's an excellent option for startups. It's only a three-day engagement, but it's a perfect start to get an idea of where they're at. If we find that they have many input validation issues, that shows that there's something wrong with the process that needs to be addressed. If we see many configuration issues, you know, you might get insight into where they're having problems. So that can give them kind of focused advice.*"

Social Opportunities: Some practitioners (E15, S7) perceived organizations dealing with sensitive customer data or safety-critical systems adopt software security practices for compliance reasons. However, without the support of the business, the adoption will not occur. For example, S7 pointed out: "*Security is a thing that they need to do for compliance or contractual reasons. But if the business does not support security concerns, they will not adopt it. Some organizations might be willing to sacrifice security and take penalties on contracts if that works out business-wise in their favor.*"

B. Facilitating the Adoption of Software Security by Developers

Developers pointed out that they usually do not see the immediate benefits of adopting security practices but instead the adverse effects of introducing security practices that affect their development pipeline, e.g., delays. Based on our analysis, we describe the organization's crucial role in easing the adoption of software security by developers. Fig. 7 shows the relationships between the COM-B model components and the drivers from our analysis.

D8: Shaping developers' attitudes towards security. Software practitioners indicated that it is easier for developers to identify the disadvantages of adopting security instead of the immediate benefits, introducing negative attitudes towards its adoption. Developers perceived that organizations have a relevant role in shaping developers' attitudes towards security by introducing security as a "fun activity" and providing opportunities, e.g., through gamification, to adopt and learn software security practices effortlessly.

[12]https://gdpr-info.eu/
[13]https://www.pcisecuritystandards.org/
[14]https://www.hhs.gov/hipaa/index.html
[15]https://nrc.canada.ca/en/support-technology-innovation/about-nrc-industrial-research-assistance-program

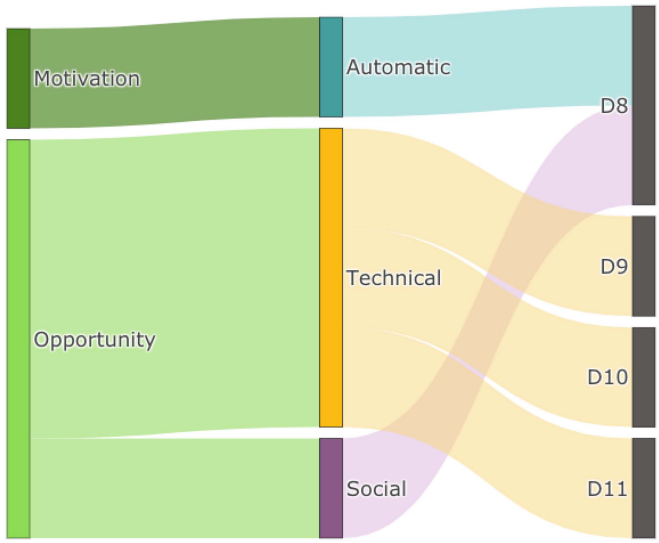

Fig. 7. *Facilitating the adoption of software security by developers:* **D8:** Shaping developers' attitudes towards security, **D9:** Tool awareness, **D10:** Standard guidelines geared at developers, and **D11:** Reduction of system complexity.

Social Opportunities: Several practitioners (E1, E2, E10, E17, S2, S3, S6-S8) highlighted that organizations play a crucial role in fostering opportunities to shape developers' attitudes towards security. Organizations that provide the conditions to get developers involved in security incident fixing processes positively affect developers' attitudes. Practitioners perceived this scenario as a promising chance to learn new technical skills as part of their regular development work. For example, E1 said: "*That's one of the things the company is good at, letting you see what happens behind the scenes so you can get a good understanding of it. And eventually, that will translate into me working on more and more security-related tickets.*"

Automatic Motivations: Most Practitioners (E1-E5, E11, E12, E16, E19, S2, S3, S5-S9) pointed out that developers working in organizations that promote software security practices often perceive security as challenging but rewarding work. In addition, developers with a positive attitude towards security feel highly motivated to adopt security practices to protect customers, users, and the company. For instance, E1 highlighted: "*I think of security as a chess game. I play one side; the attackers play the other side. It can be quite challenging, it's hard work, but it's rewarding in the end, so my motivation is to protect people and protect the company.*" Additionally, developers appreciate organizations' effort to introduce security as a fun activity. For example, E2 pointed out: "*My team did the hack your own product day, and that's a fun experience.*"

D9: Tool awareness. Organizations that aim to facilitate the adoption of software security practices typically provide the technical resources developers require to adopt it, specifically specialized security tools. Developers find it easier to adopt security if they are able to access appropriate tools.

Technical Opportunities: Several engineers (E1-E3, E5, E7, E9, E10, E13, E19) and most security specialists (S1-S5, S8, S9) believe that organizations eager to create a culture around security should actively introduce tools as part of the teams'

discussions. Developers found valuable tools that can be easily integrated into their development workflow and notify them of issues or alert them of potential risks. For instance, E5 highlighted: "*Tools involved in the process, that make a lot of sense, providing educational resources where needed. But of course, the challenge with some tools is that you don't realize them till later. The earlier we can identify that stuff, the better.*"

D10: Standard Guidelines Geared at Developers. Engineers often perceive that guidelines focused on software security are quite abstract and overly complicated for their particular information needs. Organizations play a significant role in customizing security guidelines to a developer's context and workflow. Developers will then be eager to use those security guidelines, perceive their benefit, and apply them to their regular practice.

Technical Opportunities: Several practitioners (E3, E6, E8-E10, E12, E15, E16, S1-S5) perceived that most security guidelines are not developer friendly. There is still much work needed to make those guidelines comprehensible by all security stakeholders, particularly developers. For example, S3 emphasized: "*We have a considerable fragmentation in the current application security culture. So let's say you're a developer and I'm a tester. I can't communicate with you through a unified standard. I can use CWE, CVSS, anything you can think of. And the developer will not understand what I'm talking to them.*"

D11: Reduction of System Complexity. Practitioners recognized that applications evolve innately, increasing in size and complexity, making maintenance and security management harder. Developers acknowledge organizations' effort to simplify the adoption of security by abstracting security to a specific layer/component/service in the application, e.g., applying separation of concerns.

Technical Opportunities: Inevitably, systems grow continuously, making their management more complex from a security point of view. Some practitioners (E1, E2, E5, E6, E11, E14, E15, E18, S2) perceived useful when organizations simplified security from development by fostering the adoption of security frameworks, and protocols or treating security as a core feature in the application under the management of specialized teams. For example, E11 highlighted: "*Many security details are abstracted away from development. So they are just part of the policies or the plan. And occasionally, you can contact the security team to suspend specific user permissions, but from my point of view, it's just very well embedded in the process. We don't need to know about the details, but we need to know who needs to be contacted and which requirements need to be fulfilled from a security standpoint.*"

C. Understanding Risks, Benefits, and Trade-Offs

Practitioners perceived that bringing awareness about security risks, providing information about the consequences of not adopting security practices, and discussing the benefits of building security into the development pipeline will incentivize them to adopt security. Furthermore, being aware of the trade-offs of adopting software security practices will help developers maintain their positive attitude towards security and continuous interest in inspecting security defects in their products. Fig. 8 shows the relationships between the COM-B model components and drivers.

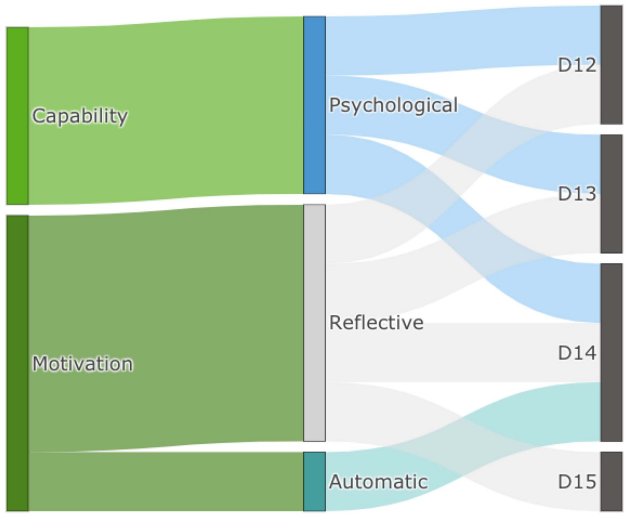

Fig. 8. *Understanding risks, benefits, and trade-offs:* **D12:** Awareness of potential risks and security incidents, **D13:** Learning from actual incidents, **D14:** Fear of non-adoption consequences, and **D15:** Knowledge of benefits.

D12: Awareness of Potential Risks and Security Incidents. Being aware of existing threats and vulnerabilities that software products face is one of the critical motivations for developers to adopt security practices. In addition, by being aware of what risks other organizations face, developers can identify what is required to secure an application and prioritize activities accordingly.

Psychological Capabilities: Most developers (E1-E6,E8-E19) often use security news on the Internet to learn about security incidents and be aware of how similar organizations have been exploited as a measure to prevent those situations from happening in their organizations. For instance, E1 stated: "*Just digging around, seeing what information I could learn about on the Internet and see what was going on. You read many security news articles about people getting hacked, and I was just curious to see how they did it.*"

Reflective Motivations: Several developers (E1, E2, E4, E5, E7, E16, E19) and some security specialists (S3, S7) perceived that when security is a significant concern in the business they belong to, they constantly gather information about security exploits in the industry. Therefore, this situation pushes developers to carefully examine their security practices to ensure a secure product for their customers. For example, E19 emphasized: "*It's hard to get a computer science degree or a software engineering degree and not be mindful and aware of security. People in this field tend to keep up with the latest news related to technology, both out of personal interest and professional interest. It's pretty much impossible to ignore what's happening with all the security breaches occurring all the time. So it's part of the consciousness of most developers.*" Additionally, E7 pointed out: "*We do care a lot about security, especially we started caring a bit more about security after some similar organizations got hacked. So we decided to look closely and see if we are doing everything up to standards.*"

D13: Learning From Actual Incidents. Practitioners indicated that having the experience of being hacked is one of the best learning resources for adopting software security practices. For instance, studying how security exploits happened, investigating how they occurred, and what the attackers were aiming for are crucial resources to prevent security incidents from happening again. In addition, patching a security vulnerability allows developers to build more confidence in handling security issues and keeps them engaged in actively incorporating security into their software development workflow.

Psychological Capabilities: Some engineers (E7, E10, E11, E13, E14, E17) and some security specialists (S4, S6) indicated that to perform security tasks, as highlighted by the COM-B model, practitioners should feel confident about executing them. Developers can gain this confidence by examining security incidents and breaches and understanding why the incidents happened in the first place. By knowing the reasons and studying a way to mitigate them, they will be more confident in assessing their own product's security. For instance, E13 stated: "*My main outcome of being hacked is my own experience that I learned from the mistakes. I've also seen and analyzed the results of what errors other people make. So, I guess it's helping more towards my experience with security.*"

Reflective Motivations: Some practitioners (E7, E10, S1, S4) highlighted that maintaining their motivations towards adopting software security practices is a complicated task. However, learning about security incidents allows developers to reflect on their practices and keep them motivated. In addition, they become aware that the chances of having a security breach are not small if they do not incorporate security into their product. For example, E7 pointed out: "*The hack of other organizations was an alert; we need to make sure that we do it better than whatever they did. And that's when we said let's introduce security restrictions to everything to maximum essentially. So there were many security features available to us that we were not using because they were potentially cumbersome.*"

D14: Fear of Non-Adoption Consequences. Practitioners considered that acknowledging the negative consequences of not adopting security in software development influences their need for adopting security.

Psychological Capabilities: Awareness of the dire consequences of not adopting security in the development workflow is highly motivating for several practitioners (E11, E12, E15, S1, S2, S4, S5, S7, S8). Among the top three adverse effects that practitioners perceive relevant are: losing time due to the considerable amount of time organizations need to invest in fixing vulnerabilities; losing money due to the number of resources required for patching and reinforcing security in the software product as well as in the engineering pipeline; and losing reputation which could lead to losing customers, subsequently causing severe financial issues in the organization. For example, E12 pointed out: "*The negative consequences of not adopting; it's undoubtedly a risk of breaches, data leaking, and risk of people getting access to something they shouldn't. And If you take it to the extreme, somebody could look remote in and wipe your whole system.*"

Reflective Motivations: The severe negative consequences of not adopting software security practices makes some developers (E5-E7, E10, E11, E13, E15, E19) reflect on to what extent their software product is "*secure enough*" and motivates them to minimize any possibility of exposure to exploits due to vulnerabilities in the software. For instance, E5 highlighted: "*I would like to protect my clients. I would like to make sure that I'm protected. I would like to make sure that our reputation is protected, then I'll be able to sleep at night.*"

Automatic Motivations: One of the main motives for several practitioners (E5-E7, E10, E13-E15, E17-E19, S7) to think about security in their software development workflow is to avoid experiencing a security incident. However, the fear of exposing confidential information is always a good reminder for developers to be careful about security implementations. For example, E6 emphasized: "*Adopting security is part of being a professional software developer and keeping your job's quality level and ethics. So it's in some sense, the quality of your work reflects your quality as a professional. So what would be very bad for the client would be very bad for me because I would probably get financial and legal charges and personal repercussions.*"

D15: Knowledge of Benefits. Practitioners recognized that awareness of the advantages of adopting security might indirectly influence their perspectives towards prioritizing security. For instance, most developers perceived that organizations reinforcing security practices build confidence among their employees to ensure a secure product for their customers. However, developers do not see the immediate benefits of adopting security in many situations; instead, they see the disadvantages of its adoption.

Reflective Motivations: Awareness of the benefits or the importance of adopting security practices makes most practitioners (E3, E7, E8, E10-E15, E17-E19, S1-S6, S8, S9) reflect on the value of introducing security at earlier stages of the software development pipeline. This way, they will avoid severe costs due to security bug fixing or paying ransomware. For instance, E7 stated: "*Starting with security much earlier in your design than what we see is like quick-to-market things. That reduces the chances that you will have to deal with emailing all your customers because of security leaks or paying fines for losing data or getting hacked and losing all your information due to ransomware attacks and paying that off.*"

Automatic Motivations: Some practitioners (E4-E7, E10, S1, S3, S7) acknowledged that they felt motivated to adopt security practices because they recognized the value of security as a selling point. Organizations that care about the security and privacy of their customers' data allow organizations to operate at a bigger scale and use it for advertising and promoting trustworthiness in their products and services. For instance, E4 stated: "*Security is my organization's most important selling point. Without security, we can't operate at scale.*" Additionally, developers indicated that they feel satisfaction when accomplishing good work by adopting security. Developers consider it rewarding to be able to prevent future headaches due to security vulnerabilities exploits. For example, E10 emphasized: "*It makes me feel like I've done a better job and that I'm helping prevent future pain from everyone else on my team.*"

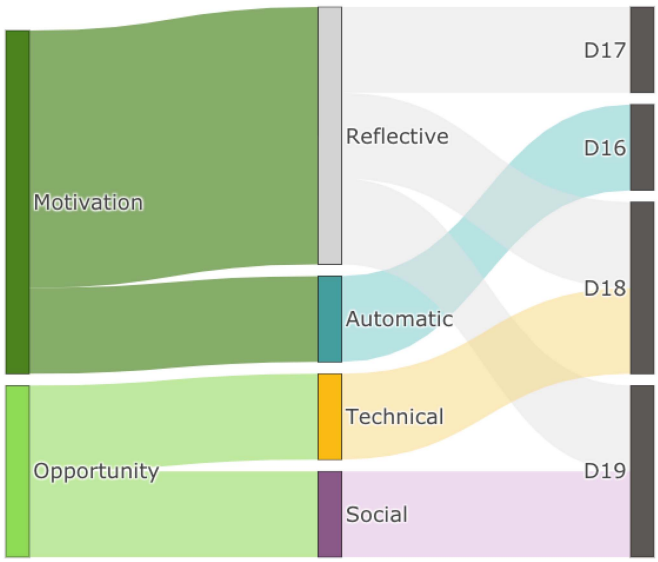

Fig. 9. *Providing contextual information to motivate developers to write secure code:* **D16:** Promoting a customer satisfaction/protection mindset, **D17:** Awareness of the influential role of the industry type in developers' disposition towards security compliance, **D18:** Awareness of developers' perceptions of the need for software security based on application characteristics, and **D19:** Aligning the perspective of what "good enough" security means.

D. Providing Contextual Information to Motivate Developers to Write Secure Code

Practitioners recognized that being aware of the context for adopting software security matters. For example, contextual information such as industry type, application characteristics, and the importance of customers' data significantly influence developers' perceptions of the need to adopt software security practices. Fig. 9 shows the relationships between the COM-B model components and drivers.

D16: Promoting a Customer Satisfaction/Protection Mindset. Organizations promoting a software security mindset oriented towards protecting customers' data better communicate the need to adopt software security practices. Developers pointed out that feeling responsible for protecting customers' data influences their mindset and priorities for security.

Automatic Motivations: Some practitioners (E9, E12, E15, E17, S1, S4, S5) feel satisfaction from adopting security practices that ensure a secure and reliable product for their customers. They feel it is their responsibility to protect customers' private information from being exposed to malicious hackers. This is particularly relevant in the case of safety-critical systems such as health care applications. For example, E15 emphasized: "*My motivation is that people should be confident that their doctor and their medical record are in good hands and that somebody does not steal it and keeps being private. So health care can continue providing services without the system being down because of some hacker attack.*"

D17: Awareness of the Influential Role of the Industry Type in Developers' Disposition Towards Security Compliance. Practitioners noticed how the prioritization of security significantly differs among some industries. For example, if developers are not dealing with sensitive information, as some practitioners disclosed in the gaming industry, they might think spending resources and time on security is unnecessary.

Reflective Motivations: In contrast to the gaming industry, Most practitioners (E1, E4, E9, E10, E12-E16, E19, S4, S5, S7, S8) perceived that some industries are more known for their security requirements. For instance, developers that work with sensitive data such as customers' payment information are more likely to feel the need to pay attention to the security implications of their technical decisions. Therefore, they feel motivated to adopt software security practices. For example, E9 emphasized: "*Major customers are coming from the financial services, so my security concern is very high.*" Additionally, E16 pointed out: "*In the video game industry, software security is not taken seriously. Some industries require like PCI compliance but not for video game industry.*"

D18: Awareness of Developers' Perceptions of the Need for Software Security Based on Application Characteristics. An application's characteristics remarkably influence practitioners' perception regarding the need to adopt software security practices. Developers perceived that the security considerations in their technical decisions differ significantly depending on features such as the application type, volume of users, or to what extent applications are exposed to internet traffic.

Technical Opportunities: Several practitioners (E6, E10-E12, E14, E15, E17, E18, S4, S8, S9) indicated that not all applications have the same security concerns. The application's technical specifications play a crucial role in identifying the value of introducing security practices. In addition, security requirements might scale to a different level in the system architecture, i.e., at a network or infrastructure level instead of at an application level. For example, E14 highlighted: "*Because usually, my code or application uses something that doesn't need input. If you don't have an input, usually, it works by itself. So what you need to check is that your platform or your server is secure, but not your code because there's nothing outside that can change it.*"

Reflective Motivations: Web applications, mobile applications, and embedded systems have different characteristics, and therefore, multiple security considerations. Some developers (E3, E4, E7, E8, E11, E12, E16-E18) do not feel motivated to adopt security when developing specific applications due to their perception of the reduced likelihood of exposure to potential risks. For instance, developers who write code for embedded systems feel security is not a significant issue while developing software, or they are not even aware of security guidelines for this particular type of application. For example, E17 pointed out: "*Security is related to the type of your application. We develop embedded systems, so I don't know if there are security rules. Thus, we do not care about the security of our code. It should be secure from a higher level.*"

D19: Aligning the Perspective of What "Good Enough" Security Means. Practitioners had a negative perception of the conflicts among different stakeholders involved in security decisions. Each stakeholder has a different priority and perspective for security, and these differing views about how the organization should apply security in software development become barriers to its adoption. Developers considered it critical for

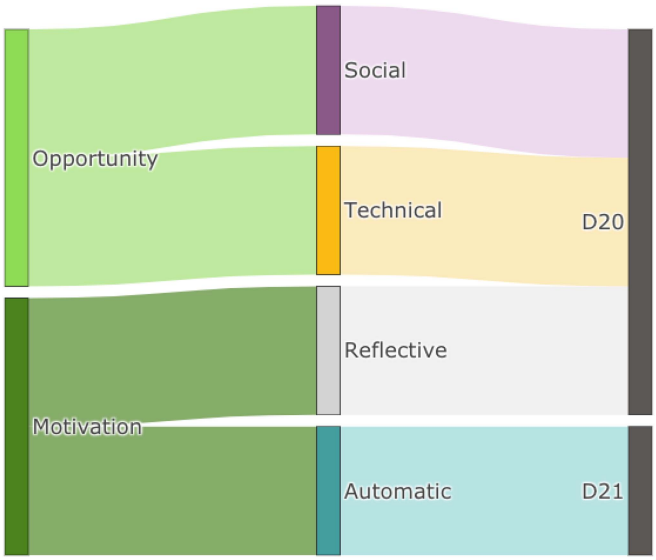

Fig. 10. *Providing justification for necessary tools and process constraints:* **D20:** Consideration of tool constraints on developers' autonomy, and **D21:** Awareness of developers' perception of security-imposed restrictions.

organizations to align stakeholders' perspectives to mitigate this barrier.

Social Opportunities: Several practitioners (E2, E3, E9, E12, E13, E16, S5, S7) indicated that when several stakeholders are involved in security decisions, conflicts naturally arise due to the different perspectives and priorities each stakeholder holds regarding security. For example, E9 highlighted: "*POs usually get together when the definition of done includes security concerns. POs and the security team should clarify and check them. The stakeholders would be mainly the PO, the user, and the security team in this case. Discussions are about understanding why the bug is an issue, how to fix it, and whether to prioritize it.*"

Reflective Motivations: Most developers (E5-E8, E10-E12, E15-E19) perceived that believing they have a fully secure software product is not realistic; there is always something to improve or learn about software security. This uncertainty keeps them motivated to stay up-to-date and adopt security practices to minimize potential risks. For instance, E5 stated: "*I believe we're doing a reasonable job, but I would like to continue improving it. So, we use things that we know are very high risk, such as credit card processing. We use third-party services which are certified to handle all of that sort of stuff. But we believe our software is secure, but I know that we could do better. I know that things are stepping up regarding what's happening with the tech, and we could continue to improve substantially.*"

E. Providing Justification for Necessary Tools and Process Constraints

Practitioners' negative attitudes towards security are usually driven by their beliefs that adopting security restricts their freedom of choice or autonomy for selecting the most convenient tools, libraries, or technologies to perform development tasks. Fig. 10 shows the relationships between the COM-B model components and drivers.

D20: Consideration of Tool Constraints on Developers' Autonomy. Typically, organizations with an innate security culture reinforce security practices by limiting the set of development tools, third-party libraries, or in general, any software component that developers can use in their regular workflow. Engineers perceive this situation affects their autonomy and freedom to choose the most convenient technologies to perform their work. Practitioners indicated that proper dissemination of the benefits and rationale behind those restrictions are crucial to mitigate any negative attitude caused by imposed restrictions.

Social Opportunities: Some practitioners (E4, E9, E12, E15, S9) had a negative perception of the standard security practices in organizations that impose restrictive policies for using tools or any third-party component in a software project. In this context, the role of a security or compliance team is to inspect and approve any potential artifact developers may want to introduce in the project and has not been validated previously. So, for instance, E12 pointed out: "*An independent security and compliance team that has to review pretty much anything. Any external dependency you introduce has to go through the security team. Any new artifact that we produce, I think, has to be reviewed by security. So there are various checks to make sure that you aren't shipping something that could be a vector for problems.*"

Technical Opportunities: Some practitioners (E1, E2, E9, E14, E17, S4, S9) have a negative perception of adopting software security practices due to the strict restrictions while choosing any tool to develop software. For instance, E9 emphasized: "*Adopting security limits the freedom to choose what libraries we can use for development, or many security checks need to be passed.*"

Reflective Motivations: Some developers (E13, E14) like the freedom to choose the most suitable tool for doing their work. Organizations imposing strict restrictions negatively affect developers' motivations to adopt security practices. For example, E13 stated: "*I think developers just like freedom of what they are doing because some people choose small companies and startups specifically for the freedom of doing what you want and how you want to do it. In big companies, it is always more restrictive.*"

D21: Awareness of Developers' Perception of Security-Imposed Restrictions. Developers perceived that adopting software security practices introduces many disruptions to their regular workflow. For instance, delays due to security inspections, waiting for feedback during security code review, or delays caused by the security team regarding the authorization to use a third-party component or library. On top of that, following security guidelines might introduce more complexity to the application, making its maintenance difficult. Therefore, software practitioners encourage organizations to identify developers' negative attitudes towards software security to foster its adoption.

Automatic Motivations: Several practitioners (E1, E3-E6, E9, E11-E16, E18, S4-S9) indicated that developers' motivations to adopt security are influenced by the perception that some security guidelines are unhelpful and unnecessary, especially

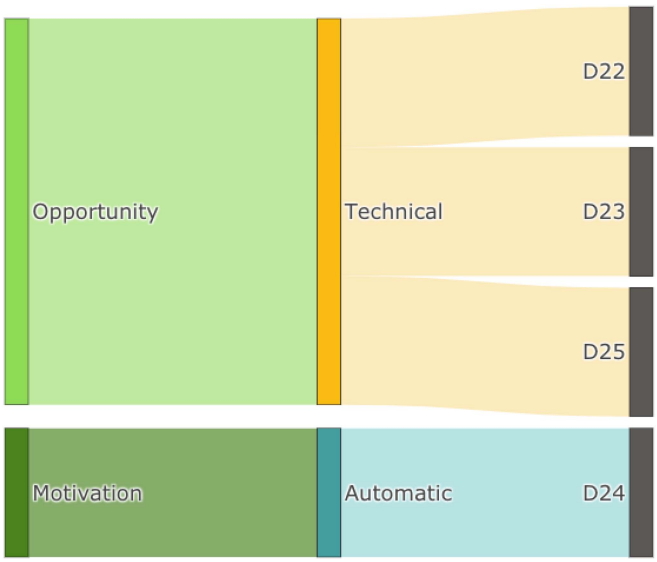

Fig. 11. *Providing (cognitive) support to developers for writing secure code:* **D22:** Availability of reminders, i.e., checklists, dashboards, etc., **D23:** Improving the usability (complexity reduction) and accuracy of security tools, **D24:** Reducing the effort required to learn or apply security, and **D25:** Integrating tools into the development workflow.

when they introduce more complexity to the system design instead of a straightforward solution; Therefore, causing extra work in terms of maintainability. For instance, E3 emphasized: "*I work in a private network in this telecommunication company, so we are not open to the Internet. So using these Web services instead of a simple protocol, which is faster, doesn't make sense. So I think that was unnecessary. That's a barrier I've found; some security guidelines are unhelpful and unnecessary.*"

F. Providing (Cognitive) Support to Developers for Writing Secure Code

Practitioners find it challenging to incorporate software security practices into their software development workflow due to the overwhelming number of topics they need to assimilate to write proper secure code. On top of that, developers perceive that security tools are pretty complex and sometimes inaccurate, which considerably affects their adoption. Fig. 11 shows the relationships between the COM-B model components and drivers.

D22: Availability of Reminders, i.e., Checklists, Dashboards, etc. Practitioners indicated that software security is not a topic off the top of their heads. Reminders such as checklists are helpful resources to prevent overlooking any security considerations during code reviews. Another useful reminder perceived by developers is security tool notifications containing actionable feedback. In some particular cases, organizations with strict security policies will not allow practitioners to move forward in the development pipeline until any security concern raised by the tool is fixed.

Technical Opportunities: Most practitioners (E2-E7, E10, E11, E13, E14, E16, E18, E19, S1, S5, S6, S8, S9) acknowledged that reminders are helpful to prevent overlooking security concerns while developing software. Besides security checklists, practitioners recognize the usefulness of dashboards, a visual tool that usually contains graphics and comprehensive summaries of crucial information regarding Q&A and security metrics. For instance, E2 pointed out: "*We have a reminder to check the dashboard. Otherwise, we will forget because it's not always on top of our minds. The dashboard highlights vulnerabilities at the level of hosts, containers, and packages. In addition, it shows how many high/medium/low vulnerability issues exist in the application.*"

D23: Improving the usability (complexity reduction) and accuracy of security tools. Practitioners pointed out that most security tools contain usability issues, making them complex to use and manage, a situation that discourages them from adopting software security practices. Additionally, security tools require sophisticated configuration to avoid a high volume of false-positive results. Default configurations result in a high level of inaccuracy and are a detriment to the value and usefulness of the tool. In this regard, developers highly appreciate security specialists' support for appropriate tools configurations.

Technical Opportunities: Some engineers (E1, E2, E9, E11-E13) are very conscious of the importance of security tools in adopting security practices. They recognized that better tools and security analyzers would help in their application of security practices during software development. However, some barriers to adopting security tools rely on usability and accuracy characteristics. For example, E13 highlighted: "*What would help me apply security practices? Having better tools, for sure, so better analyzers that would prompt right away where there is a security vulnerability. However, this needs to be fixed ... working here made me understand that these tools are unreliable and can give errors and false positives.*"

D24: Reducing the Effort Required to Learn or Apply Security. Software security is a broad topic for developers. Considerable effort and dedication are required to learn and gain enough experience to write secure code correctly. Developers highlighted that organizations' efforts to reduce the scope of learning topics and provide a learning roadmap customized to developers' information needs would facilitate their disposition to adopt software security. For instance, considering topics focused on their particular applications' attack surface, programming language, frameworks, etc., are highly valued by developers.

Automatic Motivations: Most practitioners (E1-E7, E9-E19, S1-S9) perceived that adopting software security practices demands a lot of cognitive effort, such as investing a significant amount of time for learning different techniques, absorbing a lot of information, and keeping up-to-date knowledge. These challenges highly influence developers' motivations to adopt security practices and become a significant concern for organizations interested in promoting security practices in their organizational culture. For instance, E6 stated: "*Security shouldn't be complex. If it's difficult, people will not follow it. So it seems that it's in the company's best interest to make it straightforward for developers by giving developers clear guidelines, being more proactive, and making those guidelines available when people are dubious about if something is secure or not. So let's say a good company doesn't make security difficult.*"

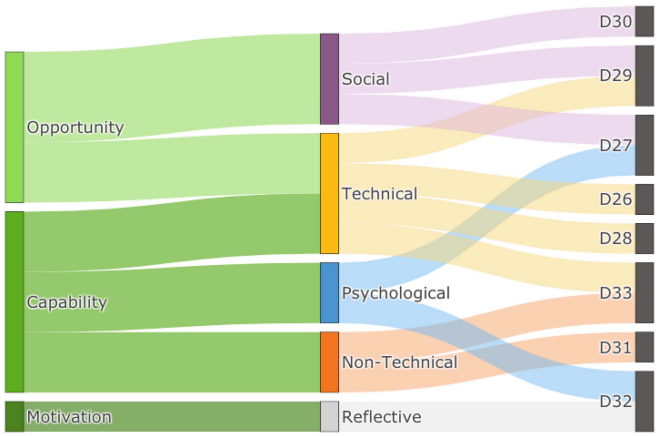

Fig. 12. *Facilitating developers' acquisition of security-specific skills:* **D26:** Access to learning resources, **D27:** Using security practices as learning tools, **D28:** Providing security education, **D29:** Fostering hands-on learning / self-learning / osmosis, **D30:** Creating and participating in Communities of Practice, **D31:** Having non-technical skills, **D32:** Confidence in their technical abilities, and **D33:** Awareness of necessary security skills.

D25: Integrating Tools Into the Development Workflow Automating software security assessments is perceived by developers as a significant facilitator to adopt security practices. Furthermore, integrating security tools into the Continuous Integration (CI) pipeline allows developers to count on proper feedback regarding any potential security flaw in their source code and react accordingly.

Technical Opportunities: Several practitioners (E11, E13-E15, E18, S1, S3, S4, S7-S9) highlighted the importance of automating security by integrating security tools into the CI/CD pipeline. This way, security scanning can be performed automatically and expose potential vulnerabilities before deployment. For example, S1 emphasized: "*Once you integrate security into the CI/CD pipeline, you make sure that your code is clean. So if you use tools like OWASP Zap* [16] *or Arcane, you know they're going to tell you whether you have a broken URL or you are managing passwords or secret information incorrectly, and this is going to spew them out during scanning time. Then, you can quickly fix them before you go to deployment.*"

G. Facilitating Developers' Acquisition of Security-Specific Skills

The ability to perform security-related tasks does not necessarily depend only on technical skills. Practitioners perceived that having support from their organizations helps them acquire the necessary technical and non-technical skills to understand security challenges. In addition, support from organizations relies on providing developers with the right tools and learning resources to facilitate the adoption of software security practices. Fig. 12 shows the relationships between the COM-B model components and drivers.

D26: Accessibility to Learning Resources. Practitioners perceived that, now more than ever, there is a massive diversity of learning resources to boost technical skills in software security. For example, developers might access free online courses, conferences, specialized blogs, and publicly available security guidelines to keep up-to-date in their knowledge of security topics.

Technical Opportunities: Most practitioners (E1-E6, E9-E12, E14, E16, E18, S2-S6, S9) indicated that they can boost their technical skills in security by benefiting from learning resources provided by organizations. An organization that offers its developers courses, training, and conferences fosters an environment for security learning. For instance, S2 stated: "*There are great webinars by OWASP. Companies like SecureIdeas, TrustedSec, Black Hills Information Security, among others, provide these worthy webinars. It's like baking cyber security into your development process because they're just like introductory level. For example, organizations, having their development team watching one of those webinars every quarter or doing a secure coding tournament from Secure Code Warrior will get developers thinking about security and start them on that path.*"

D27: Using Security Practices as Learning Tools. Engineers recognized the significant value of being involved in security practices within the organization. This situation provides an effective way to learn software security. For example, developers highlighted that the feedback they receive from security code reviews, especially when the security team is involved, helps them identify anti-patterns in their coding practices and comprehend security flaws.

Psychological Capabilities: Several practitioners (E3, E9, E14, S2, S3, S4) perceived that they gain more confidence by being involved in security practices. Experiencing how to tackle a real security issue helps developers build confidence in their capabilities. Additionally, security specialists highlighted that security games are an effective learning tool for getting developers familiar with potential threats and building a security mindset. For instance, S3 stated: "*I usually recommend STRIDE and Elevation of Privileges Game which contains 12 threats per category. It's kind of a game where developers play cards to see which threat hits on your application or your service. This game is something that they can start up with because that's around 50 threats.*"

Social Opportunities: Some practitioners (E1, E10, E12, S4) indicated that organizations play an essential role in providing developers with experiential learning resources using security practices. Practices such as code review and pair programming help practitioners have a deeper understanding of security issues. In addition, organizations that foster knowledge sharing using security practices bring more opportunities for their developers to adopt security. For instance, E12 pointed out: "*I learned more through colleagues, best practices, and being mentored. In that regard, code review is an excellent avenue for learning. As I grew as a professional, I learned a lot through people, like pointing things out in code review.*"

D28: Providing Security Education. Engineers perceived continuous learning and proper training as crucial to mastering software security practices. Developers emphasized the vital role of organizations in providing developers with the opportunities to keep up-to-date in their security knowledge. Additionally,

[16] https://www.zaproxy.org/

practitioners highlighted the need to improve security education at the university level by integrating coding best practices with security compliance source code.

Technical Opportunities: Most practitioners (E1, E2, E4, E7-E15, E17-E19, S1-S3, S5-S9) do not take exclusively security-related courses as part of their formal education in computer science. Instead, software security is a topic that they pick up along the way while developing technical expertise. For example, E10 highlighted: "*Security was a topic included in many of those courses that had things to do with networks and systems. So there were some security topics when we got into things like developing Web applications. But I would say it was rudimentary at best. It was mostly just elementary stuff. There was much more emphasis on it when we were doing things like setting up networks or configuring a Web server on Linux or things like that. But when it comes to writing secure software, I would say that education at the university level didn't treat security as important as those other security aspects.*"

D29: Fostering Hands-on Learning/Self-Learning/Osmosis. Practitioners recognized the value of organizations in providing developers with opportunities to learn software security by being indirectly involved in fixing security vulnerabilities or shadowing specialists while patching a security flaw. Additionally, organizations providing developers with the necessary time to learn software security by themselves or the opportunity of being mentored by a specialist are perceived by developers as highly effective methods to motivate developers to write secure code.

Technical Capabilities: Several practitioners (E1-E3, E5, E11, E14, E17, S2, S3, S5-S8) perceived that the ability to self-learn is crucial for starting a career in software security. A self-directed learning approach is vital to engage with the security learning process. Software security is a broad topic and requires, besides formal training, to keep up-to-date knowledge of the security exploits happening every day. For instance, S6 pointed out: "*First, I was just learning security by myself and studying on the Internet. Then I got more formal training and got involved with the community by going to conferences and giving some presentations. I think networking is a big thing in security. To start learning, I would say start small. Start with one type of vulnerability. Understand it properly, understand how to fix it. Look for that in your code or the codebase you're responsible for, and then go to other categories. We can't follow all the vulnerabilities and issues that happen every day and every week. So start small, don't try to learn about everything.*"

Social Opportunities: Some practitioners (E7, E10, E12, E14, S6, S9) recognized that advocating for security within the organization is vital to demonstrate the need to adopt security practices to all stakeholders. A starting point could be using a proof of concept to expose the security issues identified in the codebase and the implications for the organization if we overlook them. For example, S9 highlighted: "*To keep developers' motivations to adopt security, it's important to advocate for additional security controls in your environment; you can demonstrate using a proof of concept that there are severe security issues. That can help motivate folks to take action and find time to work on it. Self-learning is essential, like OWASP, and we also have the purple folks. Those are our two best resources right now, and they just partnered together to offer classes and stuff, which is fantastic. It's kind of like demystifying security in the company.*"

D30: Creating and Participating in Communities of Practice. Practitioners emphasized the crucial role of communities of practice to learn software security. Practitioners perceived that a relevant characteristic of a good security culture is organizations creating and promoting an environment where people interested in security-related topics gather together to learn and share knowledge. Additionally, practitioners considered it essential to join efforts with external communities since being hacked could affect any organization or business.

Social Opportunities: Currently, developers are using external communities to get help on security issues. For example, communities such as OWASP provide an excellent environment for practitioners to learn about security. Similarly, most practitioners (E1, E3-E5, E8-E10, E12, E14, E18, E19, S1-S9) perceived that organizations that form communities inside the company and gather all the practitioners who show security interest help build a supportive environment to facilitate security learning. For instance, S8 emphasized: "*There are many gangs in the community and people doing similar things helping each other solve security issues and learning.*"

D31: Having Non-Technical Skills. Besides having proper technical skills in software security, developers emphasized having non-technical or soft skills. Ensuring a secure software product is a collaborative effort and demands the participation of several stakeholders. Inevitably, conflicts between stakeholders may arise due to different priorities and perspectives of approaching and applying security. In these situations, developers emphasized the critical role of having soft skills such as communication, critical thinking, empathy, etc., to facilitate discussions and mitigate any potential conflict.

Non-Technical Capabilities: All practitioners (E1-E19, S1-S9) indicated that communication is one of the most important non-technical skills required to conduct software security practices. For instance, they believe communication is essential when describing the importance of security to other stakeholders, discussing security issues with developers, and handling security incidents. Furthermore, developers need to elaborate the situation to the management team and other stakeholders to discuss further actions when an incident happens. For example, E8 pointed out: "*Besides communication, researching is also a critical non-technical skill because you have to deeply investigate what went wrong when you are facing a security issue.*"

D32: Confidence in Their Technical Abilities. Engineers indicated that confidence in their technical abilities in software security significantly influences the adoption of software security practices. Developers' low confidence in their technical skills will discourage them from adopting security practices. Contrarily, over-confidence is perceived negatively and as a deterrent to their motivation to keep up-to-date knowledge to improve their security practices.

Psychological Capabilities: Most practitioners (E1-E19, S1, S3, S6-S9) perceived that confidence is essential to perform any engineering task. Developers become confident in their technical abilities to conduct software security tasks because of their experience facing security threats or due to the collective

expertise of their software teams. Having someone on the team who has security experience boosts the entire team's confidence. For instance, E16 pointed out: "*Academic knowledge can get you close to ensuring secure software. However, having expertise or having someone on your team who has experienced a security threat will give you more confidence to say you have a secure product.*"

Reflective Motivations: Some engineers (E5, E6, E14) recognized that security is not their primary focus at work; It is not a topic that they need to apply daily. However, developers are familiar with self-learning and hands-on learning techniques within a software development context, which are also essential and commonly used in software security. Therefore, it is not a drastic transition for a developer to feel motivated to dig deeper in software security and become a security champion or an application security specialist. Due to the deluge of available security information, it is also vital for practitioners to recognize that security is a never-ending learning cycle; There are new sophisticated mechanisms, tools, guidelines, and best practices to learn. Such intense and dynamic context motivates developers to aggregate security to their professional profile. For instance, E5 stated: "*So the first question is how comfortable am I with security? I think I know enough that I don't know much, but I'm not starting from zero. I know that there are many more sophisticated mechanisms out there right now. And I know that we've got a lot of development experience that we can just use it.*"

D33: Awareness of Necessary Security Skills. Engineers perceived that awareness of what they should know to properly adopt and apply software security practices significantly influences their motivation to adopt security practices. For instance, developers considered it relevant to know about malicious hackers' mindsets, their methods, most common attack vectors, and the best coding practices to prevent them.

Technical Capabilities: Most practitioners (E1, E6, E7, E9, E11-E15, E17, E18, S1-S9) acknowledged that it is essential to have a solid understanding of the security basics to understand technical aspects of security exploits, follow security guidelines, and apply security patches, and therefore, prevent harmful consequences in a potential system attack. For instance, E11 highlighted: "*Knowledge of defensive programming patterns and how to sanitize inputs is valuable. Knowing about security issues and what is needed to solve them it's part of the job.*" Additionally, E15 emphasized: "*You need to know about software architecture, software testing, many technical issues like how you design software to be secure? And you need to know what type of threats are there? How can you do the testing? How do you do penetration testing? Also, to let other people do that. So you should always think you should always question your security knowledge because you can think about everything upfront. However, still, somebody else will be able to penetrate the system.*"

Non-Technical Capabilities: Some practitioners (E9, E15, E18, S1-S3, S5, S9) recognized that to ensure a secure software product, it is crucial to consider the human aspects of the product in addition to the technical features. In this context, it is valuable to acquire knowledge about the users, where the product is used, and what abnormal product usage could potentially occur. In other words, to analyze upfront what can go wrong due to a misuse of the application by the end-user. For example, E15 emphasized: "*Regarding non-technical skills, I need to have domain knowledge and knowledge about end-users. That's very important where the system is used, what can happen if that something doesn't work or goes wrong or how people enter the system, if they share their password data or how people work with that.*"

V. Related Work

We briefly provide an overview of the related research and how it aligns with the drivers (shown in italics) our study revealed. We searched the ACM, IEEE Xplore, and Springer research databases and found 15 papers that studied the adoption of software security practices, many of which were published quite recently. Two of these papers were systematic literature reviews of work on this topic [44], [47]. We found that most (30/33) of the drivers that emerged from our study also appeared in this other research. Although most papers reported between 3 and 19 of the drivers we found, Mokhberi and Beznosov's [47] systematic literature review reported 28 of our drivers. This overlap further strengthens the relevance of our work, but we were surprised that several of the drivers we found to be quite important were not mentioned in these 15 papers. These drivers are *(1) D17: awareness of the influential role of the industry type in developers' disposition towards security compliance*, *(2) D20: consideration of tool constraints on developers' autonomy*, and *(3) D27: using security practices as learning tools.* We summarize how our research relates to the related research in Table VI. This table can be used to not only see which of the drivers have been reported before, but it also provides an index into further reading about these drivers, while highlighting that some of them may perhaps call for further research.

Building an Organizational Security Culture. Some researchers have extensively studied how organizations build a security culture, specifically *organizations promoting or mandating security (D1)*. For instance, Rauf et al. [44], Jones and Rastogi [45], and Mokhberi and Beznosov [47] exposed the lack of security culture in teams and organizations as a significant deterrent to the adoption of security. In particular, Jones and Rastogi highlighted that management should be responsible for disseminating the security policies, standards, guidelines, and procedures across all teams in the organization [45]. Additionally, some research literature has revealed the relevant role of organizations in *prioritizing security practices (D2)*. For example, Mokhberi and Beznosov [47] and Poller et al. [50] agreed that organizations that do not provide the necessary resources prevent developers from implementing security. Specifically, Poller et al. pointed out that when managers see security as a resource conflict with feature development, developers also perceive implementing security as not worth the time and energy [50].

Several other researchers have emphasized the critical function of a *security-specific role in the organization (D3)*. For example, Xie et al. [49] pointed out that security experts usually

TABLE VI
KEY DRIVERS ASSOCIATED WITH THE ADOPTION OF SOFTWARE SECURITY PRACTICES THAT HAVE EMERGED IN THE PREVIOUS LITERATURE

		Rauf et al. [44]	Haney et al. [33]	Jones and Rastogi [45]	Werlinger et al. [46]	Mokhberi and Beznosov [47]	Lopez et al. [48]	Xie et al. [49]	Poller et al. [50]	Sajwan et al. [51]	Assal and Chiasson [52]	Assal and Chiasson [53]	Votipka et al. [54]	Weir et al. [55]	Braz et al. [56]	Fischer and Grossklags [57]	Totals
Building an organizational security culture																	
D1	Organization promoting/mandating security	•		•		•			•		•	•					**6**
D2	Prioritizing security practices	•		•	•	•		•	•	•	•	•			•		**10**
D3	Having a security-specific role filled		•	•		•		•	•		•		•				**7**
D4	Overcoming the resistance to change			•		•			•	•	•	•					**6**
D5	Fostering collaboration between engineering and security teams	•		•	•	•											**4**
D6	Awareness of the social perception of security adoption in one's own organization and professional network	•	•	•		•					•	•		•			**7**
D7	Providing awareness of external incentives and compliance	•		•		•		•		•							**5**
Facilitating the adoption of software security by developers																	
D8	Shaping developer's attitudes towards security	•		•		•	•	•	•	•	•	•	•				**10**
D9	Tool awareness	•		•		•					•	•	•				**6**
D10	Standard guidelines geared at developers	•		•		•											**3**
D11	Reduction of system complexity	•				•		•					•				**4**
Understanding risks, benefits, and trade-offs																	
D12	Awareness of potential risks and security incidents	•		•	•	•	•		•	•	•				•	•	**10**
D13	Learning from actual incidents					•											**1**
D14	Fear of non-adoption consequences					•					•	•					**3**
D15	Knowledge of benefits			•		•					•			•			**4**
Providing contextual information to motivate developers to write secure code																	
D16	Promoting a customer satisfaction/protection mindset		•			•		•	•	•	•	•					**7**
D17	Awareness of the influential role of the industry type in developers' disposition towards security compliance																**0**
D18	Awareness of developers' perceptions of the need for software security based on application characteristics							•		•	•						**3**
D19	Aligning the perspective of what "good enough" security means	•		•	•	•			•		•						**6**
Providing justification for necessary tools and process constraints																	
D20	Consideration of tool constraints on developers' autonomy																**0**
D21	Awareness of developers' perception of security-imposed restrictions			•		•		•		•	•						**5**
Providing (cognitive) support to developers for writing secure code																	
D22	Availability of reminders, i.e., checklists, dashboards, etc.	•													•	•	**3**
D23	Improving the usability (complexity reduction) and accuracy of security tools	•			•	•					•						**4**
D24	Reducing the effort required to learn or apply security	•				•		•			•					•	**5**
D25	Integrating tools into the development workflow					•											**1**
Facilitating developers' acquisition of security-specific skills																	
D26	Accessibility to learning resources	•	•	•		•				•			•				**6**
D27	Using security practices as learning tools																**0**
D28	Providing security education			•		•			•	•				•			**5**
D29	Fostering hands-on learning/self-learning/osmosis					•			•	•							**3**
D30	Creating and participating in communities of practice	•				•			•	•							**4**
D31	Having non-technical skills	•	•		•	•	•										**5**
D32	Confidence in their technical abilities	•	•	•		•	•			•	•		•		•		**9**
D33	Awareness of necessary security skills	•	•	•		•	•			•	•		•				**8**
	Totals	**19**	**7**	**18**	**6**	**28**	**5**	**9**	**10**	**14**	**18**	**8**	**7**	**5**	**3**	**3**	

act as security supervisors of the whole development process. However, Xie et al. [49] also indicated in the same study that developers might exhibit a more relaxed attitude towards security when there are experts to back them up. Moreover, Poller et al. [50] highlighted that security inspections conducted by external security consultants become an eye-opener, fostering awareness among developers about the security topics they need to look after in their daily work. Furthermore, some researchers recognized the importance of the *awareness of the social perception of security adoption (D6)*. For example, when the whole team is responsible for security, the motivation for adopting and implementing security could have a snowball effect and lead to motivating more team members to acknowledge the value of adopting security [52], [53].

Facilitating Developers Software Security Adoption. Other researchers have found that organizations play a crucial role in *shaping developers' attitudes towards security (D8)*. For instance, Rauf et al. [44], Jones and Rastogi [45], and Mokhberi and Beznosov [47] found that developers usually do not perceive the usefulness of security practices. Their studies highlighted that most developers might have an attitude that security is someone else's responsibility [44], or perceive it as a hindrance [45], or in contrast, consider security to be a shared responsibility [47]. Furthermore, other researchers also emphasized that interaction

through a gamification approach is an effective tool to engage developers in security practices as developers often enjoy the physical aspects of a game [48].

Several researchers have pointed out *tool awareness (D9)* as a relevant driver. For example, the lack of awareness of security tools and vulnerabilities [44] reduces the likelihood of developer involvement in security practices. A similar lack of adoption occurs when organizations do not provide the proper training for using security tools. As a result, developers usually use security tools without a complete understanding of tool functionality [47]. Additionally, a few other researchers have emphasized the importance of organizations providing *standard guidelines geared at developers (D10)* and the *reduction of system complexity*. For instance, Mokhberi and Beznosov [47] reported that developers often face a lack of general security guidelines and no one in charge of ensuring that those security requirements are followed.

Understanding Risks, Benefits, and Stakeholders' Trade-Offs. Several researchers have recognized the value of understanding risks, benefits, and stakeholders' trade-offs, in particular, the *awareness of potential Risks and Security Incidents (D12)*. For instance, Rauf et al. [44] pointed out that developers might misplace trust on frameworks or third-party APIs. As a result, developers can introduce vulnerabilities into the source code, assuming that frameworks or libraries properly handle security by default. Additionally, Lopez et al. [48] highlighted that public incidents enable information trading and risk awareness. Developers usually build awareness by expanding on technical information and providing additional scenarios and examples from their personal experiences.

Other researchers have emphasized the crucial role of drivers such as *fear of non-adoption consequences (D14)* and *knowledge of benefits (D15)*. For instance, organizations' security efforts are less effective when developers perceive a disinterest in adopting software security practices. This situation usually happens when there are no perceived negative consequences to the customers or the business from the lack of security in the SDLC [53]. Additionally, Assal and Chiasson [52] highlighted that developers feel motivated to adopt security practices when they are aware of similar software (to the one they work on) suffering a security breach—this situation becomes an "eye-opener" for them. Finally, Mokhberi and Beznososv [47] recognized that *having experienced a real security issue (D13)* is the primary driver that increases awareness and concerns about security among developers in the long run. As a result, adopting security practices and learning about security mechanisms to protect their code becomes a priority.

Providing Contextual Information to Motivate Developers to Write Secure Code. Some researchers have emphasized the relevant role of the organization in providing contextual information to motivate developers to write secure code. In particular, Xie et al. [49] highlighted the importance of *promoting a customer satisfaction/protection mindset (D16)*. They pointed out that a critical motivator for developers is the concerns of the customer or client: If the customer cares about security, the company has to care about security. Furthermore, Poller et al. [50] confirmed a similar result, highlighting that any feedback from the customer motivates developers to write secure code. Furthermore, Assal and Chiasson [52] emphasized that developers who care about their users' security and privacy feel encouraged to adopt security practices.

Other researchers have pointed out that *aligning the perspective of what "good enough" security means (D19)* and *awareness of developers' perceptions of the need for software security (D18)* are vital drivers that organizations should pay careful attention to in order to encourage developers to implement security practices. For instance, Werlinger et al. [46] highlighted that developers have to communicate with other stakeholders that hold different perceptions of risks, sometimes considering security as a second priority and not having security culture training. Therefore, developers feel the need to persuade these stakeholders of the importance of security controls, which sometimes becomes frustrating. Additionally, Xie et al. [49] highlighted that *developers' perceptions of the need for software security (D18)* are influenced by the applications' characteristics. For instance, they pointed out that middleware developers might not be concerned about adopting security practices since they believe security should only be an issue for front-end applications. Moreover, Assal and Chiasson [52] emphasized that developers' perceptions of the need for software security might be influenced by false assumptions that the software they develop is not prone to security attacks. They might also believe that users will not be technically capable of doing anything malicious for fear of losing their jobs. Interestingly, no study reported the importance of *developers' perceptions and disposition towards security compliance based on the type of business (D17)* they develop software for.

Providing Justification for Necessary Tools and Process Constraints. Researchers have agreed that the *awareness of developers' perception of security-imposed restrictions (D21)* is a crucial driver to motivate developers to write secure code. Specifically, Jones and Rastogi [45] emphasized that developers perceive security as a barrier to functionality, adding constraints and reducing flexibility. Additionally, Xie et al. [49] highlighted that developers consider security as an expense and potentially time-consuming activity. So when the budget is limited, software security is one of the concerns that can be overlooked. Furthermore, sometimes developers commonly perceive that by focusing more on software security, companies might lose their business opportunities [52]. Surprisingly, we found no studies that reported the *consideration of tool constraints on developers' autonomy (D20)*.

Providing (cognitive) Support to Developers for Writing Secure Code. Researchers have recognized that organizations should provide developers with cognitive support to facilitate the adoption of security practices. Specifically, Werlinger et al. [46] emphasized that developers feel discouraged to adopt security practices when *security tools' usability and accuracy (D23)* become part of the problem instead of being a facilitator to fix security vulnerabilities. For instance, they pointed out several security tool issues that require attention from tool providers, such as better support for collaboration, decreased complexity, support to disseminate knowledge, flexible reporting, and better integration of security tools with communication channels used

in an organization. Additionally, other researchers recognized that *reducing the effort required to learn or apply security (D24)* is a relevant driver that organizations should not overlook. For example, Fischer and Grossklags [57] proposed encouraging developers to write secure code by providing them with reminders and recommendations that prioritize security. In this way, developers would be capable of making safer choices that lead to writing more secure code. They performed two experiments that nudged developers while copying/pasting code from Stack overflow and searching for code snippets in Google.

A few researchers have also highlighted that it is important for organizations to *integrate tools into the development workflow (D25)* [47] and *provide developers with reminders (D22)* [44], [57]. For instance, Rauf et al. [44] emphasized that developers often add security as an afterthought and forget to give attention to secure coding practices. Therefore, there is a need to remind developers about security concerns while developing software. However, Braz et al. [56] highlighted that during code reviews, developers, despite receiving a tailored security checklist as a reminder, they can not find more vulnerabilities than when are just instructed to focus on security issues. Additionally, Mokhberi and Beznosov [47] acknowledged that a lack of integration with the development environment becomes a deterrent for developers to use security tools and reduces their engagement with security practices.

Facilitating Developers' Acquisition of Security-Specific Skill Sets. Several researchers have highlighted three vital drivers to motivate developers to write secure code: having *confidence in their technical abilities (D32)*, an *awareness of the necessary security skill set (D33)*, and *accessibility to learning resources (D26)*. For instance, Mokhberi and Beznosov [47] pointed out the role of personality as one of the human dimensions of developers' challenges in engineering secure software. Lack of confidence and false confidence are reasons developers mistakenly believe that their code is secure. Thus, they are unable to recognize vulnerabilities in their code. Additionally, Votipka et al. [54] acknowledged that the primary reason why teams do not implement security is due to a lack of knowledge and, above all, experience in different types of vulnerabilities. Furthermore, Sajwan et al. [51] emphasized that organizations usually employ traditional training resources and methods that developers do not feel are practical and actionable. Most of these learning resources focus on policies and protocols, reading, watching videos, or office conversation by either internal teams or external parties.

Other researchers have highlighted four essential drivers that influence developers in the adoption of security practices: *providing security education (D28)*, *having non-technical skills (D31)*, *creating and participating in communities of practice (D30)*, and *fostering hands-on learning, self-teaching, and osmosis (D29)*. For instance, Weir et al. [55] pointed out that security-related workshops facilitated by managers appear more effective than those facilitated by developers or security specialists. Additionally, Haney et al. [33] emphasized the relevance of having interpersonal skills, in particular, communication skills for dealing with all stakeholders involved in a security issue. Furthermore, researchers have reported that knowledge sharing is crucial for learning security practices. Developers learn the best from talking to other people in their teams as they can learn more technical skills while applying existing knowledge [51]. Researchers have also emphasized the crucial role of peer-based learning as an effective method to learn security practices. Developers usually perceive mentoring as an effective way to understand the rationale behind threats and techniques to mitigate them [55]. Interestingly, no previous studies have pointed out the essential role of *using security practices as learning tools (D27)*.

Comparing and Contrasting Drivers Identified in Our Study With Current Literature. In the above, we compared the drivers from our study with those found in the literature we reviewed.

In particular, the Mokhberi and Beznosov's [47] recent systematic literature review is the closest to our work. They presented a set of 17 areas of challenges across three dimensions: human, organizational, and technological. Their results align with 28 of our drivers, but they mentioned two factors that did not emerge from our findings. First, sometimes *responsibilities and roles defined in the software teams might conflict with applying security*, and therefore, developers will not follow security practices. Second, *developers may misuse APIs/libraries*, which leads to decreasing security in an application, making it easier to exploit. Another study related to our work is the systematic literature review conducted by Rauf et al. [44]. The authors presented a catalog of factors that influence developers' security behavior. Their work introduced 17 internal factors and 11 external factors analyzed from three different perspectives: *knowledge deficit*, *attention deficit*, and *intention deficit*. Their set of factors aligns with 19 of our drivers, and they did not mention any other drivers that we did not find in our study.

As we indicated before, three of our 33 drivers have not been identified in the literature we reviewed. They are *D17: awareness of the influential role of the industry type in developers' disposition towards security compliance*, *D20: consideration of tool constraints on developers' autonomy*, and *D27: using security practices as learning tools*. We were surprised that these drivers did not appear in any of the 15 papers we reviewed (two of which were quite comprehensive systematic reviews). In the related literature, we did not find any studies that highlighted how developers' disposition towards security compliance is highly influenced by, what the general industry mindset considers relevant, in terms of security. For example, developers working in the game industry or chip manufacturing tend to be more reluctant to adopt software security practices because the entire industry focuses on security, not at the application level but the infrastructure level. Additionally, researchers have given less attention to developers' perceptions regarding how the use of tools for security could affect their autonomy. Therefore, tools can become a deterrent to adopting security due to the restrictions imposed on developers' workflows. Moreover, we did not find related literature that explores the effects of security learning when performing security practices, even though the feedback collected while implementing a security practice can be a tremendous learning tool for developers. Section IV describes these drivers in more detail.

VI. Discussion

In this section, we review the implications of our findings. First, we discuss the implications for organizations, and our recommendations for developers and security specialists when adopting or advocating security practices. Then, we discuss how researchers can use the power of behavioral theories to understand and frame research on software security.

A. Implications for Organizations

Since organizations play the most crucial role in fostering the adoption of software security practices, our study pointed to several recommendations for them. In the following, we discuss how they can use the DASP framework as a diagnosis tool to help identify which aspects of their security practices could be improved. Then, we discuss what organizations need to consider when understanding the perspectives of developers and security specialists, and additional considerations they should be aware of when applying the DASP framework.

1) How to Use the DASP Framework: Organizations interested in promoting software security practices across different teams and stakeholders should carefully consider developers' behaviors and identify their perceptions and attitudes regarding their capabilities, opportunities available to them, and their motivations for adopting software security practices. Based on our findings, we propose DASP, a framework that consists of a comprehensive list of 33 drivers that represent what needs to change or happen so that the adoption of software security practices by developers occurs. In Table V, we present the complete list of drivers. Following the three stages of the Behavior Change Wheel (BCW) approach (see Fig. 3), organizations should use the drivers as the cornerstone of *stage 1*. These drivers become the starting point for designing interventions to foster an effective organizational security culture and motivate developers to write secure code.

To complete stage 1's goal, organizations need to select from our set of 33 drivers the ones that are significant in the organization's context. To identify those drivers, organizations should conduct focus groups or semi-structured interviews, considering the perspectives of most stakeholders, from developers to managers. In the case of big corporations, where collecting data could be challenging due to the significant number of stakeholders involved in ensuring a secure product, another potential mechanism is to conduct a survey questionnaire. A survey could help collect the level of agreement or disagreement concerning the drivers influencing stakeholders to adopt software security practices. The survey results or interviews will highlight a subset of drivers appropriate to the organization's context. In this way, an organization can proceed with stages 2 and 3 of the BCW, as depicted in Fig. 3.

2) What Organizations Need to Take Into Account When Understanding Developers' Perspectives: The drivers identified by our participants revealed several attitudes and beliefs that are useful for organizations and researchers to understand the developer mindset behind software security practices. For instance, drivers *D8*, *D15* and *D21* point out that developers often perceive the *disadvantages* of adopting security more quickly than the immediate benefits, e.g., time-consuming, delay feature delivery, add restrictions to their workflow, etc. Developers' mindsets typically have the *time-pressure* concern as their goals focus on delivering features. If security is not baked into the development pipeline, it will usually be *overlooked* if there are competing priorities *(D2, D25)*. Additionally, developers are influenced to adopt security practices when other team members embrace the same practices *(D6)*. However, *management* becomes a *deterrent* if they do not support developers' proactiveness towards security *(D1)*. When a security role is present on the team, developers usually *ignore* or *delegate* security practices as they consider those practices outside their core responsibilities *(D3)*.

Furthermore, driver *D32* highlights that developers might self-assess their security knowledge and capabilities with high scores. Extensive development experience can produce *false confidence* in their ability to effectively conduct a security practice without knowing the necessary security skill set required *(D33)*. Finally, most developers who are passionate about security have experienced security exploits in the past *(D13)*. The *fear* of suffering another attack is their primary motivation to keep their security knowledge up-to-date and advocate security concerns among peers *(D14)*. External events that impact *developers' emotions* have a high likelihood of sticking in a developer's mind for a longer period of time and will shape their attitude towards understanding and evaluating future risks *(D13)*. By default, it is relevant to acknowledge that it is not part of the developer's mindset to *think upfront about risks* and what could go wrong when customers misuse the applications they develop *(D12)*. Therefore, it is essential that any corporate security training highlight these potential scenarios *(D28)*.

3) What Organizations Need to Take Into Account When Understanding Security Specialists' Perspectives: The set of drivers reported by our participants pointed out that developers and security professionals have different mindsets. For instance, driver *D5* indicated that some of the frictions between security teams and developers occur because security specialists are usually more *pessimistic* and they tend to prioritize risks over product features. Moreover, since security is the day-to-day work for a security professional, they usually employ *technical communication* that developers typically are not familiar with *(D10)*. As a result, their perspectives concerning the impact of patching a security vulnerability might differ considerably from developers or engineers. Finally, since a security professional's primary focus is on identifying potential risks/threats and complying with security guidelines, their biased perspective might make it more challenging to see the overall impact of any change on the application *(D12)*. This situation becomes a threat in the organization when security teams analyze the trade-offs of applying security without the collaboration of developers and software architects *(D5)*.

4) Additional Considerations Organizations Should be Aware of When Applying the Framework: Our participants emphasized the following considerations during the *member-checking* sessions.

A Security Diagnosis Should be Conducted Periodically. Using the COM-B model as a diagnosis tool implies identifying what drives developers to adopt software security practices.

However, practitioners' perceptions, attitudes, and motivations can change over time due to external factors or events. In this scenario, the applicability of specific drivers to the current organization's context could be outdated. Therefore, it is essential to acknowledge these potential changes and periodically monitor them to ensure that the strategies or interventions are not based on inaccurate facts or observations.

A Just Enough Security Approach for Startup Companies. In the case of startup companies, the concern around adopting software security practices when developing a product-market fit is only addressed if the targeted consumer demands it. Otherwise, other non-functional requirements will have higher priority *(D2)*. Security specialists recommend startups with limited resources focus on delivering a *just enough* secure product, and if required, invest in training developers in security practices instead of hiring specialized resources. If available, external incentives *(D7)*, such as IRAP [17] in Canada, may be beneficial to introduce security testing practices provided by external parties to guarantee a certain level of quality in terms of security. This opportunity will also help developers acknowledge their typical security mistakes and which topics to focus on in their self-directed security learning process *(D29)*.

Non-Technical Skills Matter. Practitioners highlighted how it is essential to be aware of the security skill set required to implement security in the software development pipeline *(D33)*. Although most of our participants' first thoughts pointed out several security-related technical skills, they also mentioned different challenges related to interpersonal skills *(D31)* that, surprisingly, are usually not discussed in organizational security training. And when overlooked, it could seriously impact how security is handled within the organization, especially when multiple stakeholders are involved in security decisions and product development. Moreover, most security challenges are indirectly related to conflict management, negotiation skills, communication ability, and empathy. Therefore, we encourage organizations to include topics related to non-technical skills as part of any security training program *(D28)*, which is helpful in the context of security and boosts collaboration and productivity in the company.

B. Recommendations for Developers When Adopting Security Practices

In addition to organizational recommendations, the drivers identified in our study point to specific recommendations for developers:

- Passionate software developers care about the quality of their code. Security is an essential quality aspect of any software product, and its adoption gradually grows with experience. In addition, when learning a new software technology, e.g., a programming language or a framework, developers should take the necessary time to learn how to use it securely, not just assume that default configurations or standard ways to use it are secure *(D32, D33)*.
- If security is not part of the organizational culture, this can be an excellent opportunity to advocate for baking security into the development workflow *(D2, D25)*. Developers should look for sponsorship at the management level. With management's support, any effort towards security will be more straightforward and significant *(D1)*.
- When advocating for security, there can be some reluctance to change *(D4)*. Different stakeholders have different perspectives and priorities regarding security *(D19)*. To advocate for security more effectively, developers need to acquire the ability to translate security threats into technical and business risks. Arguments based on risks are more compelling and easier to understand *(D31)*.
- The adoption of security practices has an undeniable social connotation *(D6)*. It is recommended that developers start advocating for security among their peers. Raising a collective need or concern for security from engineering teams will significantly impact management roles more than any individual approach.
- Adopting security in the software development pipeline introduces new policies and restrictions around tools, frameworks, and open-source components *(D21)*. Developers should recognize that having full autonomy in technical decisions will cause a chaotic development environment, making it challenging for any stakeholder to ensure a secure development workflow and product *(D20)*.
- Developers should leverage their current working environment to customize their learning path. The presence of a security-specific role in the organization willing to mentor through peer-to-peer learning or osmosis is a practical and valuable learning approach *(D4, D29)*. For example, some realistic scenarios are developers shadowing a security specialist or being indirectly involved in fixing security vulnerabilities.
- Developers should take advantage of software security practices conducted within the organization *(D27, D29)*. These practices are an important opportunity for developers to follow a learning-by-doing approach, master software security skills, and receive actionable feedback to improve their development practices, e.g., through security code reviews.
- We recommend developers participate in communities of practice led by software security professionals *(D30)*. They are an active community in the industry, continuously organizing meet-ups, conferences, and workshops where they share valuable security resources and their experiences dealing with security vulnerabilities.
- We recommend developers use software security practices such as code reviews to self-assess their software security skills *(D27, D32)*. Feedback from security code reviews can help developers identify flaws in their coding practices and suggest topics to include in an organizational security training program *(D28)*.
- Confidence in performing software security tasks is an essential aspect that developers achieve through proper training and experience. However, developers should be aware that overconfidence is a deterrent to improving

[17] https://nrc.canada.ca/en/support-technology-innovation/about-nrcindustrial-research-assistance-program

their security practices *(D32)*, assimilating feedback, and maintaining the knowledge required to deal with security exploits that are getting more sophisticated and harmful.

C. Recommendations for Security Specialists When Advocating for Security

The drivers we identified in our study also suggest a number of recommendations for security specialists:

- The developers in our study perceived that most security guidelines are abstract and not developer friendly *(D10)*. We suggest security specialists facilitate the comprehension of security guidelines by highlighting the relationship between security threats, non-compliant code examples, and compliant solutions. In other words, when designing security guidelines, security professionals should use a technical vocabulary that developers are more familiar with.
- Developers typically access Q&A forums, e.g., Stack Overflow[18], to gather knowledge, share expertise, and address security and development concerns *(D30)*. Therefore, security professionals should proactively approach Q&A forums to get one step closer to developers' communities. In these forums, security specialists have the opportunity to advocate for security while interacting with developers by building awareness of security tools *(D9)*, potential security threats *(D12)*, and the risks of overlooking secure coding practices *(D14)*.
- Developers demand tailored security training that reflects their particular information needs and software development context *(D28)*. Developers perceive generic security training as too abstract, time-consuming, and ineffective. When designing security training to meet developers' needs, security teams or security specialists should identify developers' typical security mistakes through their interactions with engineering teams *(D27)*. For example, the interactions during security code reviews can be a valuable source of information.
- Developers who are passionate about security are often self-taught software security practitioners. However, the amount of resources and topics to learn is typically overwhelming *(D26)*. Therefore, security teams should facilitate developers' learning process by allowing them to know the rationale behind the compliance with security guidelines *(D15)* and the necessary skills to tackle their typical security mistakes *(D33)*. In this way, developers will learn faster, remember security guidelines more easily, gradually build confidence in their technical skills, and reduce the chance of repeating the same security mistakes *(D24)*.

D. The Power of Behavioral Theories for Software Security Researchers

Our work is related to developer-centric security research. Since software development is intensely human driven, researchers must consider developers' behaviors, attitudes, beliefs, perceptions, and motivations to introduce any positive change in software engineering practices. Our study highlights the opportunity for the software engineering research community to analyze software security challenges through the lens of behavioral science theories. Most efforts from industry and academia in the area of software security has focused on providing exceptional security standards, sophisticated security tools, free online learning resources, understanding the effectiveness of having specific security roles in the engineering teams, and encouraging shifting security to the left or earlier stages of the software development pipeline. However, little attention has been dedicated to allowing organizations to design effective interventions to foster the adoption of software security practices among developers.

Additionally, our work exposes a holistic view of all behavioral aspects that affect how developers adopt a target behavior, such as *capabilities*, *opportunities*, and *motivations*. Developers will be willing to adopt a new practice if they feel confident and capable of doing it, have the right opportunities and conditions in their work environment, and feel motivated to perform the target behavior. Our study opens the door to further research to introduce behavior change techniques to understand developer and other stakeholder behaviors. This knowledge will serve as the foundation to design interventions to motivate them to adopt software security practices.

VII. Threats to Validity

In the following section, we address the validity of this study in the context of qualitative research [24], [25].

A. Transferability

Transferability is the degree to which we can transfer our results to other contexts. Our study was based on semi-structured interviews gathering the experiences of 28 software engineers and security specialists. Given that their experiences, companies, technology stacks, and business domains varied considerably, the drivers identified in our study should fit most software development organizations. However, we did not include open-source developers in our work, geographical and cultural determinants to analyze our data, such as cultural differences in power distance and individualism-collectivism [59]. In addition, we did not consider group dynamics that might influence security adoption, such as organization identity, organization climate, and culture [60]. Moreover, our study also points out the critical influence of the management role in security adoption. Although two of our engineering participants had a management role, we cannot claim their insights will apply to any industry type. Therefore, we suggest future research to understand whether the open-source community considers our findings relevant for their context, how geographical, cultural, and group dynamics-related factors may influence developers' adoption of security practices, and extend the scope of the study to include the perspectives from managemen roles.

[18] https://stackoverflow.com/

B. Credibility

Credibility concerns whether the research findings are correctly drawn from the original data. We applied four strategies to ensure credibility: (a) the list of drivers was iteratively developed by two researchers and examined by an expert reviewer in each iteration. In addition, two researchers performed the open coding of the transcribed interview data. At the beginning of the open coding process, two open coding iterations were conducted to align the perspectives of both researchers. After achieving at least 75% inter-rater agreement, both researchers started coding independently, then (b) the set of drivers, categories, and findings were discussed several times between all the authors of this paper to mitigate bias from any particular researcher involved in the study, (c) the associations between drivers and the COM-B model components were performed by the first author of this paper and validated through several discussion sessions with two domain experts (fifth and sixth author of this paper) to mitigate any potential bias, and (d) the drivers were validated through 12 individual member-checking sessions.

C. Confirmability

Confirmability is the degree to which other researchers can confirm the findings. We do not have the participants' permission to share the transcriptions of the interviews. However, we tried to show as much evidence as possible for each driver by quoting participants when describing our results. Our interview script is available in our online appendix [58].

VIII. Conclusion

Adopting software security practices is a significant concern for any industry. The exponential increase in security vulnerabilities exploited by malicious hackers pushes the need to understand why software security is neglected and why developers persist in introducing security flaws into their applications [1]. In this study, using the lens of the COM-B model, we systematically explore what needs to change or happen so the adoption of software security practices occurs. As a result, we propose DASP, which consists of a comprehensive set of 33 drivers that describes the software security adoption phenomena. Our work is the first to introduce a behavioral change approach to understand developers' behaviors when adopting software security practices. Using DASP as a starting point, we foresee that organizations will be able to design appropriately geared interventions following the Behavioral Change Wheel framework. We hope our study insights will help organizations help developers write better, more secure code, ensuring a reliable and secure software product.

Acknowledgments

The authors would like to thank the 28 interviewees for their availability in this study, Dr. James Gibson for his remarkable insights in behavioral psychology at the beginning of our study, and the members of the CHISEL group at UVic for their invaluable feedback. We also acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

[1] NIST, "National vulnerability database," 2021. Accessed: Oct. 4, 2021. [Online]. Available: https://nvd.nist.gov
[2] State of Software Security - Volume 11, 2021. Accessed: Oct. 5, 2021. [Online]. Available: https://www.veracode.com/blog/research/announcing-state-software-security-v11-open-source-edition
[3] The Daily Swig. Accessed: Oct. 5, 2021. [Online]. Available: https://portswigger.net/daily-swig/malicious-hackers-are-exploiting-known-vulnerabilities-because-organizations-arent-quick-enough-to-patch-report
[4] J. van der Pligt and M. Vliek, *The Psychology of Influence: Theory, Research and Practice*, Evanston, IL, USA: Routledge/Taylor & Francis Group, 2017.
[5] C. I. Hovland, I. L. Janis, and H. H. Kelley, *Communication and Persuasion: Psychological Studies of Opinion Change*, New Haven, CT, USA: Yale Univ. Press, 1953.
[6] M. Fishbein and J. Ajzen, *Belief, Attitude, Intention and Behavior: An Introduction to Theory and Research*, Reading, MA, USA: Addison-Wesley, 1975.
[7] N. K. Janz and M. H. Becker, "The health belief model: A decade later," *Health Educ. Quart.*, vol. 11, no. 1, pp. 1–47, 1984.
[8] D. Kahneman and A. Tversky, "Prospect theory: An analysis of decisions under risk," *Econometrica*, vol. 47, pp. 263–291, 1979.
[9] F. Strack and R. Deutsch, "Reflective and impulsive determinants of social behavior," *Pers. Social Psychol. Rev.*, vol. 8, no. 3, pp. 220–247, 2004.
[10] A. Bandura, "Self-efficacy mechanism in human agency," *Amer. Psychol.*, vol. 37, no. 2, pp. 122–127, 1982.
[11] S. Michie, M. M. van Stralen, and R. West, "The behaviour change wheel: A new method for characterising and designing behaviour change interventions," *Implement Sci.*, vol. 6, pp. 1–12, 2011.
[12] S. Michie, L. Atkins, and R. West, *The Behaviour Change Wheel: A Guide to Designing Interventions*, 1st ed., London, U.K.: Silverback, 2014.
[13] F. Barker, L. Atkins, and S. de Lusignan, "Applying the COM-B behaviour model and behaviour change wheel to develop an intervention to improve hearing-aid use in adult auditory rehabilitation," *Int. J. Audiol*, vol. 55, pp. S90–S98, 2016.
[14] M. Alshaikh, H. Naseer, A. Ahmad, and S. B. Maynard, "Toward sustainable behaviour change: An approach for cyber security education training and awareness," in *Proc. 27th Eur. Conf. Inf. Syst.*, 2019.
[15] E. R. Bull et al., "An organisational participatory research study of the feasibility of the behaviour change wheel to support clinical teams implementing new models of care," *BMC Health Serv. Res.*, vol. 19, no. 1, pp. 1–12, 2019.
[16] E. A. Fulton, K. E. Brown, K. L. Kwah, and S. Wild, "StopApp: Using the behaviour change wheel to develop an app to increase uptake and attendance at NHS stop smoking services," *Healthcare*, vol. 4, no. 2, 2016, Art. no. 31.
[17] F. S. Los, H. F. van der Molen, C. T. Hulshof, and A. G. de Boer, "Supporting occupational physicians in the implementation of workers' health surveillance: Development of an intervention using the behavior change wheel framework," *Int. J. Environ. Res. Public Health*, vol. 18, no. 4, 2021, Art. no. 1939.
[18] E. Norris and D. B. O'Connor, "Science as behaviour: Using a behaviour change approach to increase uptake of open science," *Psychol. Health*, vol. 34, no. 12, pp. 1397–1406, 2019.
[19] G. J. Y. Peters and G. Kok, "All models are wrong, but some are useful: A comment on Ogden," *Health Psychol. Rev.*, vol. 10, no. 3, pp. 265–268, 2016.
[20] J. Webb, J. Foster, and E. Poulter, "Increasing the frequency of physical activity very brief advice for cancer patients. Development of an intervention using the behaviour change wheel," *Public Health*, vol. 133, pp. 45–56, 2016.
[21] C. Wilson and M. R. Marselle, "Insights from psychology about the design and implementation of energy interventions using the Behaviour Change Wheel," *Energy Res. Social Sci.*, vol. 19, pp. 177–191, 2016.
[22] J. W. Creswell, *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches*, Newbury Park, CA, USA: Sage, 2013.
[23] H. H. Hiller and L. DiLuzio, "The Interviewee and the research interview: Analysing a neglected dimension in research," *Can. Rev. Sociol. Anthropol.*, vol. 41, no. 1, pp. 1–26, 2004.
[24] E.G. Guba, "Criteria for assessing the trustworthiness of naturalistic inquiries," *Educ. Commun. Technol.*, vol. 29, pp. 75–91, 1981.
[25] I. Korstjens and A. Moser, "Series: Practical guidance to qualitative research. Part 4: Trustworthiness and publishing," *Eur. J. Gen. Pract.*, vol. 24, no. 1, pp. 120–124, 2018.

[26] I. Etikan, S. A. Musa, and R. S. Alkassim, "Comparison of convenience sampling and purposive sampling," *Amer. J. Theor. Appl. Statist.*, vol. 5, no. 1, pp. 1–4, 2016.
[27] S. E. Hove and B. Anda, "Experiences from conducting semi-structured interviews in empirical software engineering research," in *Proc. IEEE 11th Int. Softw. Metrics Symp.*, 2005, pp. 10–23.
[28] D. S. Cruzes and T. Dyba, "Recommended steps for thematic synthesis in software engineering," in *Proc. IEEE Int. Symp. Empirical Softw. Eng. Meas.*, 2011, pp. 275–284.
[29] G. R. Gibbs, *Thematic Coding and Categorizing*, Newbury Park, CA, USA: SAGE, 2007, pp. 38–55.
[30] J.M. Corbin and A. Strauss, "Grounded theory research: Procedures, canons, and evaluative criteria," *Qual Sociol*, vol. 13, pp. 3–21, 1990.
[31] M. B. Miles, A. M. Huberman, and J. Saldana, *Qualitative Data Analysis: A Methods Sourcebook*. Newbury Park, CA, USA: SAGE, 2014.
[32] A. L. Strauss and J. M. Corbin, *Grounded Theory in Practice*, Newbury Park, CA, USA: Sage, 1997.
[33] J. Haney, W. Lutters, and J. Jacobs, "Cybersecurity advocates: Force multipliers in security behavior change," *IEEE Secur. Privacy*, vol. 19, no. 4, pp. 54–59, Jul./Aug. 2021.
[34] J. Haney and W. Lutters, "Security awareness training for the workforce: Moving beyond "check-the-box" compliance," *Computer*, vol. 53, no. 10, pp. 91–95, Oct. 2020.
[35] S. Michie, L. Atkins, and H. L. Gainforth, "Changing behaviour to improve clinical practice and policy," in *Novos Desafios, Novas Competências: Contributos Atuais da Psicologia*, Braga, Portugal: Axioma - Publicações da Faculdade de Filosofia, 2016, pp. 41–60.
[36] S. Michie et al., "The behavior change technique taxonomy (V1) of 93 hierarchically clustered techniques: Building an international consensus for the reporting of behavior change interventions," *Ann. Behav. Med.*, vol. 46, no. 1, pp. 81–95, Aug. 2013.
[37] A. Bandura, *Self-Efficacy*, in V. S. Ramachaudran Ed., Encyclopedia of human behavior vol. 4. New York, NY, USA: Academic Press, pp. 71–81.
[38] A. Bandura, "Self-efficacy: Toward a unifying theory of behavioral change," *Adv. Behav. Res. Ther.*, vol. 1, no. 4, pp. 139–161, 1978.
[39] A. D. Stajkovic and F. Luthans, "Social cognitive theory and self-efficacy: Going beyond traditional motivational and behavioral approaches," *Org. Dyn.*, vol. 26, no. 4, pp. 62–74, 1998.
[40] R. W. Rogers, "A protection motivation theory of fear appeals and attitude change," *J. Psychol.*, vol. 91, pp. 93–114, 1975.
[41] K. Witte and M. Allen, "A meta-analysis of fear appeals: Implications for effective public health campaigns," *Health Educ. Behav. Official Publication Soc. Public Health Educ.*, vol. 27, no. 5, pp. 591–615, 2000.
[42] I. Ajzen, "The theory of planned behavior," *Org. Behav. Hum. Decis. Processes*, vol. 50, no. 2, pp. 179–211, 1991.
[43] C. Carver and M. Scheier, "Control theory: A useful conceptual framework for personality-social, clinical, and health psychology," *Psychol. Bull.*, vol. 92, pp. 111–35, 1982.
[44] I. Rauf et al., "The case for adaptive security interventions," *ACM Trans. Softw. Eng. Methodol.*, vol. 31, no. 1, 2022, Art. no. 52.
[45] R. Jones and A. Rastogi, "Secure coding: Building security into the software development life cycle," *Inf. Syst. Secur.*, vol. 13, pp. 29–39, 2004.
[46] R. Werlinger, K. Hawkey, D. Botta, and K. Beznosov, "Security practitioners in context: Their activities and interactions with other stakeholders within organizations," *Int. J. Hum.-Comput. Stud.*, vol. 67, pp. 584–606, 2009.
[47] A. Mokhberi, K. Beznosov, and S. O. K. Human, "Organizational, and technological dimensions of developers' challenges in engineering secure software," in *Proc. Eur. Symp. Usable Secur.*, 2021, pp. 59–75.
[48] T. Lopez, H. Sharp, T. Tun, A. Bandara, M. Levine, and B. Nuseibeh, "Talking about security with professional developers," in *Proc. IEEE/ACM Joint 7th Int. Workshop Conducting Empirical Stud. Ind. 6th Int. Workshop Softw. Eng. Res. Ind. Pract.*, 2019, pp. 34–40.
[49] J. Xie, H.R. Lipford, and B. Chu, "Why do programmers make security errors?," in *Proc. IEEE Symp. Vis. Lang. Hum.-Centric Comput.*, 2011, pp. 161–164.
[50] A. Poller, L. Kocksch, S. Turpe, F. Anand Epp, and K. Kinder-Kurlanda, "Can security become a routine? A study of organizational change in an agile software development group," in *Proc. ACM Conf. Comput. Supported Cooperative Work Social Comput.*, 2017, pp. 2489–2503.
[51] L. Sajwan, J. Noble, C. Anslow, and R. Biddle, "Why do programmers do what they do? A theory of influences on security practices," in *Proc. HATS Workshop Usable Secur. Privacy*, 2021, pp. 161–164.
[52] H. Assal and S. Chiasson, "Think secure from the beginning: A survey with software developers," in *Proc. CHI Conf. Hum. Factors Comput. Syst.*, 2019, pp. 1–13.
[53] H. Assal and S. Chiasson, "Motivations and amotivations for software security," in *Proc. SOUPS Workshop Secur. Inf. Workers*, 2018, pp. 1–4.
[54] D. Votipka, K. R. Fulton, J. Parker, M. Hou, M.L. Mazurek, and M.W. Hicks, "Understanding security mistakes developers make: Qualitative analysis from build it, break it, fix it," in *Proc. USENIX Secur. Symp.*, 2020, pp. 109–126.
[55] C. Weir, I. Becker, and L. Blair, "A passion for security: Intervening to help software developers," in *Proc. IEEE/ACM 43rd Int. Conf. Softw. Eng. Softw. Eng. Pract.*, 2021, pp. 21–30.
[56] L. Braz, C. Aeberhard, G. Calikli, and A. Bacchelli, "Less is more: Supporting developers in vulnerability detection during code review," in *Proc. IEEE/ACM 44th Int. Conf. Softw. Eng.*, 2022, pp. 1317–1329.
[57] F. Fischer and J. Grossklags, "Nudging software developers toward secure code," *IEEE Secur. Privacy*, vol. 20, no. 2, pp. 76–79, Mar./Apr. 2022.
[58] E. Larios-Vargas, O. Elazhary, S. Yousefi, D. Lowlind, M. L. W. Vliek, and M.-A. D. Storey, "DASP: A framework for driving the adoption of software security practices (V1.0)," 2022, *arXiv:2205.12388*.
[59] V. Taras, B. L. Kirkman, and P. Steel, "Examining the impact of culture's consequences: A three-decade, multilevel, meta-analytic review of Hofstede's cultural value dimensions," *J. Appl. Psychol.*, vol. 95, no. 3, pp. 405–439, 2010.
[60] B. Schneider, M. Ehrhart, and W. Macey, "Organizational climate and culture," *Annu. Rev. Psychol.*, vol. 64, pp. 361–388, 2012.