The Case for Adaptive Security Interventions

IRUM RAUF, MARIAN PETRE, THEIN TUN, and TAMARA LOPEZ, School of Computing and Communications, The Open University, UK
PAUL LUNN, The University of Manchester, UK
DIRK VAN DER LINDEN, Department of Computer and Information Sciences, Northumbria University, UK
JOHN TOWSE, Department of Psychology, University of Lancaster, UK
HELEN SHARP, School of Computing and Communications, The Open University, UK
MARK LEVINE, Department of Psychology, University of Lancaster, UK
AWAIS RASHID, Bristol Cyber Security Group, University of Bristol, UK
BASHAR NUSEIBEH, School of Computing and Communications, The Open University, UK, Lero-The Irish Software Research Centre, Republic of Ireland

Despite the availability of various methods and tools to facilitate secure coding, developers continue to write code that contains common vulnerabilities. It is important to understand why technological advances do not sufficiently facilitate developers in writing secure code. To widen our understanding of developers' behaviour, we considered the complexity of the security decision space of developers using theory from cognitive and social psychology. Our interdisciplinary study reported in this article (1) draws on the psychology literature to provide conceptual underpinnings for three categories of impediments to achieving security goals, (2) reports on an in-depth meta-analysis of existing software security literature that identified a catalogue of factors that influence developers' security decisions, and (3) characterises the landscape of existing security interventions that are available to the developer during coding and identifies gaps. Collectively, these show that different forms of impediments to achieving security goals arise from different contributing factors. Interventions will be more effective where they reflect psychological factors more sensitively and marry technical sophistication, psychological frameworks, and usability. Our analysis suggests "adaptive security interventions" as a solution that responds to the changing security needs of individual developers and a present a proof-of-concept tool to substantiate our suggestion.

This work was partially supported by UKRI/EPSRC (EP/P011799/1, EP/P011799/2, EP/R013144/1, and EP/T017465/1), NCSC, and SFI (13/RC/2094 and 16/RC/3918).
Authors' addresses: I. Rauf (corresponding author), M. Petre, T. Tun, T. Lopez, and H. Sharp, School of Computing and Communications, The Open University, Milton Keynes, UK; emails: {irum.rauf, marian.petre, thein.tun, tamara.lopez, helen.sharp}@open.ac.uk; B. Nuseibeh, School of Computing and Communications, The Open University, Milton Keynes, UK and Lero, University of Limerick, Republic of Ireland; email: bashar.nuseibeh@open.ac.uk; P. Lunn, The University of Manchester, Manchester; email: paul.lunn@manchester.ac.uk; D. van der Linden, Department of Computer and Information Sciences, Northumbria University, Newcastle-upon-Tyne, UK; email: dirk.vanderlinden@northumbria.ac.uk; A. Rashid, Bristol Cyber Security Group, University of Bristol, Bristol, UK; email: awais.rashid@bristol.ac.uk; J. Towse and M. Levine, Department of Psychology, University of Lancaster, Lancaster, UK; emails: {john.towse, mark.levine}@lancaster.ac.uk.

1049-331X/2021/09-ART9 $15.00
https://doi.org/10.1145/3471930

CCS Concepts: • **Security and privacy** → **Human and societal aspects of security and privacy**; • **Software and its engineering**; • **Human-centered computing** → *Empirical studies in collaborative and social computing*;

Additional Key Words and Phrases: Security decisions, developers, security goals, security interventions, cognitive psychology, social psychology, adaptive software engineering

ACM Reference format:
Irum Rauf, Marian Petre, Thein Tun, Tamara Lopez, Paul Lunn, Dirk van der Linden, John Towse, Helen Sharp, Mark Levine, Awais Rashid, and Bashar Nuseibeh. 2021. The Case for Adaptive Security Interventions. *ACM Trans. Softw. Eng. Methodol.* 31, 1, Article 9 (September 2021), 52 pages.
https://doi.org/10.1145/3471930

1 INTRODUCTION

"Secure coding" is a coding practice that ensures that the software does not contain known vulnerabilities. With the efforts of security and open-source communities, security practices and tools [76], security testing tools [64], security checklists [119], and vulnerability databases [54] are available to developers to facilitate secure coding. However, common code vulnerabilities continue to threaten the security of applications [54]. According to a 2018 Veracode report on software security [4], 70% of applications reviewed were not **OWASP (Open Web Application Security Project)** compliant, i.e., they did not address the most critical security concerns outlined in the security awareness document by OWASP, which is readily available to the developer community.

Common vulnerabilities, such as injection attacks and cross-site scripting attacks, also continue to appear in the **Common Vulnerabilities and Exposures (CVE)** list [54]. Thus, despite the efforts of the security community to provide security knowledge to developers, many developers continue to write code that is not secure.

Researchers have conducted various studies in recent years of developers' practices in producing secure software. We observe a division in the types of such studies. The first type identifies developers as end-users of security tools and studies what makes it hard for developers to use them. Examples include studies on misuse of cryptographic APIs by developers [60], insufficient advice given by the security tools to developers [69], and difficulties faced by developers in understanding security APIs [78]. The second type of research study investigates how developers act to improve security in software (security behaviour) and how they think and feel about security (attitudes). Examples of this second type include studies on how developers talk about security in the software development life cycle [23], how they use security tools in their workflow [148], how they socialize with other developers and adopt security practices [183], how they perceive security in code [179], and how they communicate about security in social fora [100]. While the first type investigates *usability issues* with security tools, and the second type studies *developers' behaviour and attitudes* with respect to security, these are not sufficient to explain why developers continue to write code with common vulnerabilities. The impact of cognitive processes is acknowledged within the literature at one level, for example with a recognition that developers are subject to heuristics and biases when considering security [67, 115]. The influence of social psychology literature is, however, under-explored. There is a need to widen this scope to explore the lessons learned in the cognitive and social sciences and to translate relevant insights to inform developer-centered security research.

This interdisciplinary research explores the security decision space of developers using theories from cognitive and social psychology. We widen the scope beyond developers' usability problems with security tools and their behaviour toward security, and we build a case for identifying the

particular needs of individual developers to help them to meet security goals. Security goals prescribe the intention to protect the assets of a system against harm—such as *confidentiality*, *integrity*, *availability*, and *non-repudiation* [71]. While ensuring complete protection of a system's assets against all kinds of attack may not be achievable, meeting security goals is often translated as avoiding known bugs that can lead to specific security breaches [125]. In doing so, we identify and answer two research questions.

RQ1: How can cognitive and social psychology explain the impediments to developers' efforts to meet security goals?

In answering this question, we look at two distinct areas of literature: the developer-centered security literature, and the cognitive and social science literature. **Developer-centered security (DCS)** is a term recently coined in security literature [124, 153]. DCS focuses on understanding how developers write secure code and facilitates the development of tools and techniques for the production of secure code by the developers [153]. Psychologists have conducted in-depth studies on the pursuit of goals by individuals [57], gaps between their knowledge and behaviour [59], and gaps between their intention and behaviour [154]. We attempt to map this multi-dimensional psychological account onto existing empirical work on security.

We first study some key psychological theories on why individuals do not meet their goals and identify key elements that influence behaviour. The key psychological theories are used as a lens on the security literature to seek explanations for behavioural challenges, such as why developers do not achieve security goals and why developers do not apply their security knowledge.

This is followed by a review of developer-centered security literature. This identifies, from the reported empirical studies, different factors that negatively influence developers' security behaviour. We distil a catalogue of factors that negatively influence a developer's security behaviour and attitudes by combining the knowledge from our analysis of developer-focused research studies and from the psychological explanations of human behaviour. This catalogue is built on the mapping between key cognitive elements that hinder behavioural goals and factors that influence a developer's security behaviour and attitudes.

RQ2: How can we design interventions to help developers reduce the obstacles in operationalising security goals?

To answer this question, we study the existing landscape of security interventions available to developers and analyse the needs of developers (investigated in RQ1) that these interventions address. We reviewed authoritative resources (detailed in Section 5.1) that reflect the "state-of-the-art" of security interventions. In the context of this work, "security interventions" refers to those actions and events, triggered either manually or autonomously, that occur during the code development process to promote the developer's security behaviour. Analysis shows that the design of current security interventions tends not to take sufficient account of the social and individual context that shapes developers' development decisions—and hence the interventions tend not to address the different security needs of different developers in different contexts. We suggest adaptive security interventions as a solution to address this gap.

This work highlights that, while the research community strives to provide better technical solutions to developers to identify vulnerabilities in code in an efficient manner, more effort is needed to provide interventions that fit the developer's immediate context, both social and technical, informed by behavioural science theories.

The article is organized as follows: Section 2 discusses security goals in light of the psychology literature and provides underpinning concepts for characterising different categories of impediments to secure coding. Section 3 presents these categories in detail, based on an

in-depth analysis of the existing security literature on developers' security behaviour. Section 4 answers RQ1. Section 5 discusses the existing landscape of security interventions available to the developer and highlights some ways in which these security interventions do not meet the security needs of developers. Section 6 answers RQ2 and builds the case for so-called adaptive security interventions by presenting a proof-of-concept tool. Section 7 concludes the article.

2 DEVELOPERS AND SECURITY GOALS

A developer can produce secure code by implementing security requirements and by avoiding, removing, or mitigating known vulnerabilities in the code. The security requirements of software may be explicit, i.e., part of system specification [71], or implicit, i.e., embedded in developers' heuristics or mindset [82].

Software security goals have been defined by Haley et al. [71] as the desire of stakeholders to protect assets of the system from harm. Such harm can negate security properties—such as *confidentiality*, *integrity*, *availability*, and *non-repudiation*. The security goals are then operationalised as security requirements, which, when implemented, constrain the system's behaviour.

From the psychology perspective, goals are defined as "mental representations of behaviours or behavioural outcomes that are desirable or rewarding to engage in or to attain" ([57], p. 470). Goals guide human actions and involve the willingness of individuals to act toward setting and attaining goals [57]. Looking through the lens of psychology, the desire to protect the system is a security goal, and the security requirements that operationalize security goals are seen as means to achieving security goals. Thus, the Haley et al. definition above can be interpreted as a specific instance of the more general psychological definition, as it identifies the mental representation (protection of assets—expressed as requirements) of the developer's behaviour (writing code using security requirements) and behavioural outcomes (code that preserves confidentiality, integrity, availability of assets, and non-repudiation) that are associated with positive affect (e.g., satisfaction or reward associated with the protection of assets).

Jones and Rastogi [82] argue that developers often fail to recognise or internalise security goals. In the Haley et al. definition, security goals belong to the software owner and are given as explicit security requirements (e.g., to penetration testers in the case study of Poller et al. [126]). When goals are operationalised as requirements, they are assigned as responsibilities to different agents such as humans, devices, and software [167]. However, Jones and Rastogi argue that, even if security goals are not assigned to the developer in the form of requirements, security goals should be a standard part of the developer's mindset and practice: "the basic tenets of coding secure applications ... should always be in the minds of the developers and should be reflected in the code that they write" ([82], p. 36).

Regardless of where the security goals originate, whether through operationalisation as a requirement [71] or as part of the developer's heuristics or mindset [82], ultimately, the developer must act to achieve them in the code. Goals are achieved by performing tasks, that is, sequences of actions that are executed by actors [98]. It is the transformation of the security goal into a personal goal that can be met through action [65] "at the desk" that results in goal attainment. Whenever a developer ignores a particular vulnerability, refuses to handle it, or is not able to secure the code against it, the developer is unsuccessful in achieving the security goal (i.e., in protecting the system from harm). Similarly, if the transformation of the goals into actions is not obvious, or is obstructed by other things, then the goals may not be achieved. For example, a developer's failure to handle an SQL injection vulnerability might *not* be a simple failure to take action (i.e., might *not* be that the developer did not prioritise the security goal and so did not act), but may instead be associated with the difficulty of identifying and taking appropriate action (e.g., the vulnerability is more complicated than the developer realises, the developer may not have the necessary resources, the

developer may have outdated information—for example, algorithms change continually, and the developer may not be familiar with the latest algorithms). Thus, "at the desk," the security goals become behavioural goals of the developer who must take required actions to satisfy the security goal. In this work, we focus on studying the reasons why developers do not meet security goals with the assumption that developers have internalised these security goals, i.e., they have become their own goals, regardless of their origin.

Van Lamsweerde and Letier [167] discuss *obstacles* (in the context of specifying and elaborating requirements) as conditions that prevent the achievement of functional goals; Van Lamsweerde [166] later extends this to the discovery of obstacles to application-specific security requirements. Although Van Lamsweerde and colleagues' focus is on the discovery of obstacles to the required function of the system to be developed, by analogy, in software development "at the desk," there are obstacles to developers' actions in pursuit of security goals. Hence, we shall call these *impediments* to acknowledge that Van Lamsweerde and Letier are referring to technical obstacles in software systems to be developed, and we are referring to cognitive and social obstacles for the developer in the development of software.

Van Lanweerde and Letier argue that, when we expect stakeholders (human or software agent) to satisfy a goal, without taking into consideration the obstacles that obstruct the required action, we over-idealize the goal. This over-idealization results in unrealistic (and hence unsatisfied) requirements. Poller et al. [127] carried a longitudinal study with developers in an organization to study how security fits into their routine. They observed that despite the security audits and security workshops delivered to the developers, developers tend to focus on daily tasks that take most of their attention. Although managers considered security as an implicit part of their working pattern, developers did not get time to work on it in detail. A Hewlett Packard white paper on studying security behaviour in the software engineering community [61] also highlights this point. In this whitepaper, authors provide an authoritative argument to encourage organizations into engaging employees in security behaviour. It stresses that asking employees for too much attention and effort will reduce the impact of organisational efforts to communicate the importance of security, and developers will "tune out" and focus on their primary work.

The focus of this article is to understand the impediments to developers meeting security goals in their personal, social, and technical context (and hence to consider what interventions might be introduced to help). The next section draws on literature from psychology (particularly the literature on why people fail to meet their goals) to characterise impediments to achieving security goals in a useful way.

2.1 Why do Individuals not Meet Their Goals? A Psychological Perspective

Despite knowledge of code vulnerabilities and how to avoid them, developers often write code with common vulnerabilities. For example, Oliveria et al. [115] showed that 53% of participants could not correlate their knowledge of relevant code vulnerabilities to their task-in-hand. Xiao et al. [183] also showed that many developers did not use security tools, despite realizing the importance of security in code. Studies by Naiakshina et al. reinforce the view that professional and student developers will not do security unless they are told to [107]. Tahaei and Vainiea reviewed the developer-centred security literature and observed that existing developer-centred research does not fully examine the issue of security as a "secondary" requirement [153].

Secure code development requires cognitive effort [115], and under constraints of time and resources developers struggle to keep security at the top of their priority list. Security is a secondary, rather than a primary task for most workers [123]. Secure behaviour is an investment that protects organisations and individuals from harm, but is not necessary to complete most production tasks. Because of this, it is important that security "fits" within the workday [87], or it

will be set aside [127]. In the context of software engineering, other concerns in the developer's environment, such as poor fit to the developer's job, other competing goals, or overload may conflict with security goals and override an organisational culture that prioritises secure behaviour [61].

These findings are consistent with the findings of psychologists: that awareness and knowledge alone do not suffice to trigger behaviour [57]. Both cognitive and social psychologists have considered discrepancies between an individual's knowledge and behaviour using different theories and in different application areas, and so we sought insight from that literature about factors that may influence secure coding behaviour.

This review was shaped by: (1) our motivation to study how individual developers code "at the desk"; (2) emergence of developers as a community that values social interactions [129]; and (3) our familiarity with literature on developer-centered security, which includes both studies of developers' security behaviour with cognitive tasks [116] and studies to investigate social influences on developers' security choices [183]. Hence, we wanted to be informed by both individual and social perspectives on factors that influence behaviour. Acknowledging that the field of psychology is mature and rich in theories, we were guided by the psychologists in the team in our selection of literature with a focus on the gap between goals and actions, to better understand *inaction.* Three key concepts emerged from our reading as key elements that influence behaviour: *knowledge*, *attention,* and *intention.* These are discussed in turn.

2.1.1 Knowledge. Cognitive psychologists work on the assumption that a person's existing knowledge and mental skills influence how the person perceives things [21]. Siergist and Cvetkovich [143] observed that when the individuals have personal knowledge, social trust does not significantly influence their judgements and risk perceptions. However, when individuals lack personal knowledge, they rely on social trust and managing authorities in perceiving risks of potential hazards. Similarly, we see that people who consider themselves to have required skills to protect themselves on the Internet exhibit safer cyber-security behaviours [33].

However, research suggests that despite knowledge of a task, individuals may not exhibit desired behaviour. Duncan et al. [59] suggests that activation of a goal can be hindered by the novelty of the required behaviour, lack of environmental cues, and multiple concurrent requirements. They define this as goal neglect behaviour: "disregard of a task requirement even though it has been understood and remembered" ([59], p. 265). The work of Kollmuss and Agyeman [90] and of Kennedy et al. [86] highlights the gap between people's knowledge and behaviour in different domains. Kollmuss and Agyeman [90] investigated the barriers to pro-environmental behaviours to understand the gap between people's behaviour and knowledge. Based on the analysis of some of the most influential frameworks for pro-environmental behaviour, they present a model for pro-environmental behaviour identifying internal, external, and demographic factors. Internal factors include motivation, knowledge, values, awareness, attitudes, and responsibilities. External factors include social and cultural factors, and institutional and economic infrastructure. Demographic factors include gender and years of education. Kennedy et al. [86] studied the knowledge-behaviour gap among clinicians based on a qualitative study of the gap between what individual clinicians know and how they behave. While they observed that the level of certainty and a sense of urgency promoted action, they also observed that clinicians used similar rationalizations to justify opposite behaviours. They also point out that the removal of barriers to actions may not always produce expected results and may produce different results in different contexts. Therefore, it is essential to understand the impact of different factors in different contexts. These works highlight that the gap between an individual's knowledge and actions often exists, and that the interplay of different factors may result in such a gap.

2.1.2 Attention. The limited capacity of the mind to process information is an accepted assumption in cognitive psychology [21]. While knowledge (related to the task-in-hand) plays a key role in goal attainment, mistakes and inaccuracies can arise from temporary lapses of attention [121]. Attention is "the extent to which incoming information is processed" ([57], (p. 469) and is "a state in which cognitive resources are focused on certain aspects of the environment rather than on others."[1] However, "attention is not a unitary concept" [120]. Although James [79] claimed that "everyone knows what attention is," the modern reality is far less simple but also considerably more exciting. Scientific theory and data overlap only partially with everyday usage of the term, but we now have a hugely sophisticated and detailed set of laboratory data and models on which to draw, as well as applications of theories across a range of real-world environments. The "commonsense" understanding of attention is amplified by concepts of selective attention and divided attention. *Selective attention* ranges from, on one hand, processing particular sources of input at the deliberate expense of others (e.g., thinking about one stream of events without distraction from others [168]), and, on the other hand, failing to notice salient events or stimuli in plain sight, illustrated by the phenomenon of inattention blindness [145]). *Divided attention* refers to multitasking, i.e., attending to multiple stimuli simultaneously [145].

Attention modulates human behaviour toward goal attainment [57]. Building on a literature review in social psychology, cognitive psychology, and neuroscience, Dijksterhuis and Aarts [57] suggest that an individual's *attention* is a strong influence on goal attainment, and that *consciousness*, i.e., "the ability to be aware of things" ([57], p. 468), is not. Norman also suggests need for attention at "critical action points" to avoid errors [114]. These "critical action points" are particularly important when people are confronted with novel problems that require conscious attention [114]. Attention and working memory are "closely interwined" ([26], p. 201) and both the processes are important to access relevant information required for a given task [26].

2.1.3 Intention. Another strand of work in psychology looks at the significance of intention in producing the desired behaviour [70]. Intention defines how hard an individual is willing to try and amount of effort an individual is willing to exert to perform a behaviour [13]. Ajzen's [13] *theory of planned behaviour* suggests that intention is a strong influence to behave in a specific way and has been widely adopted in research [14]. Ajzen's theory is still relevant after decades and is being extended and used in different behaviour domains [38]. The role of intention is also being actively explored in secure code development reported in the next subsection. Research also suggests that an intention-behaviour gap may exist. Tanner [154] notes that several factors constrain the behaviour of individuals even if they have the intention to act. Tanner identifies two classes of constraints that inhibit the actions of individuals: subjective factors and objective factors. Subjective factors, such as sense of responsibility and perceived behavioural barriers, define the individual's preference among alternative options. Objective factors, such as income level and place of residence, prevent the performance of particular behaviour alternatives. Blake [36] emphasizes the importance of identifying the constraints that limit an individual's desired behaviour. He identifies three obstacles to actions: *individuality*, *responsibility*, and *practicality*. *Individuality* includes barriers that are related to an individual's attitude and personality; *responsibility* refers to factors that limit individuals' understanding of how their actions may influence others and individuals' taking responsibility for an action; and *practicality* refers to social and institutional constraints that prevent people's action—in contrast to the previous two that affect people's intention to act.

2.1.4 The Impact of Social Processes on Goals. It has long been recognised that software development is a collective and coordinated process (e.g., Reference [52]). This means that, in addition to

[1] https://dictionary.apa.org/attention.

influence of cognitive psychological factors, the goal orientation of individuals can also be shaped by social processes. How developers acquire knowledge, what they pay attention to, and how they intend to act, are all subject to social psychological dynamics. The literature on these social processes is extensive and too large to be comprehensively reviewed in this article. We focus therefore on a brief review of three key social psychological dimensions that shape developer orientations toward security practice.

Social Influence: Because developers, and the code they produce, are embedded in a complex set of social relations, they are constantly exposed to the potential influence of others. This becomes particularly important under conditions of uncertainty, when we all become more likely to look to others for guidance on how to make sense of the situation we are in. This is what Cialdini [46] calls "social proof." The influence of others has a number of important dimensions. The first is informational influence, in which information seeking is often shaped by interpersonal ties. However, Turner and colleagues [161] have extended the analysis of informational influence beyond interpersonal relations to exploring the role of group memberships using the concept of **referent informational influence (RII)**. These influence concepts are useful for analysing the impact of repositories like Stack Overflow on the behaviour of individual developers. In addition, we need to be able to consider the impact of social norms on developer behaviour. Normative influence is usually understood as referring to the unwritten rules that govern behaviour for a group, society, or culture. Social norms tend to work in an implicit manner, where individuals' perceptions of normative behaviours are used to guide behavioural patterns and intentions. Social norms are particularly useful when thinking about promoting a greater emphasis on security in coding practice as they have been used extensively in behaviour change work in other fields – for example in health behaviour [131], environmental behaviour [113] and violence [111]. One of the key discoveries in this area has been the importance of the distinction between descriptive norms and injunctive norms [47, 149]. This is the difference between what we think people should do and what people actually do. In other words, injunctive norms reflect perceptions of what we think most others approve or disapprove of. We should be motivated to behave in ways that are normatively approved and avoid behaviours that are socially sanctioned. Descriptive norms reflect the perception of whether other people actually perform the behaviour—irrespective of whether it conforms to the ideal. This is the idea that "If everybody else is doing it, (or in the case of security behaviours, perhaps not doing it), then it must be a good/sensible thing to do." Evidence shows that the predictive utility of norms as determinants of behaviour can be improved by taking into account the distinction between the descriptive and injunctive norms [48].

Social Identities: A second way in which group processes shape security orientation is through the impact they have on how developers see themselves and act in the world. There is an extensive body of literature on the ways in which psychological identities shape behaviour, and how those identities are shaped by the social processes in which we are embedded. For example, the **Social Identity Approach (SIA)** [130] argues that, rather than having a single, fixed, identity, the way in which we see ourselves can be different at different times, with consequences for the way we behave. SIA suggests that individuals who develop code can sometimes define themselves in terms of a personal identity (an idiosyncratic or individually distinctive identity) and sometimes define themselves in terms of a membership of a social group (as a software engineer, for example). In fact, each individual has multiple social identities (including gender, national, cultural, political, religious, family, age, regional identities, etc.) that can become salient to them at different times.

There is now an extensive body of research that shows how social identity dynamics can affect decision-making and behaviour in organisations [74], in health [80], in politics [163], in sustainability [39], and so on. This suggests that, when it comes to software development, the same individual

can think about themselves in different ways at different times (during the code development process) and this is likely to affect the norms and the values that they draw on when setting or enacting (security-related) goals [8]. Understanding the dynamics of social identification in the collective and coordinated process of software development is therefore a key challenge for those interested in understanding why developers might not meet their goals.

Social Context: A third way in which social psychology can illuminate the study of secure code production is through its analysis of the role of social context. The importance of recognising the interaction between cognition and social context is well established [146]. However, social psychological research offers several important insights about how social context might impact on the goal orientations of developers. As we have already seen in our discussion of the role of social identities, social context is key to understanding which identities might be salient, and also why identities might change. As we move from physical location to physical location, so identities might change—and as the social composition of the people that we interact with changes, so identification may change also [162]. This dynamic model of identity change—based on changes in the immediate social context—helps to explain why the same individual might exhibit changes in their goal orientation over a very short time frame.

In addition, there are more general insights into the impact of social context on behaviour that are helpful. For example, Pettigrew shows how context can impact on behaviour at three different levels: micro, meso, and macro [122]. The micro level refers to the impact of individual-level factors such as personality. The meso level allows for the impact of face-to-face or situational factors. The macro level explores how thinking and behaviour is shaped by structural or cultural factors. Taken together, this approach orients us to the importance of conducting multi-level analyses on our examination of goal-orientated thinking and behaviour.

2.1.5 Cognitive and Social Psychology Insights. The field of psychology related to individuals' behaviour is vast and has applications in different domains. A comprehensive study of the many perspectives exceeds the scope of our work. With direction from our psychologist co-authors, we collect key insights from the cognitive and social psychology literature on factors that impede individuals from meeting their goals and observe that the gaps between an individual's goals and actions are a common concern in different domains and not a novelty in **developer-centered security (DCS)**. We postulate that lessons learned from relevant research in other disciplines can be applied to DCS. Thus, we have discussed here only a few of the key studies, enough to distil a framework to help us understand why developers do not meet security goals. Our review of psychology literature above identifies three key elements—knowledge, attention, and intention—that play a role in goal-attainment, but are not sufficient *per se* to ensure that individuals meet desired goals. These literature studies stress explicitly the need to identify reasons for inactions instead of actions. This is the focus adopted in this article: to investigate factors that impede the security behaviour of developers. Based on these three key psychological elements, the next section outlines categories of (security) goal impediments.

2.2 Categories of Security Goal Impediments

This section discusses knowledge, attention, and intention more specifically in the context of secure code development—showing how they provide insight both into what contributes to the achievement of secure coding goals, and into characterising and understanding impediments to secure coding.

2.2.1 Knowledge and Knowledge Deficit. Knowledge is an important determinant to completing a task correctly and influences the individual's behaviour positively toward a desired

goal [177]. Security knowledge—the information, understanding, and skills required to achieve a security goal—leads to security awareness. Security knowledge can be acquired from various sources via various means, including education and practice. Security-aware organizations often provide training for software developers, testers, and architects to increase their security knowledge [95, 103]. Software security concepts and practices are taught to students in a variety of ways, for example: integrating secure coding practices into the curriculum [155], security clinics [34], security tools for education [175], and specialized security lab set-ups [58].

The absence of, or insufficient, security knowledge can result in code with known vulnerabilities. We refer to this as *Knowledge Deficit*. Developers may not meet their security goals if they are using insecure information sources for help [10], have insufficient security knowledge [109], or are not sufficiently experienced with security tools [27]. If a developer does not have the right security knowledge, then even with security intention, attention, and action, security goals are not met. In such a case, either the developer does not write secure code because (a) the developer does not have the right knowledge to detect, identify, or mitigate the vulnerability, or (b) the developer uses incorrect or inappropriate security knowledge.

2.2.2 Attention and Attention Deficit. The previous subsection discussed that attention has a strong influence on goal attainment. Although "attention" has manifold implications depending on the context [120], for our current purposes, *attention* refers to how developers actively focus on and process specific information while coding.

Programming requires a high level of concentration [18]. Realizing the significance of developers' attention on the quality of developers' work, recently different mechanisms have been used to track the variation in developer's attention during programming. For example, EEG devices [18] are being used to track developers' concentration, keystroke dynamics are studied to detect stress among developers [89], and pupillography is used to collect information about developers' cognitive and emotional state [51]. Pair programming is also considered as an effective practice to eliminate distractions during programming and keep programmers focused on the programming task [144].

Lapses of attention caused by interruptions are detrimental to desired behaviour [177]. We refer to lapses of attention as *Attention Deficit*. Evidence from the literature shows that individuals are more likely to exhibit insecure cyber-behaviour involving phishing emails, exposing sensitive information, or downloading vulnerable applications when they are multitasking. Williams et al. [177] interviewed university employees who dealt with sensitive information to investigate what kind of computer-based tasks were perceived to be more disruptive. Their work suggests that, although multitasking is considered to be part of the job, interruptions such as email, phone calls, and face-to-face interruptions influenced their performance negatively.

In the context of secure coding, attention deficits occur when the developer has the intention to code securely but does not maintain focus on security in code as the coding task progresses. Developers must maintain their focus on secure coding, and relevant security knowledge must be part of their working memory to achieve security goals. These may be longer-term psychological objectives, but they must be represented and applied in-the-moment when decision forks with security ramifications are encountered in software. Developers process a large amount of information while coding; of all the information that the developer receives, only a limited amount is retained in working memory (which also draws on long-term declarative knowledge). Hence, some information "falls by the wayside." Limitations in the capacity of human working memory can lead to poor decision-making [84]. Such poor decision-making can exhibit itself when multitasking (divided attention), e.g., implementing logic of adding a new product by the user simultaneously—not just contiguously—with writing injection-safe queries. In such a case,

when striving to add functionality and security at the same time, the developer must maintain a focus on secure coding, and relevant security knowledge must be part of the developer's working memory to achieve security goals.

At another time, instead of working on two tasks at the same time, a developer may defer one task, releasing pressure on the working memory—but may later forget to recall it. This is consistent with work from psychology that suggests that compromise of an individual's attention, due either to unexpected interruptions or switching between tasks, results in either a time-cost to activate the primary task or in forgetting of the task [97]. A developer's mind may "wander" between the security goal and some other code goal, or some non-coding thought (divided attention). For example, to facilitate testing, developers may add information temporarily to the user interface; this may introduce security vulnerabilities if these changes are forgotten and not removed [85].

A developer may also experience attention deficit with respect to security goals if other objectives "muscle their way in" to decision-making (selective attention). For example, a developer may push security down the priority list—despite having sufficient security knowledge—if there are functional features to implement and the deadline is near [183]. Developers continuously process and represent information that is a crucial activity [75] and as such attention plays an essential role in keeping relevant information at the forefront of thinking, and hence reaching cognitive goals [28].

2.2.3 Intention and Intention Deficit. Intention indicates "people's self-instructions to achieve desired outcomes" ([142], p. 1) and "perform particular actions to attaining these outcomes" ([142], p. 1).

Woon and Kankanhalli [182] use a theory of reasoned action [70] and a theory of planned behaviour [13] to investigate the factors that influence the intention of professionals to develop secure applications. Their findings confirm that subjective norms and attitudes have a significant impact on the intention to develop secure applications. In recent work, Sallinen [136] uses a **protection motivation theory (PMT)** to instigate secure programming intention among developers, with positive results. Natural language processing techniques are also being used to understand and classifying developers' intentions by inspecting source code—distinguishing between benign and malicious source code [43].

While intention to code securely is a strong influence on developers' security behaviour, many factors may weaken the intention to practice secure coding. We refer to this as *Intention Deficit*. Intention deficit occurs when the developer is not steadfast in the intention to write secure code or make an extra effort to avoid vulnerabilities. There can be many reasons why a developer's intention to code securely is weakened, other than a "not-my-responsibility" or "do-not-care" attitude. Developers are often constrained by limitations in available resources, e.g., time and budget. When product owners do not consider security an important product feature [126], or customers do not understand its significance [92], developers often choose not to code securely. Secure coding practices require extra effort from developers on various fronts, for example, extra time to run security tools to detect/mitigate vulnerabilities in code [157] or extra effort to convince the client to pay for implementing security [126]. Developers may make a pragmatic decision to defer secure coding, with the intention of addressing security during refactoring; but then other factors may prevent refactoring, and hence the deferred secure coding is not undertaken. Intention deficit can result in technical debt. In (security) intention deficit, the security knowledge of the developer is not the key, as both developers who know how to code securely, and those who do not, choose not to code securely.

Table 1 presents the conceptual underpinning of the categories of impediments to security goals that are discussed above. While the security goal is an abstract concept and not directly measurable,

Table 1. Conceptual Underpinning of Impediments to Security Goal Categories

	Knowledge	Attention	Intention	Action	Security Outcome
Knowledge Deficit	✗	✓	✓	✓?	✗
Attention Deficit	✓?	✗	✓	✗	✗
Intention Deficit	✓?	✗	✗	✗	✗

This table illustrates how different configurations of cognitive elements (on the horizontal axis—knowledge, attention, intention) distinguish categories of impediments to achieving security goals (on the vertical axis—knowledge deficit, attention deficit, intention deficit). Key: ✓? = somewhat/sometimes present, ✓= present, ✗= absent/not sufficient.

security outcome is more specific and measurable, i.e., secure code that does not contain known vulnerabilities. Table 1 shows that a deficit of knowledge, attention, or intention will often result in insecure outcome. Developers perform security *actions* when they write secure code to avoid, detect, or mitigate a known vulnerability. These actions determine whether or not they meet security goals. The knowledge deficit row shows that if (security) knowledge is absent or insufficient (i.e., ✗), then even with intention and attention to code securely (i.e., ✓), developers may either not take security actions or if they do the actions might be faulty (i.e., ✓?), thus leading to an insecure outcome (i.e., ✗). The intention deficit row shows that even if security knowledge may be somewhat present (i.e., ✓?), the attention to security will not be present (i.e., ✗) if the intention to code securely is not present (i.e., ✗). Consequently, the developer does not perform security actions (i.e., ✗) resulting in an insecure outcome (i.e., ✗). The last row, attention deficit, suggests that although security knowledge may be somewhat present (i.e., ✓?), and the developer may intend to code securely (i.e., ✓), lapses of attention will prevent developer from taking security actions, leading to an insecure outcome (i.e., ✗). Thus, whenever any of these cognitive elements is absent or is not sufficient, the necessary actions will not be performed, and the resulting code will be insecure.

This section has discussed the relevance of these cognitive elements in the context of developer-centered security. The discussion so far has focused on what impedes developers from taking appropriate actions *despite* security awareness, and hence writing code with common vulnerabilities. The next section applies this taxonomy in a literature review of published research on the security behaviour of developers.

3 FACTORS THAT INFLUENCE DEVELOPERS' SECURITY DECISIONS

One of our research aims was to map between the psychology literature on impediments to behavioural goals and the software engineering literature on security behaviour of developers. This section describes and presents the results of an in-depth analysis of research on key elements that influence the security decisions of developers—we refer to these key elements "factors." The analysis produced a detailed catalogue of such factors (Appendix B: Tables 6–8), which is presented succinctly in Tables 2 and 3.

We studied the existing literature covering empirical studies of developers' security behaviours to conduct an overarching analysis of the "state-of-the-art" in this domain. The review was conducted using the guidelines suggested by Kitchenham et al. [88] to cover the relevant literature rigorously. Forward and backward snowballing approaches [180] were used to cover, as comprehensively as possible, the studies conducted in this domain in the past 10 years, i.e., 2009 to 2019. We consider this time interval reasonable, because research in this area is relatively recent. A narrative synthesis was conducted to answer our research question from primary studies [88]. Narrative synthesis was used due to the heterogeneity of different empirical methods used in the studies, their data analysis approaches, and variation in target audience [37]. Additionally, since we were interested primarily in identifying different factors that influence developers' security

Table 2. Summary of the Selection Process

	IEEE Xplore	ACM DL	SpringerLink	ScienceDirect	InternetSociety	Total
Search Results	413	2,193	5,875	1,717	-	9,320
After Reviewing titles/ keywords	20	300	40	30	-	390
After Manual Search	10	18	23	10	-	61
Snowballing	14	20	23	12	2	69
After reading abstracts and removing duplicates	12	16	10	5	2	43
Final Papers: after skimming/ reviewing	9	15	3	0	2	29

Table 3. Internal Factors, where, KD= Knowledge Deficit, AD= Attention Deficit, ID= Intention Deficit

No.	Internal Factors	Impediments to Security Goal	Citations
I1	Misconceptions	KD	[23, 108]
I2	Use of outdated information	KD	[108, 109, 148]
I3	False assumptions / Inferences	KD	[23, 148]
I4	Misplaced trust on frameworks / third-party API	KD	[108],[23]
I5	Lack of clarity about regulation	KD	[30]
I6	Lack of domain knowledge	KD	[106, 173]
I7	Lack of experience (with tools, especially security tools, APIs, programming languages)	KD	[27],[11, 78]
I8	Lack of awareness of security tools/ vulnerability	KD	[92, 148]
I9	Not identifying security blind-spots in tasks	AD	[115, 116, 148],[92, 173]
I10	Not handling cognitive load	AD	[115, 148],[85, 92]
I11	Developer's non-secure routines	AD	[23, 148]
I12	Lack of curiosity	AD	[116, 178, 179]
I13	Loss of focus on security	AD/ID	[23],[92, 126, 185] [108],[109, 115, 183]
I14	Requires extra effort	ID	[23, 29],[92]
I15	Not perceiving usefulness of secure practices	ID	[24, 30, 170, 178, 179] [183],[157],[92]
I16	Perceived lack of own security knowledge	ID	[24, 108, 179]
I17	Attitude of "someone else's responsibility"	ID	[23],[92],[101],[173, 185]

behaviour—and not in ranking them—we do not report on the quality assessment of individual articles. These limitations of the review are discussed in Section 3.2.

3.1 Methodology to Gather Primary Studies

A research question that drives this research is: *What factors influence the security behaviour of developers?*

To find relevant studies that can help us answer this research question, the following inclusion and exclusion criteria were defined:

Inclusion and exclusion criteria. The inclusion criteria for primary studies are:

- papers published in past 10 years, i.e., 2009 to 2019
- papers that studied developers with respect to secure coding behaviour.

The exclusion criteria include:

- papers not written in English
- papers that were previous work of a more recent and mature work by the same authors. For example, Reference [147] was not selected in favour of Reference [148], the more recent and mature work of the same authors.
- non-peer-reviewed papers, books, dissertations, and position papers
- papers that studied problems that security practitioners face in development teams and not the problems that developers face when developing software applications, e.g., References [72, 174]
- papers that studied the effect of a particular security intervention on developers' behaviour, e.g., the work of Nguyen et al. [112] and Gorski et al. [69]. Such works are discussed under security interventions in subsequent sections of this article.

Search String and Database. We first identified major keywords: developers, code, and security. We then identified alternative words for them as synonyms and finally combined them using Boolean operators. The final search string was:

```
{( (developer OR programmer) AND (secur*
    OR vulnerabilit* ) AND/OR
   (cod* OR program* OR tool))
< in abstract, keywords, and title > }
```

The selected search repositories are: IEEE Xplore, ACM Digital Library, **SpringerLink (SL)**, and **ScienceDirect (SD)**, as they cover almost all important workshops, conferences, and journal papers that are published. Table 2 shows the results generated by the search string in different databases. IEEE Xplore and ACM Digital library search is conducted on all metadata. The search results of Springer Link are refined for the following categories: English, Computer Science, Software Engineering, Articles—resulting in 4,997 papers. The search results of ScienceDirect are refined for Computer Science, focusing on the following publication titles: *Procedia Computer Science, Information and Software Technology, Journal of Systems and Software, Future Generation Computer Systems, Computers & Security, Computers in Human Behaviour*—resulting in 1,717 publications. All the available publication titles that could encompass research related to developer-centered security were selected.

The search results were filtered to find relevance to the inclusion and exclusion criteria by reviewing their titles and keywords. The first author read the abstracts of the shortlisted articles to select candidate studies. These studies were read in full, and using forward and backward snowballing [180] a few more articles were identified; i.e., we looked into the references of the articles that looked relevant (*forward snowballing*) and also looked at the papers that cited the selected studies (*backward snowballing*). We shortlisted 10 new studies, of which 2 were published by the Internet Society,[2] which was not previously on the list of publishers. The resulting studies were further shortlisted by reading their abstracts and removing duplicates. The results of each step are listed in Table 2.

We also used Google Scholar, because it indexes a collection of databases [181]. However, as the Google Scholar search was run *post hoc*, it produced studies already picked up from different databases and snowballing.

[2] https://www.internetsociety.org/.

For the last step, another author of the study reviewed the selection of the articles by the first author for inclusion after skimming. There were no conflicts of opinions on the studies included in the review.

Results. The final corpus comprises 29 research papers (Appendix A, Table 4).

Although we did not specifically look for studies in this area before 2009, our forward and backward snowballing did not yield any relevant work (i.e., empirical studies of factors that influence the security behaviour of developers) in this area before 2009. We consider it essential to provide an analysis of the existing body of knowledge at this point to harness the momentum of this decade of research and provide a better focus for future efforts to facilitate developers in writing secure code.

All 29 studies involved some empirical evidence, i.e., their findings were based either on direct observation of developers or the online resources they use, or on analysis of data available in online repositories such as Stack Overflow and app stores. Chen et al. [45], Fischer et al. [63], Egele et al. [60], and Yang et al. [187] studied online evidence of developers' interaction on online fora, and/or analysed software in online repositories for vulnerabilities. Acar et al. [12] studied online advice resources that developers typically use. The rest of the studies collected data directly from developers, using surveys, interviews, user studies, and controlled experiments.

The analysis was conducted using both inductive and deductive analysis [156]. Inductive methods were used to compile and collate the factors identified by different research studies. We gathered a raw list of 101 factors identified by the authors of the 29 studies. After consolidating the factors by the same author that had similar meanings, we reduced the list to 82 factors. After combining factors implying the same meaning by different authors and from different articles, we had a final list of 28 factors. Each of the consolidation steps had an element of independent coding, with each of two researchers reviewing the factors and looking for overlaps; then the two independently produced groupings were compared, and any discrepancies (of which there were few) were resolved through discussion.

We then employed deductive analysis in two stages to categorize the factors:

Categories of Security Goal Impediments: First, each of the 28 factors was categorized into at least one of the three categories of security goal impediments: **knowledge deficit (KD)**, **attention deficit (AD)**, or **intention deficit (ID)**. Tables 2 and 3 list these 28 factors and also show to which of the categories of security goal impediments they belong. We considered the implications of factors based on how the author(s) of the respective study discussed it or its meanings. (*N.B.* This categorisation, and dichotomising the world into extreme states (goal failure, and goal achievement) is a simplification for the purpose of mapping out the relevant problem space.)

Internal/external: Second, we distinguished between internal and external factors within each category, inspired by the work of Kollmuss and Agyeman [90] (discussed in Section 2.1). This approach takes a high-level perspective and identifies whether the factors arise primarily from the developer's internal world or from the outside world. Drawing on Kollmuss and Agyeman, we define internal factors as those that stem from an individual's knowledge base, personality traits, beliefs and attitudes, value systems, and habits. External factors are those that surface in the environment with which a developer interacts. These may stem from social, technical, economic, institutional, and/or cultural infrastructure surrounding the developer.

Table 2 lists all the internal factors, and Table 3 lists all the external factors.

Our understanding is that the factors influencing the decision space of developers is quite complex, and providing a high-level view with this categorisation illuminates this decision space. As pointed out by Kollmuss and Agyeman, making a distinction between internal and external factors is "to some extent arbitrary" (p. 248), because different factors influencing human behaviour are

broadly defined, *do not have clear boundaries*, and can be *interrelated*. Thus, we see that some of the factors in Tables 2 and 3 appear in more than one category (i.e., I13, E5, E6, and E11) and are shown as such in the tables. Consider, for example, the *loss of focus on security* (I13 in Table 2); this was categorized as primarily an internal factor, because the developer drifts from an intended focus on security because of other things demanding attention or taking priority in the moment. For example, the developer may lose focus on secure coding while "in a flow" to achieve functional correctness. However, the loss of focus may also be influenced by other factors, such as limited time and budget, i.e., *limited resources* [23] (E6 in Table 3). These implicit connections between internal and external factors may blur the distinction between internal and external factors when we reason closely about them. Our purpose is not to provide an exact categorisation of individual instances, but rather to draw out the different factors and to draw attention to both internal and external influences so they can be addressed collectively. This can help researchers and tool designers to address the challenges faced by the developers.

The detailed categorisation, giving the instances for each factor, is in Appendix B, in Tables 6–8. Tables 6, 7, and 8 list internal and external factors of knowledge deficit, attention deficit, and intention deficit, respectively, with their instances found in the existing security literature.

To illustrate the categorisation, we consider the following three factors, each associated with a different security goal impediment.

Outdated information is first categorised under *knowledge deficit*, as it occurs because the developer's knowledge is not current and is hence incomplete (e.g., developers using methods that are outdated [109]). Second, this is identified as an internal factor, because the impediment arises from limitations of the developer's own knowledge base [148].

Inability to recognize security blind spots in task is first categorized under *attention deficit*, because security blind spots may occur due to an oversight by the developer (i.e., lack of attention on corner cases [115]). Second, this is identified as an internal factor, because it occurs when a developer is not able to recognize a security vulnerability in code (e.g., a developer's inability to correlate the learned vulnerability to the working task [116]).

Lack of prioritisation of security features by stakeholders is first categorized under *intention deficit*, as it influences the developer's intention to code securely, because the effort is not valued by the stakeholders [126]. Second, this is identified as an external factor because it is caused by external stakeholders.

Some factors may influence more than one of the cognitive elements that characterise our taxonomy of security goal impediments. For example, the *absence of explicit expectations of secure coding (E5)* results in insecure code, as it does not direct developer's attention to secure coding, i.e., *attention deficit*. However, the *absence of explicit expectations of secure coding* may cause the developer to think that secure coding is not required in the given work, leading to *intention deficit*. In the former example, security thinking is not part of the developer's heuristics until reminded [115]; in the latter, developers expect security as an explicit requirement [108]. Similarly, *limited resources (E6)* appears under both attention deficit and intention deficit. In the former, overload combined with limited resources leads to loss of attention; in the latter, limited resources influence a decision not to prioritise security. We discuss the outcome of the categorisation in detail below.

3.2 Limitations of the Review

This review was intended to mine existing empirical research for factors that influence developers' security behaviour. The literature was gathered and analysed systematically. One potential limitation is the completeness of the collected literature; however, based on the systematic approach taken, we consider the literature representative, if not comprehensive. The analysis provides a well-grounded basis for further work that may extend or refine the collection of factors.

We drew evidence from studies that varied in methods and participants, and we compiled and interpreted that evidence in a way that exceeded the scope of the primary studies:

Different study designs: Some studies only report evidence collected directly from developers using empirical methods such as observations, interviews, surveys, user studies, and experiments (e.g., Naiakshina et al. reported an online study with freelance developers [108]). Some reported analyses of online resources in combination with interviews or surveys with developers (e.g., Nadi et al. analysed stack overflow posts and Github projects along with survey results from developers [106]). And some studies analysed only online repositories for evidence of how developers work (e.g., Chen et al. [45] analysed security-related Stack Overflow posts). Some conducted lab studies (e.g., the qualitative user study reported in the work of Naiakshina et al. [109], while others report their findings from studying developers in real environments (e.g., the case study reported by Poller et al. [126]).

Different study participants: As developers come from a range of programming, educational, and professional backgrounds, we report findings from different studies that study a range of developers. Some studies report findings of studies conducted with students [109], some study professional developers [24], and some study both professional developers and students [10].

Our intention was to be as inclusive as possible, and so we considered the variety of studies to be an advantage for our purposes. We did not undertake a quality assessment of the primary articles, because it was not necessary for our purpose; rather, our focus was on breadth. In reporting these different studies with a range of developers, we aimed to address how developers from different backgrounds are influenced by different factors—and to identify as many factors that influence secure coding behaviour as possible from the literature. However, each of the empirical studies on which we drew had its own threats to validity, which might in turn affect the reliability of this review. The analysis compiled and compared evidence between studies, which provided some mitigation of the impact of potential threats. However, as our goal was to map contributing factors as broadly as possible, and so the validity of an individual factor is less crucial. Nevertheless, the results should be viewed as informative, rather than definitive.

The review might be biased by our interpretation of the psychology literature, and hence by the categorization in terms of the three impediments to security goals (knowledge, attention, and intention deficit). We have tried to make the categorization reasoning explicit and to acknowledge the complexity of the factors with multiple categories where appropriate. Further, the categorization as internal and external factors, was, as discussed previously, "to some extent arbitrary." As discussed, our purpose was not to provide an exact categorisation, but rather to draw out the different factors and to draw attention to both internal and external influences so they can be considered, discussed, and addressed collectively.

3.3 Knowledge Deficit

The security knowledge of developers varies [108], and some security topics are considered more difficult than others by developers [186]. As discussed earlier, insufficient, erroneous, or misapplied knowledge—whether drawn from a developer's own knowledge base or from external resources—can prevent developers from taking proper action to handle a security vulnerability, regardless of their intention to code securely or their attention to it. This section discusses the internal and external factors categorised as contributing to knowledge deficit.

3.3.1 Internal Factors – Knowledge Deficit. Eight internal factors (rows I1–I8 in Table 2) were categorised as knowledge deficit and are discussed in turn.

Misconception (I1): Developers may have a false sense of security in code although the code may have some obvious vulnerabilities. In such cases, developers may have misconceived notions of

Table 4. External Factors, where, KD= Knowledge Deficit, AD= Attention Deficit, ID= Intention Deficit

No.	External Factors	Impediments to Security Goal	Citations
E1	Inadequate information sources	KD	[10, 45, 148] [63, 108, 109] [173, 178]
E2	Lack of information sharing among teams	KD	[173]
E3	Task complexity	AD	[92, 108]
E4	Lack of division of labour	AD	[126, 173]
E5	Absence of explicit expectation of secure coding	AD/ ID	[23, 108, 115], [92, 109, 185]
E6	Limited resources	AD/ ID	[23, 157, 178],[173], [78, 92, 185]
E7	Security tools not reachable	ID	[24, 179, 183],
E8	Lack of security culture in teams/ companies	ID	[24, 126, 179], [23, 78, 183]
E9	Lack of prioritisation of security features by stakeholders	ID	[78, 126], [157],[92], [78]
E10	Lack of social influence	ID	[108, 179],[183], [157],[101, 183, 185]
E11	Usability issues with security tools and APIs	KD/ ID	[178],[30, 148, 157], [106]

security. For example, some developers think of encryption and hashing as synonyms and some may reduce secure storage of password to a visual representation using Base64 [108]. Likewise, some developers do not have right mental models of security. These developers consider security as something that only comes with the use of secure functions like use of secure communication protocols and hence, do not consider vulnerabilities that come with implementation mistakes [23].

Use of outdated information (I2): Some developers rely on their existing knowledge to avoid security vulnerabilities. The problem with this approach is that at times either their knowledge of the vulnerabilities is outdated [108] or the code has been changed without their knowledge [148].

False assumption/inferences (I3): Some developers have incorrect assumptions that a certain vulnerability has been already handled securely [148]. In some cases developers may make assumptions about method names that are overloaded, for instance, in the study conducted by Smith et al. [148], developers failed to recognize a potential source of tainted data because its name confused the with a JUnit test case.

(Mis)trust of frameworks/third-party APIs (I4): Developer may have incomplete or wrong information about the framework they are using. For example, some developers fully trust the frameworks that they are using to handle security [23] or blindly trust code from a reputable source [115].

Lack of clarity about security regulations (I5): The security regulations, in place at different app publishing stores, serve as a check on security violations. Developers who are less aware of security regulations are more likely to make accidental errors that violate security regulations [11]

Lack of domain knowledge (I6): Developers often fail to identify vulnerabilities in code due to lack of sufficient domain knowledge [106]. In addition, developers often need to have a good understanding of attacker profiles and descriptions to tackle security vulnerabilities that developers often do not have [173].

Lack of experience with tools (especially security tools, APIs, programming languages) (I7): Lack of experience with the technology in hand [11] (I7) can hinder developers' efforts to write secure code. Baca et al. [27] reported that the number of security questions answered tripled with a combination

of the developer's security experience *and* experience with the security analysis tool [27]. Acar et al. also found a significant effect of Python experience on functional and secure results for Python-based tasks [11].

Lack of awareness of security tools and new vulnerabilities (I8): Although several security tools are available to handle different types of vulnerabilities, developers often struggle to find the right security tool to support their tasks [148, 178], while some developers are unaware of the vulnerabilities in their software [92] (I8). Conversely, many developers often find themselves struggling with keeping themselves up-to-date about a growing number of vulnerabilities [30].

3.3.2 External Factors – Knowledge Deficit. Three external factors (E1, E2, and E11 in Table 3) were categorised as knowledge deficit. Developers often use external resources that exist "in the world" and are part of the developer's broader development ecosystem, including information sources (both online and offline), other developers, and security tools. Knowledge issues that exist in these resources are beyond developers' control and responsibility, but they impede developers' efforts to meet their security goals.

Inadequate information sources (E1): Developers often seek advice from different information sources while working on a piece of code [10]; when these resources are inadequate, they contribute to knowledge deficit. These information sources may not always contain up-to-date security information [12] as new security threats and tools are emerging fast. Also, several of these information sources contain conflicting advice [109].

The influence of on-line social fora like Stack Overflow on the developer's community is highlighted by a number of researchers (e.g., References [10, 45, 63]). These works suggest that developers often follow insecure advice available in them or copy-paste insecure code snippets from them and publish them [63]. Official documentation and corporate guidelines address security concerns, but the level of detail presented in them is often insufficient for developers [12]. Interestingly, studies also showed that law and regulations regarding security were mentioned only in two on-line guides [12].

Lack of information sharing among teams (E2): Security in code is often affected by lack of knowledge sharing among different teams in an organization (E2), which makes it difficult to address vulnerabilities in code appropriately and in-time [173]. Developers may also be limited in their knowledge of how their mobile apps are being used (E1). This can prevent them from knowing actual exploits of their apps and from identifying usage patterns to provide on-the-spot updates for security [173].

Usability issues with security tools and APIs (E11): Security tools developed to facilitate developers in handling vulnerabilities may negatively affect developers' security knowledge instead of aiding it. This arises as a result of usability issues with these security tools (E11), for example, poor documentation, misleading defaults, and bad API designs [106].

3.4 Attention Deficit

(Security) attention deficit occurs when developers lose attention to secure coding (e.g., when they do not attend to it, do not attend to it fully, or lose attention to it), despite them being aware of security goals and having the intention to code securely. Different internal and external factors that are a part of a developer's personality, habits, security skills, and environment challenge the developer's attention to security.

3.4.1 Internal Factors – Attention Deficit. Internal factors that limit developers attention to secure coding include factors originating from developers' inability to identify security blind-spots in tasks, their personality traits and habits, and the challenge of keeping security in working memory in the presence of other competing factors.

Not identifying security blind spots (I9): Developers often find it cognitively challenging to detect security blind-spots in programming tasks (I9). Developers may have difficulty in correlating the vulnerabilities they know with the task-in-hand [115]. Some developers overlook certain program paths to true attacks while focusing on another subset of possible attacks [148], while some developers may fail to spot their own security errors due to a single point of view [173].

Not handling cognitive load (I10): "Security thinking requires cognitive effort" ([115], p. 1). Developers often find it difficult to remember which security vulnerabilities they have addressed and which they have ruled out [148]. Sometimes, to facilitate usability—especially when testing—developers may add security vulnerabilities that they later forget to remove and hence release their software with them in. Likewise, developers consider the practice of updating their dependencies as an extra effort and defer the task until they have more time [92].

Lack of curiosity (I12) and non-secure routines (I11): Developers' lack of curiosity as a personality trait (I12) and their non-secure routines (I11) can also encourage non-secure behaviour when they are developing "in a flow." Developers who take pleasure in leaving their comfort zone and exploring new security tools and practices are more likely to adopt security tools [179]. The lack of curiosity by developers [179], and their comfort with error-prone strategies even in the presence of reliable tools and strategies, [148] can lead to common vulnerabilities in code.

Loss of focus on security (I13): Attention provides a selective process to filter out irrelevant aspects of incoming information [57]. Developers' attention may be focused on other aspects of programming, and security may be added as an afterthought even when developers are asked to program securely [109]. Functional features compete for the developer's attention, and the developer may be tempted to overlook or postpone security considerations [115]. Even after realizing the importance of making security decisions, developers may concentrate on functionality to the neglect of security[172].

3.4.2 External Factors – Attention Deficit. Four external factors were categorised as attention deficit and are discussed in turn.

Absence of explicit expectations of security (E5): Developers often add security as an afterthought [109]. In the absence of explicit expectations from the stakeholders to code securely, developers often forget to give attention to secure coding practices. However, some developers are quick to add security when prompted for it [115].

Limited resources (E6): The limited availability of resources such as time and budget push security out of developers' working memory, even if developers have sufficient security-related skills [30].

Task complexity (E3): The complexity of the task-in-hand can also make it difficult for developers to identify security blind-spots in tasks [116]. Developers may find it difficult to give enough attention to security when working on complex applications. This resonates with the factor "not identifying security blind-spots" (I9) as the complexity of the task-in-hand may introduce security blind-spots that developers fail to notice.

Lack of division of labour (E4): Security is considered as an intrinsic requirement by some developers or/and their teams, and as an extrinsic requirement by others [126]. When responsibilities are not clearly defined due to lack of communication, security may be neglected in team environments.

3.5 Intention Deficit

Intention Deficit occurs when developers lack the intention to develop applications securely, for example not prioritising secure coding, or deferring it, or deciding that it is not their task. Intention is a strong influence on behaviour of an individual in a specific way [13]. Sometimes developers may decide to develop their applications without consciously following secure practices for reasons that may or may not be in their control.

3.5.1 Internal Factors – Intention Deficit. Five internal factors that influence developers' intention to code securely were identified.

Requires extra effort (I14): The landscape of software security may change when new vulnerabilities surface. Balebako et al. observe that developers may find it daunting to keep up-to-date with the latest vulnerabilities and the sheer amount of security knowledge scattered over the Internet. Doing so requires extra effort [30]. Assal and Chiasson report that sometimes developers may avoid the extra effort to rewrite existing code and in doing so end up accidentally misusing frameworks and introducing vulnerabilities in code [23]. Some developers do not find any benefit in investing their time and effort in using security tools [178]. The extra effort required to code securely may preclude intention and hence action.

Perceived lack of own security knowledge (I16): Developers' lack of confidence in their security knowledge also impedes their efforts to achieve their security goals [24]. Security knowledge requires continuous learning, as more and more attacks and vulnerabilities surface over time. Developers who are provided means to continue updating their security knowledge are more likely to adopt secure practices [179], whereas developers who perceive their security knowledge to be insufficient are least likely to adopt them [24].

Not perceiving usefulness of secure practices (I15): Developers may have a number of reasons for not considering security practices useful or important. They may not realize the negative consequence of writing insecure code [157], may not value protecting their users [24], or may simply not consider security practices useful or having any tangible benefits [179]. Since security is often not part of coding standards, developers usually do not find them important [23]. Some do not find security to be a "short-to-market" feature [126] and hence do not prioritise it (p. 1293). Developers may also consider security practices unnecessary if they perceive their user bases to be only authenticated users [178, 183]. Since secure coding practices require extra effort and cost, developers and team managers often struggle with evaluating their usefulness against their cost, and the developers using security tools find them expensive to use [30, 178].

Attitude of "someone else's responsibility" (I17): When working in team environments, especially when security experts or teams are present, developers often do not consider security as their responsibility [23]. Some who are not required to code securely do not consider security as their responsibility [24]. Developers who do not use security tools may feel that they can depend on code reviews [183].

Loss of focus on security (I13): Just as competing demands may influence attention (as discussed above in Section 3.4.1), they may also influence intention, causing a similar loss of focus on security. Developers get overwhelmed with many false positive warnings produced by security tools, which discourages them to use security tools [157]. Also, when sufficient time and budget is not available, developers sometimes explicitly overlook security [184].

3.5.2 External Factors – Intention Deficit. The security culture of a team is an essential determinant of security in code [24]. In companies where security practices are not encouraged, and security plans do not exist, developers ignore security in code [23]. However, in organizations where developers often talk about security in informal settings, developers are more likely to adopt secure practices [183]. Companies that are willing to change their software development processes and integrate secure practices into them have a positive influence on developers' secure coding behaviour [127]. Eight factors that lead to intention deficit were identified.

Absence of explicit expectation of secure coding (E5): Developers may avoid secure practices if they are not required to code securely [23, 108, 115], whereas developers are willing to write secure code if they required to do so [179, 183]. In the absence of explicit expectation of secure coding, developers often do not make secure decisions when coding.

Limited resources (E6): Lack of resources such as time, budget, and expertise also often lead to intention deficit. Adopting secure practices incurs additional cost on the project. Development teams may decide to avoid secure practices altogether if they do not have security expertise in their team and are limited in time or budget to meet security challenges [23, 157].

Security tools not reachable (E7): Developers are also deterred from writing securing code if the security tools are not reachable to them, as determined by the tools' *accessibility* and *availability*. In companies where accessing security tools requires formal procedures, developers avoid investigating and using them [183]. Similarly, if required security tools are not available, then developers tend to avoid security practices [24].

Lack of prioritisation of security features by stakeholders (E9): Just as not all developers prioritise security, not all stakeholders prioritise security, which then influences developer behaviour. Customers often do not accept security as a product feature [126]. Team managers and development teams often find it challenging to make security a product feature and adopt secure practices[126].

Lack of security culture in teams/companies (E8) and Lack of social influence (E10): Social influence plays a strong role in developers' decisions to develop securely. Developers who observe their peers using security tools are more likely to adopt them [183]. Developers feel encouraged to adopt security practices if they feel prestige in using them [179]. When developers interact more often with security teams, they feel a greater sense of social responsibility to secure their code [183]. However, without those social and cultural influences, developers may not decide to code securely.

Usability issues with security tools and APIs (E11): Developers readily abandon tools and APIs with which they are not comfortable [128]. Usability issues with security tools are an important deterrent from using security tools. If developers find security tools complex [106]; find it difficult to interpret results [157] or to set up the environment [106]; or struggle with finding right security tools or a combination of them [148], then they may give up the use of security tools altogether.

The next section looks at the different factors that influence developer's security decisions, elicited in this section, through the lens of cognitive and social psychology.

4 ANSWERING RQ1: CAN COGNITIVE AND SOCIAL PSYCHOLOGY HELP TO EXPLAIN THE IMPEDIMENTS TO DEVELOPERS' EFFORTS TO MEET SECURITY GOALS?

4.1 Knowledge Deficit

Section 2.1 discussed that people's perception of their knowledge and skills influences their likelihood of meeting their goals [21]. Developers who feel that they have insufficient security knowledge lack motivation to build secure software [24]. It is also an important determinant of social acceptance among peers. Developers often reach out to other developers who are more knowledgeable if they are working on an unfamiliar task [134] or if they need help in dealing with a confusing API [129]. This type of information seeking behaviour is noted by social psychologists as *informational influence*, where developers seek information from others as a way to resolve uncertainty—what Ciadini calls "social proof" [46] (discussed earlier in Section 2.1.4).

Developers come from a range of programming backgrounds, work in diverse environments, and have different knowledge models [7] that shape their perception. The analysis of knowledge deficit suggests that security goals may be impeded by several internal factors that stem either from developers' lack of sufficient knowledge and skills or from their unhealthy reliance on their own heuristics. Although internal factors are linked implicitly, we look at the subtle differences between them in the way they are acknowledged by the developers and researchers in the reported studies.

Half of the internal factors that lead to knowledge deficit are concerned with a developer's lack of sufficient knowledge and skills. This is a natural consequence of the growing spectrum of developers and technological advancements. Developers are challenged with continuous streams of knowledge with which they must keep pace. New security threats keep emerging, and programming languages and APIs are evolving continuously—e.g., making older versions out-of-date, providing new features, fixing bugs, replacing old methods with new more efficient and secure methods. Developers are expected increasingly to gain a basic understanding of a large number of new technologies to face market competition and meet time pressures [124]. Although developers build their understanding of different technologies, not all the information is stored or processed equally. Cognitive psychology considers several types of memory [159], of which episodic and semantic memory systems are of particular importance in the context of this discussion. Both episodic and semantic memory systems are considered part of the long-term memory system [159]. While semantic memory "stores knowledge about the world in broadest sense" [159], (p. 1) commonly referred as general knowledge, episodic memory "enables a person to remember personally experienced events" ([159], p. 1). Episodic memory lasts longer than semantic memory, as it is associated with emotions and images that make it more resistant to forgetting. This explains why developers who have had negative experiences with security vulnerabilities have stronger intention toward secure coding [24] (discussed later in Section 4.3, Intention Deficit), and why recall of general knowledge from semantic memory may require more cognitive effort during coding than developers are able to expend in the presence of higher-priority competing demands.

Under increasing demands on cognitive resources, individuals often retreat to heuristics [141]. Smith et al. [148] observed that, despite the availability of alternate strategies, developers continued with their old practices. This explains why half of the internal factors of knowledge deficits relate to unhealthy reliance on heuristics. Heuristics allow decision makers "to process information in a less effortful manner than one would expect from an optimal decision rule" ([141], p. 1). Developers' reliance on heuristics may lead to outdated practices [108] and misconceptions [148] often borrowed from working with different frameworks and APIs [23]. Developers may also assume common cases of vulnerabilities, ignoring the corner cases [116], again relying on their existing knowledge of where vulnerabilities may lie. Though various security tools and awareness documents are available to supplement developer's security knowledge, under the time constraints and false perception of their own security knowledge, developers often tend to rely on their existing knowledge and heuristics [67].

Many external resources, though available to help developers write secure code by providing information, communication, and tools, may at times impede the achievement of security goals, for example if they are inadequate, erroneous, misleading, difficult, or demanding to use. The concept of *referent informational influence* extends the analysis of informational influence beyond the interpersonal relations to the role of group membership, which we see in studies of the impact of fora like StackOverflow on individuals' behaviour. Several studies on StackOverflow show how developers security perceptions are shaped by conversations in StackOverflow [99] and how security is effected in their code by copy-pasting code from StackOverflow [63]. Our study on how developers use social fora as a knowledge base also shows that developers are influenced by cognitive biases when choosing which advice to follow, e.g., favouring advice based on how it looks regardless of whether the advice works or is secure [165]. The limitations of these awareness documents are discussed in more detail in Section 5.2.

This psychological analysis of knowledge deficit suggests that different developers have different knowledge bases. Tools and techniques are required that provide security knowledge to developers that map to their knowledge base in a personalized manner and use techniques to embed correct security knowledge into their mental models. Developers should be provided

with a personalised infrastructure of support that can share the responsibility and ease the pressure upon them. In most of the development scenarios, developers often consult different information sources and rely on information about the tools they use provided to them by their team members—which may vary and may be inadequate. The existing limitations of information sources and the increase in expectations on developers call for more focused efforts by researchers to better equip developers in writing secure code while requiring less cognitive effort.

4.2 Attention Deficit

The internal factors that account for attention deficit surface due to developers' limited capacity to handle cognitive load and to familiar routines that may be insecure. Cognitive psychologists understand the mind as an "information processing system" with a limited capacity to process information [21]; this explains why, under cognitive burden, individuals often revert to practices that require less effort [141].

It is important to recognise that software developers differ from each other in their cognitive makeup. For example, it is now widely recognised that working memory is an important source of variation between individuals [49]. Working memory has been described loosely as the workbench of cognition, the system that combines current mental processing with retention and transformation of thoughts and ideas [28]. Working memory capacity, used for simultaneous memory and processing operations [55], is a stable mental ability and is linked to intelligence [93], distraction [104], and inhibitory control [133]. Developers attend to security in code differently, depending on their cognitive makeup. Where software developers need to develop code to reflect multiple constraints, functions, and code interactions, internal mental representations and mental models [81] are going to be strained, and some developers—those with better-chunked working memories—are better positioned to juggle and resolve complex mental dynamics. Of course, software failures are not some ineluctable consequence of smaller working memory capacity *per se*; developers can draw on external props, aids, and strategies to check their work and augment their working memory. However, the fundamental message is that one should not assume that all developers achieve the same end point through the same psychological processes.

The external factors that result in attention deficit relate to competing demands coming from the environment or the task-in-hand. These competing demands direct developers' attention away from security. To understand attention and its role in programming activities, psychologists have refined the concept of "attention investment" [35], the notion that people manage the "investment" of a scarce resource (attention), based on what they value and on perceived risk of failure. Developers' attention to security matters is often compromised in favor of other competing aspects, such as functionality, that provide an apparent better return on investment to developers, especially in the presence of limited resources. A quote from one of the participants in the study of Balebako et al. [30] illustrates this: *"Even self-described privacy advocates and security experts grappled with implementing privacy and security protection with limited time and resources."* Researchers need to provide novel approaches that can help developers in understanding the "return on investment" that security brings to their applications, and hence prioritise it (rather than relegate it as a secondary concern). Researchers can learn from models such as "attention investment" models [35] to capture developers' attention to security in the face of competing calls on their attention.

Developers' shifting attention with respect to security can also be explained in terms of the taxonomy. Other factors that vary from one person to another (e.g., habits and personality traits, security skills) and from one situation to another (e.g., task complexity, security expectations, stakeholder priorities, and lack of division of labour) affect developers' attention. Tanner [154] suggests that objective constraints emerging from the environment may prevent the performance of the desired behaviour—i.e., that not all factors are within developers' control. Security researchers

need to develop novel methods and techniques to monitor such internal and external factors that affect developers' attention to inform how developers' attention can be directed toward security.

4.3 Intention Deficit

Goals are achieved when actions are executed by actors [98]; when developers do not intend to code securely, they do not execute all the actions necessary to meet the security goal.

Although many research studies identify that developers at times do not intend to develop securely, recent studies show that developers *do* engage with other developers in meaningful discourse about security problems [99], showing their interest in secure behaviour. Our analysis also suggests that external factors play a crucial role in influencing developers' intention to code securely; it is not just a matter of a "disinterested attitude." Developers often work under tight constraints. Limited budget [23], tight deadlines [157], and the customers' lack of interest in prioritising security often limit developers' intention to put extra effort and time into coding securely [78]. Lack of security culture may also limit management's willingness to facilitate secure coding [126] and may reduce peer influence on developers to exhibit security behaviour [183]. Nevertheless, managers are able to make changes that support secure development, for example by promoting a development culture that values security, investing in tools, including security in specifications, and budgeting for secure coding [73].

While we outline factors that negatively influence a developer's intention to code securely, we also note contrasting observations in recent research that developers may exhibit secure coding behaviour in the absence of a clear security rationale [164] (i.e., may exhibit secure coding behaviour without specific intention to do so). That research also observed that social considerations affect developers' coding decisions, resulting in developers exhibiting secure coding behaviour for reasons other than security goals [129]. We find evidence that suggests that developers shift from secure choices to insecure choices if they do not consider themselves part of the team anymore [129], which also aligns with what social psychologists suggest about changes in the immediate social context and its effect on goal orientation [162].

While coding is a cognitively engaging activity, it is increasingly becoming a social affair. Developers who use security tools feel more appreciated for their secure behaviour [179], and developers who lack security culture in their organizations [23] do not exhibit secure practices. Developers are likely to adopt security tools under influence of their peers [183], use code-snippets written by others [63], and actively engage with and "tend to security problems" in their conversations with other developers on social platforms ([99], p. 1). Coming from a social psychology perspective, Smith and Louis [150], show that the attitude-behaviour relationship is mediated by group processes. Attitudes and behaviour tend to be more consistent when the norms that guide behaviour come from salient and important reference groups. More importantly, Smith and Louis show that different kinds of group norms can have different consequences for behaviour. They draw an important distinction between descriptive norms (what group members do) and injunctive norms (what group members approve of). These different kinds of norms—derived from seeing behaviour through a more dynamic, inter-group lens—are important concepts for helping to explain why the intention to develop securely might not be reflected in practice.

The key takeaway message from this discussion is that impediments to security goals come from a spectrum of deficits that stem from the cognitive and social makeup of developers. We have presented evidence of a multi-dimensional set of psychological limits that can explain security vulnerabilities reported in the research literature. The next section investigates security interventions available to developers during coding. In addressing RQ2 in Section 6, we postulate that security interventions should respond to a spectrum of deficits and help the developer overcome these psychological limitations.

5 WHY ARE SECURITY INTERVENTIONS NOT MORE EFFECTIVE?

This section presents an overarching analysis of the "state-of-the-art" in security interventions. First it discusses the role of security interventions in addressing vulnerabilities in code and sketches out a landscape of different types of security interventions for developers. Then it considers how these security interventions address different factors influencing security goals. The analysis shows that current security interventions fall short of taking a holistic view of socio-technical systems; i.e., they fall short of considering "the human in the loop,"[3] which can be a critical vulnerability in software applications.

We postulate that security interventions "at the desk" of developers need to be based on a broad socio-technical perspective, one that considers the interdependence between different factors in the design of a system that include humans, human interactions, context, and the system itself [31]. Socio-technical systems require a holistic view of security requirements with an integrated view of different layers of the system, such as social, application, and infrastructure layers [98]. Pieczul et al. [124] also suggest the need for *new thinking* to support a continuum of developers with varying skills and expertise. We suggest "adaptive security interventions" that take the socio-technical context into account, and therefore respond to the different security needs of the developer.

The next section uses the impediments taxonomy—and the compilation of factors impeding the efforts to meet security goals—to shed light on security interventions.

5.1 Overview of Security Interventions

Security interventions play a vital role in improving code security. For example, Gorski et al. [69] showed that placing secure programming hints closer to the programmer as API-integrated advice can improve code security significantly. Acar et al. [10] also showed that the use of different information sources during coding affected code security. In their study with developers, they observed that the developers who used Stack Overflow produced significantly less secure code than those who used official documentation or books to get help during coding.

Security interventions were defined earlier as those events that occur in between the developer and the coding activity to promote the developer's secure coding behaviour. In this context, security interventions are an active event, rather than a passive resource. So, for example, a static analysis tool or an OWASP list of common vulnerabilities [152] is a resource available for developers to use. However, when a developer uses one of them actively while coding to avoid vulnerabilities, the same resources are referred to as *security interventions*.

Developers use security interventions at different phases of development. For example, a developer may use the OWASP top 10 list *during* coding to know how to mitigate specific vulnerabilities and implement the right mitigation strategy. A developer who is interested in detecting vulnerabilities in *existing code* might use a static analysis tool to review the code [76], e.g., **static application security testing (SAST)** [27] tools integrated into the development environment as IDE plugins. These examples also indicate that the broad range of "interventions" have different "presentation styles": they may take different forms (e.g., information sources for developers that developers can consult during coding, tools to analyse code that are triggered through IDE), may have a different focus (e.g., improving developer awareness on how to improve their code, identifying vulnerabilities in code), and may be invoked and controlled in different ways (e.g., manually by the developer, or automatically within an IDE).

This section presents the results of our high-level thematic analysis of current security interventions to understand the patterns in their "presentation styles" and how those patterns may

[3]We've borrowed this term from Falcone et al. [62], who use it in a different context.

influence the usage of the interventions. The review targeted authoritative resources that would reflect the current "state-of-the-art" of security interventions:

- We reviewed key security resources available in popular and well-established online resources. These include Cybersecurity resources available online by MITRE [50] and list of different security tools available online by OWASP [2, 3, 5]. We visited these resources, as they are considered authoritative sources and are visited frequently by developers.
- We also reviewed other authoritative online resources that are visited frequently by developers for security information. The work of Acar et al. [12] lists and studies 19 authoritative websites that developers trust and use frequently for security advice. (While their work studies such resources in detail for the kind of advice they present and identifies open issues in these sources of advice, our aim was to look at these sources of security information from the viewpoint of their presentation styles.)
- Our earlier research on developer's security behaviour (mentioned in Section 3.1) included search strings such as "developer security tools." We reviewed that literature for any sources not covered by the previous lists.

This analysis of existing security interventions identified three broad "presentation styles": *awareness interventions* (that just provide security information to the developer), *automated interventions* (that developers can run on code to detect and sometimes mitigate vulnerabilities), and *interactive interventions*, (that "interact" with the developer to provide task-specific advice and assistance with processes such as refactoring and auto-completion).

All the security tools and resources presented in the MITRE cybersecurity resources [50], and in the OWASP list of security tools [2, 3, 5] can be categorized either as awareness interventions or automated interventions. The 19 information resources studied by Acar et al. [12] can be categorised as awareness interventions. While many of the research papers presented tools that can be categorised as awareness or automated interventions, we also came across literature that offers research prototypes of an interactive nature. These interactive security interventions incorporate both awareness and automated interventions to provide on-the-spot advice to developers when coding.

Figure 1 shows the existing landscape of security interventions discussed here. Interactive interventions are a more recent development that builds on the other two presentations styles. Hence, Figure 1 shows awareness and automated interventions as a "first slice" and interactive interventions as a "second slice" emerging from them. Each of these presentation styles is discussed briefly below and examples are given.

5.1.1 Awareness Interventions. Awareness interventions are security information resources, such as check-lists, vulnerability databases and FAQs, used by the developer during coding to improve security in the code. These security information resources are disseminated to provide secure coding information to developers. These interventions make developers aware of code security, but leave it to the developers to take action. For example, the **Common Vulnerability and Exposures (CVE)** [54] system provides a list of publicly known cybersecurity vulnerabilities with reference IDs. The CVE list makes it easy to share knowledge about vulnerabilities across networks. While the CVE gives a reference list of vulnerabilities, the Seven Pernicious Kingdoms taxonomy [6] provides a categorization of vulnerabilities to provide a high-level understanding of them. The OWASP Top 10 [119] is another awareness document often used by developers. It is updated regularly by the **Open Web Application Security Project (OWASP)**, an international, non-profit organization that is working on improving the security of software through various projects [118]. It provides information on the most critical security concerns, along with

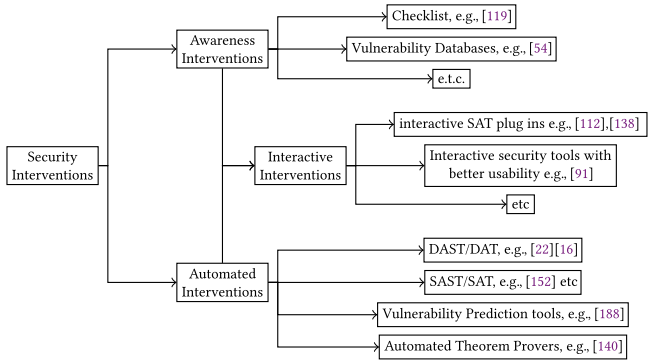

Fig. 1. Landscape of security interventions for software applications.

suggestions on how to address them. Awareness resources are often used by tool-smiths to design and evaluate the effectiveness of tools. For example, Nguyen et al. [112] and Sampaio et al. [138] use a subset of vulnerabilities from the OWASP list of common vulnerabilities to evaluate the effectiveness of their security tool prototypes.

5.1.2 Automated Interventions. Automated interventions provide fully automated methods and are typically run on code or software applications to detect and sometimes mitigate vulnerabilities in them. These include, but are not limited to, **Static Application Security Testing (SAST)** [152] (also known as **Static Application Testing (SAT)**) and **Dynamic Application Security Testing (DAST)** (also known as DAT) [44] tools, automated theorem provers[44], and vulnerability prediction tools [188].

SAST tools analyse source code and detect potential vulnerabilities without executing it [152]. The area of static analysis for security has matured over time, and static analysis is used widely in industry. It is also an active area of research that focuses on improving the efficiency and accuracy of the results [96].

DAST tools scan the applications from the outside, examining their behaviour and looking for security vulnerabilities [44]. DAST does not require source code to analyse the software for security and produces false-positive rates close to zero [138]. Examples include penetration testing [16, 22], fuzz testing [68], and source code fault injection [77].

Automated theorem provers (ATP) are programs that use formal reasoning to evaluate if a given formula is universally valid or not [140]. Many works use ATP to analyse code security; for example, Jurjens [83] applies ATP to an industrial-strength biometric authentication protocol for determining security goals.

Vulnerability prediction tools [188] are used in large projects to predict vulnerable parts of software before releasing it to focus efforts on those parts that need more attention. These tools use software metrics [188] or machine learning, and mining techniques [110] to predict vulnerabilities in different parts of the software.

All these different types of interventions deploy automation to analyse and/or fix vulnerabilities in code. The automated interventions generally work as a black-box. The role of developers is limited to initiating the tools and receiving the output generated by the tool. It may be that developers use these tools iteratively in their practice, in a way that may seem "interactive" to them (e.g., running the tool to find potential vulnerabilities (perhaps ranked by severity) and then applying patches if the developer agrees with the warning). However, the tool itself is not "interactive."

5.1.3 Interactive Interventions. Interactive interventions, the "second slice" of security interventions, include semi-automated systems for providing task-specific advice and support for processes such as refactoring and auto-completion. Such tools are invoked by the developer or are run in the environment. They provide analytics to address specific vulnerabilities in code and require developers' involvement in making a secure decision, e.g., providing notifications to developers on vulnerabilities in their code while they are working on it [138], highlighting vulnerable parts of code and offering secure defaults [112], providing integrated security advice to developers on how to deal with security warnings [69], and aiding code refactoring by providing secure solutions to developer while coding [185].

Developer responses may then prompt re-analysis and further interaction. Interactive interventions include a degree of automation, but they are typically not autonomous: The developer must engage and agree before code is changed.

Fixdroid by Nguyen et al. [112], ASIDE by Xie et al. [184], Eclipse plugin by Sampaio et al. [138] for continuous detection of vulnerabilities, and implementation of a security patch for PyCrypto API by Gorski et al. [69] are all examples of such interactive security interventions. We only found a few examples of interactive security interventions in the past decade, but with more efforts in the past few years suggesting that researchers realize the importance of involving developers in security decisions in the secure coding process.

5.2 Security Interventions and Goal Impediment Categories

This section presents an analysis of how different styles of security interventions may—or may not—help to avoid or address different security goal impediments. The discussion draws on both the conceptual underpinning of the goal impediment categories and the specific factors identified in Section 3. It examines both why interventions may help and why they may not be effective.

5.2.1 Awareness Interventions. Awareness interventions, as their name suggests, are typically used by the developers to foster their security knowledge during coding.

Awareness interventions address *knowledge deficit* among developers. However, Tables 2 and 3 show that knowledge deficit occurs due to number of factors, but awareness interventions largely address *inadequate information sources (E1)*. While factors like *misconceptions (I1)* and *outdated knowledge of developers (I2)* can be addressed effectively by using these awareness interventions, studies show that these factors emerge when developers rely on their existing incorrect knowledge base instead of using reliable information sources.

Awareness interventions can also effectively propagate knowledge about security tools to mitigate the internal factor *lack of awareness of security tools and new vulnerabilities (I8)*. Many authoritative online resources provide a list of security tools for different programming languages. However, these community resources often do not endorse any particular tool, and the responsibility of selecting the "right" tool often falls on the shoulders of developer (or development team). Developers also often feel overwhelmed by the influx of security information over the Internet, which can impede their efforts to achieve security goals.

In team environments, different types of information sources have been used effectively in improving security culture in teams. Weir et al. [171] use intervention techniques such as security games, on-the-job training, and threat assessment in a lightweight approach that influences developers' security thinking positively. Although the use of an intervention package with awareness techniques can improve the security culture in teams, without the presence of an external authority pressing the need for security, awareness documents may not effectively address knowledge deficit among developers.

Awareness interventions may not be an effective tool to address *attention deficit* and *intention deficit*. In fact, the additional cognitive effort that some awareness interventions require (for example, because they provide a great deal of information [148]) may *contribute* to information overload leading to *attention* or *intention deficit*. Developers may find it hard to remember all the information available in awareness documents or to track which of the vulnerabilities they have already addressed [148]. Lack of contextualization of security awareness documents to a developer's skills and task-in-hand may impede the effectiveness of these interventions [148]. Also, developers often struggle with finding required information about security tools in awareness documents: Some developers do not find the right security tool for their task-in-hand [148], and some find the great amount of information in these resources too overwhelming [29].

5.2.2 Automated Interventions. Automated interventions are popular in industry and academia, and security tools are often recommended by security experts to developers to improve security in their code (e.g., Reference [117]). The main strength of automated interventions is their efficiency in detecting vulnerabilities in code without much human effort. In addition, these tools have matured over time, due to the large body of work underpinning them and continuous efforts by the security community to improve their effectiveness.

At the heart of these interventions is the idea of reducing the amount of effort that developers need to invest by automating manual tasks such as code reviews. While automated interventions ease the task of code reviews, other human factors that require extra effort surface. Developers find it difficult to stay up-to-date with the volume of new vulnerabilities and security tools, and to know which security tools address which vulnerability. Doing so is a challenge that requires extra effort.

Security tools can help developers in finding security blind-spots in code. Recent advances in algorithms, such as analysing context-sensitive data-flow [138], can be effective in identifying vulnerabilities that might be difficult to detect otherwise. Oliveria et al. [116] also suggest the use of on-the-spot programming as an effective mechanism for addressing security blind-spots; however, empirical evidence is needed.

In spite of their wide popularity, automated interventions may also contribute to a number of factors that lead to *knowledge deficit* and *intention deficit*. Developers may introduce vulnerabilities in code if they do not use security tools correctly and have less development experience [78]. Developers who find security tools hard to use do not adopt them [178].

Usability issues with security tools also impede developers' intention to code securely. Recent advances in usable security aim to address this problem [10]. For example, Assal et al. [25] propose use of a visual analysis environment to help developers better identify vulnerabilities in code.

5.2.3 Interactive Interventions. By involving developers in security decisions, interactive interventions take a step further (than awareness or automated interventions) to address impediments to developers' efforts to code securely. Interactive interventions can address a developer's misconceptions by providing "quick fixes" to security issues in code [69, 112] and generating alerts if misuse occurs [91]. They provide recommendations for secure code snippets in developers' IDEs [69]. Integrated security advice in APIs informs a developer of the consequences of particular misuse [69], which can alert the developer to the negative consequences of insecure programming practice; i.e., the advice is contextualised to the task-in-hand.

On-the-spot, contextualised warnings highlighting the vulnerabilities in code [69, 112, 184] compensate for the absence of explicit expectations of secure coding (E5). If the impediment is attention deficit, then this on-the-spot warning can encourage security behaviour at the cost of interruption flow.

In team environments, interactive interventions can share knowledge by generating logs that provide information on how different team members address security during development [184].

This can help in sharing the cognitive load of remembering which vulnerabilities have been addressed and by whom.

The effectiveness of interactive interventions has been reported by a number of researchers. Gorski et al. [69] reported 73% improvement in code security among participants who used an API that provided security advice. Ngyuen et al. [112] also reported a significant difference in security solutions for developers using FixDroid, which provided on-the-spot security alerts. Researchers should also investigate the role of interactive interventions in addressing other factors that influence security decisions.

The field of interactive interventions is relatively new, with only a few studies published in the past decade. Further research in this field can address various other factors that make it challenging to meet the security goal. Researchers and tool-smiths can identify better approaches and techniques by working with cognitive and social psychologists who have long studied impediments to behaviour goals [105].

Interactive interventions provide code-context information to encourage secure coding practices combined with the lessons from human computer interaction and human factors in computing. However, developers work in a rich cognitive and social environment, which may differ from one developer to another. Section 6 looks at interactive security interventions through the lens of psychological theories and makes a case for *adaptive* security interventions that are cognisant of the developer's complex socio-technical environment.

5.3 Discussion

This section has discussed how different types of security interventions map to different impediments to efforts to achieve security goals. With the efforts of the security community and advancements in security technology, security interventions have also matured. Authoritative awareness interventions have developed, and several centralized repositories exist that provide rich information to developers at different stages of development—if a developer knows where to look. Automated security tools have matured over time and have automated several review- and testing-intensive tasks in a more efficient and effective manner—if a developer knows which tool to use and for which task. Interactive interventions have recently come to the fore and aim to increase developers' involvement in making secure decisions while coding—if the developer does not find them intrusive.

While these interventions address critical issues that needed attention, several open issues remain. Most "state-of-the-art" interventions are each able to address effectively one or a few of the factors contributing to impediments to security goals. However, issues such as individual differences, stakeholder priorities, information overload, interrupting notifications, and competing demands present significant challenges to addressing a wider range of factors and the impediments to which they contribute.

6 TOWARD ADAPTIVE SECURITY INTERVENTIONS

The previous sections discussed literature from cognitive and social psychology about elements that influence human behaviour with respect to meeting—or failing to meet—goals, investigated developer-centered security literature focusing on studies of developers' security behaviour, and sketched the current landscape of security interventions that are available to the developer *during* coding. The analysis identified: (1) three categories of security goal impediments, (2) a list of factors that lead to security goal impediments, and (3) three types of "presentation styles" of security interventions. The last section discussed how different types of security interventions address (or lead to) different types of security goal impediments.

This section makes a case for adaptive security interventions. Section 6.1 explicitly discusses our answer to RQ2 and argues that the security interventions produced for developers should be informed by theories from psychology in a socio-technical context. We suggest adaptive security interventions to address the gap between developer needs and existing solutions. Section 6.2 outlines an architecture that responds to this proposition, the proof-of-concept implementation of it, and a number of open research questions and challenges.

Early works on dealing with information security in computer systems emphasised the role of behavioural sciences in designing and developing secure systems [137]. Despite such emphasis, advances in software security and research surrounding it have revolved mainly around the technical aspects of security, involving the developer as one of the components whose behaviour can be specified and controlled [139].

While studying the landscape of security interventions available to the developer "at the desk," recent years have seen more focused efforts to bridge technical security tools with human decision-making through interactive tools. Prior to that, efforts to enhance security in software were less oriented to the developers' reasoning and behaviour: Awareness documents provided human-readable technical knowledge to avoid or mitigate vulnerabilities in code, while the security tools such as SAST and DAST had little consideration of the usability of tools [10].

The **systematic literature review (SLR)** on human aspects of software engineering [94] establishes behavioural software engineering as a recognized area in software engineering. In a similar line of research, SLRs on motivation [32] and personalities [53] also shed light on how software engineers work. A recent SLR by Tahaei and Vaniea [153] on developer-centered security also reports on security studies with software developers, providing an overview of the methodologies being used in the DCS literature. However, despite recent advances in studying human aspects of secure software development, the overarching analysis on understanding software developers with particular focus on security is scarce—which is what we try to address with this work.

6.1 Answering RQ2: How Can We Design Interventions to Help Developers Reduce the Obstacles in Operationalizing Security Goals?

The answer to RQ1 presented in Section 4 succinctly summarises the psychological limits of developers in addressing security vulnerabilities and the need to address a spectrum of deficits with security interventions. This section first outlines the lack of psychological underpinning in current research on security interventions and its drawbacks. It then suggests that security interventions should be adaptive and should take the socio-technical context into account. It lays out how adaptive security interventions can be designed and developed, informed by psychological theories, to address a spectrum of deficits.

Recent efforts to deliver interactive interventions (e.g., References [69, 112]) to support developers in writing secure code are built on lessons learned from research on usable security tools. This research suggests capturing developers' attention toward security issues in code through the user interface (e.g., References [9, 10, 109]). Developers process varying amounts of information during coding. Advances in user interface design strategies have focused on aggregating relevant information for the task-in-hand and displaying it at the user interface using indicators that grab the developer's attention. These indicators also serve as external memory and provide easy access to required information. The advantage of this approach is that it improves developers' productivity and efficiency in producing secure code—as evaluated through usability studies and controlled experiments in laboratory settings. Despite this positive effect, psychologists suggest that the main drawback of "display-based strategies" is that they lack involvement of cognitive processes to commit information to memory [19], and they will negatively affect activation of memory traces in

the future [169]. Security researchers have not yet studied how developers' cognition is affected by ease-of-access to information.

Programming requires a high level of concentration [18], which is vital to developers' productivity [177]. The provision of various prompts to security practices and of continual tips to handle security misuses may interrupt the code-flow, especially for developers whose "attention-investment" model [35] does not align with security. The attention-investment model (discussed earlier in Section 4.2) postulates that people manage the "investment" of a scarce resource (attention) based on what they value and on the perceived risk of failure [35]. Cognitive psychology literature suggests that individuals are more prone to making unintentional mistakes and producing inaccuracies due to temporary lapses of attention caused by interruptions [17]. Research is needed to study how continual interruption and the ease-of-access to relevant security information affects developers' (security) learning—with negative consequences on developer's (security) behaviour in the absence of these external memory resources [19]. Research on interactive security interventions needs to go beyond the "on the spot" display of relevant information to capture developers' attention, and should be informed by psychology theories on improving human knowledge and behaviour to complement these efforts.

Further, we found little research into the long-term effect on developers' cognition of "display-based" security interventions. Some security intervention studies suggest improvement in developer productivity for the task-in-hand [112], while some others discuss a rise in the security awareness of developers in teams [127].

Poller et al. observed security behaviour of developers in an organizational setting after security audits and workshops. They noticed that, although these interventions were effective in raising developers' security awareness and behaviour for a short period of time, developers did not continue with their security practices over a longer time (one year). Weir et al. [170] showed that a series of lightweight interventions over a three-month period was effective in changing security culture in teams, even without the involvement of a security expert. Their evaluation was based on a series of interviews conducted before and shortly after the interventions, and then 12 months after the intervention. These examples study security culture within organizations. However, research is lacking in the study of how the security interventions that developers use *during coding* develop long-term change in their security behaviour and how different types of developers' needs are fulfilled by a given security intervention.

As discussed earlier, theories from social psychology and evidence from developer-centered security suggest that we need to leverage social influences on developers to encourage security behaviour during coding. Different factors influence how developers approach security in code and can cause knowledge deficit, attention deficit, or intention deficit. Further, a developer can experience any of these deficits at different points in time, depending on changes in the developer's socio-technical context. Our analysis suggests that different deficits require different forms of intervention to help developers achieve security goals. We therefore suggest that such (interactive) security interventions need to be *adaptive*, recognizing the different individual and development contexts and the security-relevant factors that arise within them and tailoring the interventions in response to changing circumstances.

Adaptation is seen increasingly as an effective approach to addressing uncertainty and change in the environments in which software systems are created and used [56]. We suggest adaptive security interventions that intercede between developers and their code during software development to address the needs implied by knowledge deficit, attention deficit, and intention deficit. Information concerning the developer's coding behaviour is collected and analysed to identify the developer's approach to security, such as how and when the developer attends to security in code, the types of vulnerabilities the developer addresses, and types of information sources

the developer uses. For example, to address knowledge deficit, the developer can be notified of the security vulnerability and the developer's response observed. If the response is absent or incomplete, then suggestions may be offered (in terms of possible actions or relevant information sources). Attention deficit can also be addressed through notifications, but frequent interruption of the developer's code flow in the IDE (which might place competing demands on attention) can be reduced by using diverse communication channels, such as Slack, that developers use often beside the IDE. To address intention deficit, the vulnerabilities in code can be made explicit before code commits to highlight security issues to developers and clients. Developers are able to make conscious decisions to attend to vulnerabilities in code or leave them if they are not critical. In the case of critical vulnerabilities that require extra resources, developers are able to make decisions in consultation with the clients instead of ignoring security in project discussions. This helps dealing with intention deficit where developers compromise security goals due to limited resources and offers them a platform to negotiate security with clients.

Such adaptive security interventions are to be designed with an aim to trigger long-term behaviour change in developers and avoid the negative effects of frequent interruptions (that may contribute to attention deficit). For example, memory-based-strategies [169] may be incorporated in the design of security interventions. Memory-based strategies suggest increasing information access cost to improve the user's information recall [169]; hence, by making some information less-readily available at the interface, (e.g., by adding a delay of a few seconds in displaying relevant information) the user is encouraged to commit relevant information to memory. Similarly, interventions can can be informed by psychological theories of human decision-making, in terms of which information to prioritise (e.g., References [19, 20]).

6.2 Operationalizing Adaptive Security Interventions

Our operationalization of adaptation in the context of security interventions is based on the well-known architectural pattern, the MAPE-K loop, also called the *adaptation loop*, used to support adaptation in software systems [135]. The adaptation loop iterates through the activities of **monitoring (M), analysis (A), planning (P), and execution (E) activities over a knowledge base (K)** to vary the response of a system following changes in its operational environment.

Conceptually, information about developers' behaviour is *monitored* and then *analysed* to identify impediments to achieving security goals, and a *plan* is developed to determine what interventions to *execute* based on *knowledge* of appropriate strategies to address such impediments. The aim of adaptation in our work is to diversify the channels by which security interventions are delivered (not just via IDEs, but also via other communication and collaboration tools), by varying the timing of intervention (not just instantly inside IDEs, but also delayed interventions), and by escalating the interventions when some are not effective (not just directly to the developer, but to the wider social context such as the development team or even publicly). Furthermore, adaptive intervention is non-transactional, as it is a learning process that accumulates knowledge after each iteration and thus can improve over time. The proof-of-concept tool we have built is available online.[4] The tool is a useful lens through which to explore adaptive security interventions further.

To operationalize adaptive security interventions, we first make the development context concrete (Figure 2). This involves making a few operational choices about programming language (Python[5]), the development environment (Visual Studio Code[6]), security tools for Python

[4] https://github.com/open-university-rse/johnny-django-docker.
[5] https://www.python.org/.
[6] https://code.visualstudio.com/.

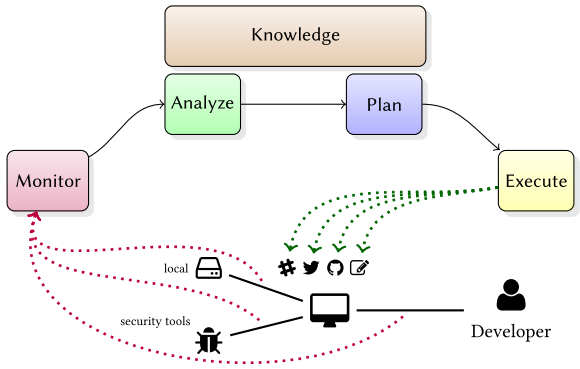

Fig. 2. Developer and work context.

(Bandit[7]), local repository management (git[8]), communication tools (Slack[9] and Twitter[10]), and collaboration tool (github[11]). These choices represent a typical development context of a developer according to a recent industry survey [1].

The **Monitor** component intercepts (red dotted lines in Figure 2) and logs (a) the program texts the developer has written in Visual Studio Code when they are saved on the local disk, (b) all the security warnings provided by the security tools that are shown to the developer, and (c) online searches the developer performs. These events are stored chronologically in a database. Using prior knowledge about patterns of deficits and how they manifest in developer behaviour, the **Analyse** component identifies the impediments to security goals. This is done using pattern matching. The **Plan** component identifies actions that are most likely to produce a positive response from the developer, based on prior knowledge of developer's behaviour. The **Analyse** and **Plan** components are partially implemented to demonstrate the example below. The **Execute** component performs the action identified by the **Plan** component. A history of how the developer has responded to interventions, known deficits and associated behaviours, and available actions are part of the **Knowledge** component, which is also partially implemented.

Consider a scenario in which a developer enters program text that contains a security vulnerability identified by Bandit. Visual Studio Code would highlight the vulnerable part of the code. This is a form of security intervention that is static: The editor will continue to highlight the vulnerability until it is fixed or the relevant code is removed. In an adaptive intervention, factors that influence developer behaviour can be dynamically leveraged. For example, in this scenario, the adaptive system includes a rule that, if the developer did not react to a Bandit warning more than three times (suggesting that the same vulnerability is present in the code after the code has been modified and saved three times), the behaviour is recognized by the system as an instance of attention deficit. This determination is based on the prior finding that developers lose focus on security and only address functional requirements if they are limited in time and resources [30]. As more information about the developer behaviour is gathered over time, better predictions can be made by the system of the developer's salient impediment. The plan generated in this case increases the visibility of the vulnerability within a team by sending a friendly message to a group

[7]https://pypi.org/project/bandit/.
[8]https://git-scm.com//.
[9]https://slack.com/.
[10]https://twitter.com/.
[11]https://github.com/.

channel on Slack to which the developer is subscribed. If the developer is working alone, then the Slack channel serves as an additional prompt to gain the developer's attention (green dotted lines in Figure 2). If the vulnerability is still not addressed, then the issue can be escalated further by creating a pull request on github.com when the developer commits the code. The pull request makes an explicit demand on the developer to actively consider security. If the developer or/and the other team members make a conscious decision to leave the vulnerability in the code, then the decision can be discreetly noted with the merge. In participating in the pull request review process, the adaptive system increases opportunities for a developer and other stakeholders of the project to prioritise and address the security issue. Both examples given here bring social and team dynamics into play, so responsibility for addressing the vulnerability becomes collective, shared among individual developers, groups, and other stakeholders of the projects.

The implementation has raised a number of open research questions and challenges. For **Monitoring**, research is needed *to scope which variables in a developer's environment are worth monitoring, and how to monitor them to understand a developer's secure coding and related behaviours.* A developer's interactions with the interface of a coding environment can help, and research on adaptive user interfaces [15] has suggested ways in which interfaces can serve as a means of flexible engagement with users (in this case developers) and their environment. For example, adaptive interfaces have been studied to address accessibility needs [66], cultural needs [132], mental workload [42], and cognitive processes [151]. We can learn from such work to understand the diverse needs of users from the data they generate. Indeed, developers can generate considerable data while coding [41], which can be used to make informed decisions or interventions. For example, Bruch et al. [41] advise collection of usage data from developers' IDEs to improve recommendation systems, and Matsumoto et al. [102] suggest the use of developer metrics to predict faults in software.

For **Analysis** and **Planning**, research is need *to identify and evaluate analysis and planning mechanisms that can provide an effective strategy to address impediments to security goals.* Different types of adaptive mechanisms have been used to analyse data collected from the operating environment and plan adaptive responses to changes in that environment. In addition to methods in statistical machine learning, argumentation approaches have been used in requirements engineering for deriving adaptive security controls [160], and self-adaptive reporting algorithms have been used to analyse and plan self-adaptive forensic activities. This and related work on developing adaptive systems can provide a solution space for deploying analysis and planning for effective adaptive interventions.

For **Execution**, research is need *to identify the full range of actions that are effective in addressing impediments to security goals in different contexts.* We have so far identified a few actions that can achieve certain goals, such as: visibility (private vs. group messaging), timing (instant vs. delayed messaging), and affect (positive vs. negative messaging). The delivery of interventions is very much dependent on the socio-technical context [31] in which it takes place. It may not always take the form of an explicit security "fix," but may instead be a more implicit and nuanced intervention that provides background support to the developer, or even defers interceding until some later time when the monitoring, analysis, and planning steps determine that intervention is a priority (e.g., when a security vulnerability has been neglected repeatedly).

This proof-of-concept tool based on the MAPE-K loop affords an opportunity to conceive security interventions as contextually sensitive, dynamic, and more engaging forms of support to software developers. Developers are often engaged in challenging tradeoffs between the many demands on their time and cognition and the plethora of options for engaging developers exhibiting insecure coding behaviours and practices. It remains an open challenge to fully implement

and evaluate such automated tools that extend and enrich developers' coding environments. Our future work will investigate this challenge.

7 CONCLUSION

Writing secure code requires not only technical knowledge of the programming language and the domain, but also a security mindset to think critically about how a system can be exploited to cause harm to its assets [176]. Developers often work under significant cognitive load, with multiple issues demanding their attention—including (but not limited to) functional requirements, performance and usability issues, deadlines, and lack of resources [185]. In addition, the social context in which developers' work encourages (or constrains) their security behaviour [183]. The combined influence of developers' cognitive load and social-technical context often leads to software bugs that can cause security breaches [185]. It is but natural then that even the security experts often grapple to keep their focus on implementing security under the cognitive burden [29].

This work has striven to provide a psychological account of the different factors that prevent developers from producing secure code. The analyses presented exposed the importance of the rich context in which software is developed and how individual and contextual differences can either support or impede the achievement of security goals. Although this review has focused more on the individual, we have noted the value of addressing more fully a richer social perspective as well. Software development can benefit from insights provided by social psychological research, arguing, for example, for a more dynamic model of psychological processes that underlie behaviour [158]. This review has highlighted some example instances of social influence on developer behaviour with respect to secure coding. The social identity approach [40, 74] argues that the way in which we think about ourselves (our identity) shapes the way we act —and is sensitive to contextual variation; further empirical work is needed to capture that interplay more fully and to adapt security interventions accordingly.

The paper has made a case for adaptive interventions to promote and improve secure coding that can respond to a developer's particular context, both in the broad socio-technical setting and in response to the needs in-the-moment for the task-in-hand. In making this case, the article presented the following:

- Informed by the psychology literature, it considered the interplay of key cognitive elements (knowledge, attention, intention) within the developer's rich socio-technical context and how these elements characterise key impediments to meeting security goals (knowledge deficit, attention deficit, intention deficit).
- It reviewed the software engineering literature to identify what other researchers have identified as factors impeding secure coding.
- It reviewed existing security interventions, mapping that landscape in terms of the "presentation style" of the interventions (awareness, automated, interactive), and it discussed the broad styles in terms of the impediments framework.

This provided a perspective on why existing interventions can be effective in addressing some impediments—but not all. It thus made the argument for adaptive interventions that are contextually aware. The MAPE-K adaptive systems architectural pattern was used to operationalize adaptive security interventions and present a proof-of-concept tool to unpick questions and challenges associated with generating such adaptive interventions.

APPENDICES

A APPENDIX: INCLUDED STUDIES

Table 5 lists the 29 primary studies included in the review.

Table 5. Selected Papers

Authors	Title	Year	Reference
Weir et al.	"Interventions for software security: Creating a lightweight program of assurance techniques for developers"	2019	[170]
Naiakshina et al.	"If you want, I can store the encrypted password": A password-storage field study with freelance developers	2019	[108]
Assal and Chiasson	"Think secure from the beginning": A survey with software developers	2019	[24]
Chen et al.	How reliable is the crowdsourced knowledge of security implementation?	2019	[45]
Kula et al.	Do developers update their library dependencies? An empirical study on the impact of security advisories on library migration	2018	[92]
Thomas et al.	Security during application development: An application security expert perspective	2018	[157]
Assal and Chiasson	Security in the software development lifecycle	2018	[23]
Smith et al.	How developers diagnose potential security vulnerabilities with a static analysis tool	2018	[148]
Oliveira et al.	API blindspots: Why experienced developers write vulnerable code	2018	[116]
Naiakshina et al.	Why do developers get password storage wrong?: A qualitative usability study	2017	[109]
Acar et al.	Developers need support, too: A survey of security advice for software developers	2017	[12]
Acar et al.	Security developer studies with github users: Exploring a convenience sample	2017	[11]
Fischer et al.	Stack overflow considered harmful? the impact of copy&paste on Android application security	2017	[63]
Lo Iacono and Gorski	I do and I understand. Not yet true for security APIs. So sad	2017	[78]
Weir et al.	I'd like to have an argument, please: Using dialectic for effective app security	2017	[173]
Nadi et al.	Jumping through hoops: Why do Java developers struggle with cryptography APIs?	2016	[106]
Acar et al.	You get where you're looking for: The impact of information sources on code security	2016	[10]
Yang et al.	What security questions do developers ask? A large-scale study of stack overflow posts	2016	[187]
Poller et al.	First-time security audits as a turning point?: Challenges for security practices in an industry software development team	2016	[126]
Witschey et al.	Quantifying developers' adoption of security tools	2015	[179]
Witschey et al.	Technical and personal factors Influencing developers' adoption of security tools	2014	[178]
Loser and Degeling	Security and privacy as hygiene factors of developer behavior in small and agile teams	2014	[101]
Balebako et al.	The privacy and security behaviors of smartphone app developers	2014	[30]
Oliveira et al.	It's the psychology, stupid: How heuristics explain software vulnerabilities and how priming can illuminate developers' blind spots	2014	[115]
Xiao et al.	Social influences on secure development tool adoption: Why security tools spread	2014	[183]
Egele et al.	An empirical study of cryptographic misuse in Android applications	2013	[60]
Xie et al.	Why do programmers make security errors?	2011	[185]
Karppinen et al.	Why developers insert security vulnerabilities into their code	2009	[85]
Dejan et al.	Static code analysis to detect software security vulnerabilities—does experience matter?	2009	[27]

B APPENDIX: DETAILED CATALOGUE OF FACTORS INFLUENCING DEVELOPER'S SECURITY BEHAVIOUR

Tables 6–8 show the detailed catalogue of factors that influence security behaviour of the developers.

Table 6. Factors Associated with Knowledge Deficit

Knowledge Deficit	
Internal Factors	**Instances**
Misconceptions (I1)	*"We found a number of freelancers were reducing password storage security to a visual representation and thus using Base64 as their preferred method to ensure security. Additionally, encryption and hashing were used as synonyms, which was often reflected by the freelancers' programming code." [108]*
	"some developers' mental model of security revolves mainly around security functions, such as using the proper client-server communication protocol. However, the developer did not discuss vulnerabilities due to implementation mistakes that are not necessarily preventable by security requirements." [23]
Use of outdated information (I2)	*"Even participants who attempted to store passwords securely often did so insecurely because the methods they learnt are now outdated."* [109]
	"A number of freelancers used outdated methods to store user passwords securely." [108]
	"a common strategy for participants was to rely on their existing knowledge of sensitive operations and data sources in the application. ... such reliance may be failure-prone whenever the code has been changed without their knowledge." [148]
False assumptions/ inferences (I3)	"*during some tasks, participants incorrectly assumed input validation had been handled securely.*" [148]
	"For instance, any class that started with Test participants assumed was as JUnit test case, and thus was not user-facing, and therefore not a potential source of tainted data...this strategy fails in situations where the word "Test" is overloaded; this happens in iTrust where "Test" can also refer to a medical laboratory test." [148]
	P-T7 said, "I think they kind of assume that if you're a developer, you're not necessarily responsible for the security of the system, and you [do] not necessarily have to have the knowledge to deal with it." [23]
Misplaced trust on frameworks/ third-party APIs (I4)	*"Developers assume that frameworks will handle security developers fully trust existing frameworks with their applications' security and thus take security for granted."[23]*
	A fair number of our freelancers argued that they trust standards and third party APIs to do the right thing and store passwords securely... However, this trust is sometimes misplaced." [108]
	"...developers tend to blindly trust code from a reputable source, e.g., API code." [115]
	"...developers fully trust existing frameworks with their applications' security and thus take security for granted." [23]
Lack of clarity about regulations (I5)	*"When asked about current and upcoming privacy and security regulations, participants showed little knowledge." [30]*
	"developers who have some knowledge of legal requirements are less likely to make accidental errors that violate regulations." [11], hence, developers are likely to make accidental errors if they have more knowledge of legal requirements
Lack of domain knowledge (I6)	*"S1 showed that another obstacle faced by developers is lack of domain knowledge."* [106]
	..it becomes a question of which defenses to implement: where one should spend the time and effort defending the system to deter the largest and most damaging potential exploits. Making those choices requires an understanding of the potential attackers... Neither attacker profiles not attack descriptions, however, are conventional knowledge for a software developer." [173]
Lack of experience (with tools, especially security tools, APIs, and programming languages) (I7)	"*Specific SAT experience more than doubled the number of correct answers and a combination of security experience and SAT experience almost tripled the number of correct security answers"* [27], hence the number of security solutions can be affected by the lack of security and SAT experience.
	Developers think that they are not able to implement correct security mechanisms, as they have less development experience. [78]
	"We did, however, find a significant effect for Python experience." [11]

(Continued)

Table 6. Continued

Knowledge Deficit	
External Factors	**Instances**
Lack of awareness of Security Tools and *vulnerabilities* (I8)	*"participants wanted to determine whether certain input parameters were ever validated, but were not aware of any tools to assist in this process.* [148]
	Table 6 shows that many of the affected projects were unaware of the vulnerability to their software. [92]
	"Three interviewees who used dynamic programming languages did not use security tools because they did not believe good security tools existed for these languages." [178]
Inadequate information sources (E1)	Insecure advice on online social forums. [45, 63]
	"web search engines were not fully aware of participants' programming contexts, they returned information about a superset of the relevant attacks" [148] and the developer may be distracted by irrelevant information. [148]
	"official and book participants said they would have liked to access Stack Overflow or search engines such as Google, so that they could search for their specific problems rather than reading background information." [10]
	"Different standards and security recommendations make it difficult for developers to decide what is the right course of action, which creates frustration." [109]
	"Similar to students' solutions from the lab study, we identified freelancers' security code on the Internet." [108]
	One book user mentioned the 'danger that books could be outdated." [10]
	Variance in coverage of different topics. [10] Laws and regulations are mentioned in only two guides. [12]
	"Despite years of intensive research and commercial development creating and improving program-analysis tools, these tools are also only mentioned in seven guides." [12] Lack of continuous feedback *"To keep apps secure requires continuous feedback, both to detect actual exploits, and to detect trends of use that may represent longer-term threats. Getting such feedback is much more difficult with mobile apps than with servers....changes to the supporting environment often have security implications requiring changes to apps to support them."* [173]
Lack of information sharing among teams (E2)	Sometimes *"organizational politics"* can make security issues worse as one team does not share *"all the facts"* with others.[173]
Usability issues with security tools and APIs (E11)	*"Participants' main obstacles are lack of high-level APIs, poor documentation, and bad API design (e.g., misleading defaults and difficult debugging)."* [106]
	"another obstacle faced by developers is lack of domain knowledge." [106]
	"...respondents generally state they had problems just 'doing it right.' Eleven others explained in more detail that they have struggled to understand or use an API." [78]
	"When participants navigated through chains of method invocations, they were forced to choose between different tools, where each tool had specific advantages and disadvantages." [148]
	"..the default behavior in crypto-graphic libraries is often not a recommended practice." [60]
	Insufficient documentation *"..the default behavior in crypto-graphic libraries is often not a recommended practice."* [60]
	Defaults are not recommended practice *".. the default behavior in crypto- graphic libraries is often not a recommended practice."* [60]

Table 7. Factors Associated with Attention Deficit

Attention Deficit	
Internal	**Instances**
Not identifying security blindspots in tasks (I9)	*"... some participants erroneously considered only a subset of the possible attacks ... By failing to consider cross site scripting attacks, participants overlooked the program path that exposed a true attack."* [148]
	"I knew about it because I happen to work on another project where we had to fix this very problem, but I didn't connect two dots." [92]
	"Developers assume common cases..as software vulnerabilities often lie in the corner cases and unusual information flows, they tend to be left out from developers' heuristics." [115]
	"...security education helps, but developers can have difficulties correlating particular learned vulnerability or security information with their working task." [115]

(Continued)

Table 7. Continued

Attention Deficit	
Internal	**Instances**
	"The presence of blindspots correlated negatively with the developers' accuracy in answering implicit security questions and the developers' ability to identify potential security concerns in the code." [116]
	"...a team will always suffer to some extent from 'groupthink'; the need to generate a shared understanding brings with it the danger that...[it] may include misunderstandings and blind spots." [173]
	"It is notoriously difficult to spot one's own errors... especially true when the errors are faults in complex reasoning, or are due to misunderstandings. Thus, a programmer working solo is likely to create avoidable security problems, just because they can naturally have only one point of view." [173]
Not handling cognitive load (I10)	*"Regardless of the strategies and tools participants used, they had to manually track their progress on each task ... Participants had to reason about each of those attacks individually and remember which attacks they had ruled out."[148]*
	"security thinking requires cognitive effort." [115]
	"... to facilitate remembering details, and to understand what the system is doing in different situations, developers bypass security features and add information to the user interface. This increase in usability facilitates testing but also introduces security vulnerabilities because there is a risk that these "improvements" are forgotten and later fielded." [85]
	".. developers perceive the practice of updating their dependencies as added effort and responsibilities that should be performed in their 'spare time.'" [92]
Developer's non-secure routines (I11)	*"Participants used error-prone strategies even when more reliable tools and strategies were available."* [148]
Lack of curiosity (I12)	*"Developers exhibiting greater openness as a personality trait were more likely to detect API blind spots"* [116]
	"We found security tool adopters were more likely than non-adopters to say they actively sought out information about security tools." [179]
	"...a developer's inquisitiveness about security tools affects the likelihood she will adopt a security tool."[178]
Loss of focus on security (I13)	*"most of our participants focused on the functionality and only added security as an afterthought—even those who were primed for security."* [108]
	"security is not part of developer's mindset while coding." [115]
	"...most of our participants focused on the functionality and only added security as an afterthought." [109]
	"..so even if I might have heard of the current vulnerability, I simply forgot to address it." [92]
	"So as the time or budget is limited, software security is one of the concerns that get overlooked, either explicitly or implicitly." [185]
External Factors	**Instances**
Task complexity (E3)	*"The complexity of the application... were named as reasons for this."* (insecure storage) [108]
	"responses... also provide evidence that the complex nature of these inter-dependencies (colloquially termed as 'dependency hell') is an influencing factor on whether or not a dependency will be migrated." [92]
Lack of division of labour (E4)	*"This caused a first gap between the individual groups and the management: teams enjoyed autonomy but as a consequence management attributed security to them as an intrinsic quality requirement." [126]*
	"...there is a frequent danger that security problems can 'fall between two stools,' remaining ignored because two teams each think the other is responsible for the problem. The problem is exacerbated if the developers are not natural communicators." [173]
Absence of explicit expectation of secure coding (E5)	*"This study showed how priming security information when developers need it, on the spot, changed their approach towards security and adapted them to include security thinking in their repertoire of heuristics..." [115]*
	"freelancers do not store passwords securely unless prompted."[108]
	"Through the feedback, we find that out of the 16 responses, 11 (69%) immediately thanked us for the notification and proceeded to update their dependencies to the safer dependency versions." [92]
Limited Resources (E6)	*"Even self-described privacy advocates and security experts grappled with implementing privacy and security protection with limited time and resources." [30]*

Table 8. Factors Associated with Intent Deficit

Intent Deficit	
Internal	**Instances**
Loss of focus on security (I13)	*"As shown both by existing research ... and our results, analysis tools produce many false positives, and is one reason that developers are discouraged from using such tools."* [157]
	"So as the time or budget is limited, software security is one of the concerns that get overlooked, either explicitly or implicitly." [185]
Requires extra effort (I14)	*"...[developers] intentionally introduce complexity to avoid rewriting existing code, and misuse frameworks to fit their existing codebase without worrying about introducing vulnerabilities."* [23]
	"One developer described keeping up with privacy and security practices as a daunting task, saying, 'Unfortunately, I very rarely have time to actually sift through it and try to digest everything that's going on. I primarily rely on other people to let me know.'" [30]
	"...we find developers perceive the practice of updating their dependencies as added effort...[quoting a developer] 'Just that it's not very easy to keep track of it.' " [92]
Not perceiving usefulness of secure practices (I15)	*"Participants are motivated by the challenge or by their own values (e.g., to protect their users."* [24]
	"Most implied security motivation as a fundamental requirement." [170]
	"...interviewees who interacted frequently with security teams were more likely to use security tools, out of a greater sense of personal responsibility for security." [183]
	"Security auditors in our study struggled with convincing developers or other stakeholders that a security issue was real and in need of remediation. Many...participants mentioned that developers had difficulty in seeing a harmless example exploit and understanding that the vulnerability was serious." [157]
	"Most developers may not have a concrete understanding of the consequences of deploying insecure software..." [183]
	"*perceived negative consequences"* is among the key software security motivators, hence, without that perception, developers are not motivated to code securely. [24] *"respondents (had) little belief that privacy policies were useful."* [30]
	"...in general, adopters think security tools have a positive impact on their work," hence those who did not consider it positive were less likely to adopt security tools [179]
	"developers from the security inattentive group prioritise functionality and coding standards over security." [23]
	"...the remaining two projects stated that the update was unnecessary as the affected component had little impact on the project." [92]
	Both parties focused on short-to-market feature development; security could not (yet) serve this purpose. [126]
	"...interviewees said security tools generated many false positives, and one noted that this greatly increased the time 'cost' of tool use without increasing its benefits." [178]
	"...some developers reported that the software they developed was only used by authenticated, trusted users, so they thought security was not important. As a result, these developers did not use security tools." [178]
	"Perceived user base of the applications developed by software developers had an influence on participants' security tool adoption decisions as well." [183]
Perceived lack of own security (I16) knowledge	*"most frequent deterrents to software security were ... being unequipped for security because of a perceived lack of security knowledge." [24]*
	"Security tool adopters were more likely to report they were educated or had opportunities to continue their education in security," [179] hence developers without security education and opportunity were less likely to adopt security tools.
	"Several freelancers stated that they were unsure about the security of their solutions." [108]
Attitude of someone else's Responsibility (I17)	*"..some developers have security background, but do not apply their knowledge in practice, as it is neither considered their responsibility nor a priority."* [23]
	"another developer from the latter project deferred the responsibility to another project." [92]
	"If a requirement is important enough, somebody else can implement that." [101]

(Continued)

Table 8. Continued

Intent Deficit	
Internal	**Instances**
	"...there is a frequent danger that security problems can 'fall between two stools,' remaining ignored because two teams each think the other is responsible for the problem. The problem is exacerbated if the developers are not natural communicators." [173]
	"Non-users (of security tools) felt they could depend on code reviews." [183]
	"...key inhibitor toward secure software development practices is a it's not my responsibility attitude." [185]
	"...many stated that they are not responsible for security and they are not required to secure their applications." [23]
	"we find developers perceive the practice of updating their dependencies as added...responsibilities that should be performed in their 'spare time.'" [92]
	"...while security teams can make developers feel social pressure to code securely, as discussed previously, they also can make developers feel that security is not their responsibility." [183]
Absence of explicit expectation for secure coding (E5)	*"...expectation of security knowledge directly affects the degree of security integration in developers' tasks."* [23]
	We found that security prompting had a statistically significant effect on whether the participants stored the passwords in a secure way." [108]
	".... our participants relied on client requirements when deciding whether they wanted to store the passwords securely. Therefore, task description is a main motivator when deciding to deal with security." [108]
	"Despite the fact that storage of passwords should self-evidently be a security sensitive task, participants who were not explicitly told to employ secure storage stored the passwords in plain text." [109]
	"The second strongest effect in the survey was that adopters were more likely to report their superiors expect them to use security tools," hence if not expected, developers are less likely to adopt security tools. [179]
Limited Resources (E6)	*"Some participants said their team decides their security practices based on the available budget and/or employees who can perform security tasks." [23]*
	"balancing risk against resource limitations is a key security challenge." [157]
	"Many discussed privacy and security as being part of the development process but not a top priority, and concerns like monetizing the app or limited resources often trump the desire to follow rigorous privacy and security standards." [30]
	"Implied in every decision about software security is a trade-off of the cost of the security against the benefit received. Every security enhancement needs to be weighed against other uses of the investment (financial, time, usability) required." [173]
	"Five chose not to use security tools because the ones they had used were too slow or used too many resources ... Seven said mature security tools were too expensive to be cost-effective and chose not to use security tools at all ...." [178]
	"we find developers perceive the practice of updating their dependencies as added effort...that should be performed in their 'spare time.'" [92]
	"... lack of time ... were named as reasons for this." (i.e., to concentrate on functionality first). [92]
	"Business deadlines, planned budget, customer demands, and developer knowledge all impact the priorities for the limited resources of a project." [185]
	"The organizational environment, as it is responsible for a tight timescale ... and not counting security to business priorities." [78] *"So as the time or budget is limited, software security is one of the concerns that get overlooked, either explicitly or implicitly."* [185]
	..the lack of a viable replacement dependency are some of the possible reasons why affected maintainers show no response to the security advisory. [92]
Security Tools not reachable (E7)	*"Security tool adopters ... were permitted to try security tools to adequately see what they do"* [178, 179], hence developers who were not permitted to try security tools enough were not security tools adopters.
	"using a new tool requires written authorization which may take weeks so participant rarely investigated new tools." [183]
	"most frequent deterrents to software security were ... being unequipped for security because of ... the unavailability of necessary tools." [24]

(Continued)

Table 8. Continued

Intent Deficit	
Internal	**Instances**
Lack of security culture in teams/ companies (E8)	*"Organizational changes towards security influence adoption of security practices." [126]*
	"We revealed organizational and communication challenges to show that adopting secure development practices is not just a simple developer awareness problem, but requires dealing with complex organizational and social factors in software developing companies." [126]
	"developers who use security tools were more prestigious than those who do not." [179]
	"...frequency of interaction with security experts..maybe a more important influence...the strongest of all the relationships we found was with the statement 'I have seen what others do using security tools.'" [179]
	"lack of security can stem from systemic causes within the company or team, such as whether there are consequences for the lack of security, whether security is a priority, and if specific security plans exist." [24]
	"security was ignored or dismissed by developers' supervisors, despite the developer's expertise and interest." [23]
	"...interviews with managers confirmed that managerial involvement affected company culture around security." [183]
	"The organizational environment, as it is responsible for a tight timescale ... and not counting security to business priorities." [78]
Lack of prioritisation of security features by stakeholders (E9)	*"Product management perceived security as invisible to customers hence not marketable."* [126]
	"The organizational environment, as it is responsible for a tight timescale ... and not counting security to business priorities." [78]
	"Development management did not highlight security as something that could serve as a product feature" [126]
	"...security was not an important feature for small or immature applications." [157]
	"..and I'm willing to improve this, but sometimes it is difficult to explain our customers that it is a main point to consider in the development process." [92]
Lack of social influence (E10)	*"... developers who use security tools were more prestigious than those who do not [S9], to relatively large effect ..., and that using security tools improved their image within their organizations,"* hence those who did not feel prestige were less likely to adopt security tools. [179]
	"...the strongest of all the relationships we found was with the statement "I have seen what others do using security tools,"' hence developers who did no observe peers were less likely to adopt security tools. [179]
	"Twenty-two of thirty participants reported that they had recommended security tools to other developers," hence developers recommend security tools to each other. [183]
	"We also had some instances of what is likely to be a manifestation of the social desirability bias while answering survey questions. Several freelancers stated that they store passwords securely even if not explicitly instructed to... However, these participants sent insecure solutions as their first submissions." [108]
	"One developer described keeping up with privacy and security practices as a daunting task, saying, '... I primarily rely on other people to let me know.'"[29]
	"...interviewees who interacted frequently with security teams were more likely to use security tools, out of a greater sense of personal responsibility for security." [183]
	"Especially in cases where ordinary developers interacted often with the security team, participants reported that they felt social pressure from the security team by having them audit the code and review the new features from a security perspective. This ultimately made developers feel personally responsible for their code's security." [183]
Usability issues with security tools and APIs (E11)	Some interviewees did not adopt security tools because of the tools' complexity. [178]
	"Several participants ... mentioned that it was very difficult for developers to interpret the results of dynamic analysis scans, locate the problem in the code, and then apply the correct fix." [157]
	"APIs are too complex to use." [106]
	"developers seem to have problems in correctly setting up their environments to use the APIs." [106]
	Difficulty in choosing the right tool or combination of tools. [148]

ACKNOWLEDGMENTS

This work benetted from conversations with Arosha Bandara, Joseph Hallett, and CharlesWeir.

REFERENCES

[1] Stack Overflow. 2019. *Developer Survey Results*. Retrieved from https://insights.stackoverflow.com/survey/2019.
[2] Dave Wichers. 2020. Free for Open Source Application Security Tools | OWASP. Retrieved January, 2020 from https://owasp.org/www-community/Free_for_Open_Source_Application_Security_Tools.
[3] OWASP. 2020. *Source Code Analysis Tools | OWASP*. Retrieved January, 2020 from https://owasp.org/www-community/Source_Code_Analysis_Tools.
[4] Veracode. 2020. *State Of Software Security*. Retrieved January, 2020 from https://tinyurl.com/uaa4ock.
[5] OWASP. 2020. *Vulnerability Scanning Tools | OWASP*. Retrieved January, 2020 from https://owasp.org/www-community/Vulnerability_Scanning_Tools.
[6] Fortify. 2020. *Fortify Taxonomy: Software Security Errors*. Retrieved January, 2020 from https://vulncat.fortify.com/en.w.
[7] Mohd Syazwan Abdullah, Ian Benest, Andy Evans, and Chris Kimble. 2002. Knowledge modelling techniques for developing knowledge management systems. In *3rd European Conference on Knowledge Management*.
[8] Dominic Abrams, Margaret Wetherell, Sandra Cochrane, Michael A. Hogg, and John C. Turner. 1990. Knowing what to think by knowing who you are: Self-categorization and the nature of norm formation, conformity and group polarization. *British J. Soc. Psychol.* 29, 2 (1990), 97–119.
[9] Yasemin Acar, Michael Backes, Sascha Fahl, Simson Garfinkel, Doowon Kim, Michelle L. Mazurek, and Christian Stransky. 2017. Comparing the usability of cryptographic apis. In *IEEE Symposium on Security and Privacy (SP)*. IEEE, 154–171.
[10] Yasemin Acar, Michael Backes, Sascha Fahl, Doowon Kim, Michelle L. Mazurek, and Christian Stransky. 2016. You get where you're looking for: The impact of information sources on code security. In *IEEE Symposium on Security and Privacy (SP)*. IEEE, 289–305.
[11] Yasemin Acar, Christian Stransky, Dominik Wermke, Michelle L. Mazurek, and Sascha Fahl. 2017. Security developer studies with github users: Exploring a convenience sample. In *13th Symposium on Usable Privacy and Security (SOUPS'17)*. 81–95.
[12] Yasemin Acar, Christian Stransky, Dominik Wermke, Charles Weir, Michelle L. Mazurek, and Sascha Fahl. 2017. Developers need support, too: A survey of security advice for software developers. In *IEEE Cybersecurity Development (SecDev)*. IEEE, 22–26.
[13] Icek Ajzen. 1991. The theory of planned behavior. *Organiz. Behav. Hum. Decis. Process.* 50, 2 (1991), 179–211.
[14] I. Ajzen. 2015. The theory of planned behavior: A bibliography: 1985–2015. Retrieved from https://people.umass.edu/aizen/tpbrefstxt.html.
[15] Pierre A. Akiki, Arosha K. Bandara, and Yijun Yu. 2014. Adaptive model-driven user interface development systems. *ACM Comput. Surv.* 47, 1 (2014), 9.
[16] Ahmad Al-Ahmad, Belal Abu Ata, and Abdullah Wahbeh. 2012. Pen testing for web applications. *Int. J. Inf. Technol. Web Eng.* 7, 3 (2012), 1–13.
[17] Erik M. Altmann, J. Gregory Trafton, and David Z. Hambrick. 2014. Momentary interruptions can derail the train of thought. *J. Experim. Psychol.: Gen.* 143, 1 (2014), 215.
[18] Rozaliya Amirova. 2020. Attention tracking for developers. In *28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, Prem Devanbu, Myra B. Cohen, and Thomas Zimmermann (Eds.). ACM, 1690–1692. DOI: https://doi.org/10.1145/3368089.3418778
[19] John Robert Anderson. 1990. *The Adaptive Character of Thought*. Psychology Press.
[20] John R. Anderson and Christian J. Lebiere. 2014. *The Atomic Components of Thought*. Psychology Press.
[21] Jackie Andrade and Jon May. 2004. *BIOS Instant Notes in Cognitive Psychology*. Taylor & Francis.
[22] Brad Arkin, Scott Stender, and Gary McGraw. 2005. Software penetration testing. *IEEE Secur. Priv.* 3, 1 (2005), 84–87.
[23] Hala Assal and Sonia Chiasson. 2018. Security in the software development lifecycle. In *14th Symposium on Usable Privacy and Security (SOUPS'18)*. USENIX Association, 281–296.
[24] Halaa Assal and Sonia Chiasson. 2019. "Think secure from the beginning": A survey with software developers. In *CHI Conference on Human Factors in Computing Systems*. ACM, 289.
[25] Hala Assal, Sonia Chiasson, and Robert Biddle. 2016. Cesar: Visual representation of source code vulnerabilities. In *IEEE Symposium on Visualization for Cyber Security (VizSec)*. IEEE, 1–8.
[26] Edward Awh, Edward K. Vogel, and S.-H. Oh. 2006. Interactions between attention and working memory. *Neuroscience* 139, 1 (2006), 201–208.

[27] Dejan Baca, Kai Petersen, Bengt Carlsson, and Lars Lundberg. 2009. Static code analysis to detect software security vulnerabilities-does experience matter? In *International Conference on Availability, Reliability and Security*. IEEE, 804–810.
[28] Alan Baddeley. 2007. *Working Memory, Thought, and Action*. Vol. 45. OuP Oxford.
[29] Rebecca Balebako and Lorrie Cranor. 2014. Improving app privacy: Nudging app developers to protect user privacy. *IEEE Secur. Priv.* 12, 4 (2014), 55–58.
[30] Rebecca Balebako, Abigail Marsh, Jialiu Lin, Jason I. Hong, and Lorrie Cranor. 2014. The privacy and security behaviors of smartphone app developers. In *Workshop on Usable Security (USEC)*.
[31] Gordon Baxter and Ian Sommerville. 2011. Socio-technical systems: From design methods to systems engineering. *Interact. Comput.* 23, 1 (2011), 4–17.
[32] Sarah Beecham, Nathan Baddoo, Tracy Hall, Hugh Robinson, and Helen Sharp. 2008. Motivation in software engineering: A systematic literature review. *Inf. Softw. Technol.* 50, 9-10 (2008), 860–878.
[33] Laura M. Bishop, Phillip L. Morgan, Phoebe M. Asquith, George Raywood-Burke, Adam Wedgbury, and Kevin Jones. 2020. Examining human individual differences in cyber security and possible implications for human-machine interface design. In *International Conference on Human-computer Interaction*. Springer, 51–66.
[34] Matt Bishop. 2010. A clinic for "secure" programming. *IEEE Secur. Priv.* 8, 2 (2010), 54–56.
[35] Alan F. Blackwell. 2002. First steps in programming: A rationale for attention investment models. In *IEEE Symposia on Human Centric Computing Languages and Environments*. IEEE, 2–10.
[36] James Blake. 1999. Overcoming the "value-action gap" in environmental policy: Tensions between national policy and local experience. *Local Environ.* 4, 3 (1999), 257–278.
[37] Andrew Booth, Anthea Sutton, and Diana Papaioannou. 2016. Systematic approaches to a successful literature review. SAGE. Retrieval: https://www.google.co.uk/books/edition/_/JD1DCgAAQBAJ?hl=en&gbpv=0.
[38] Michael Bosnjak, Icek Ajzen, and Peter Schmidt. 2020. The theory of planned behavior: selected recent advances and applications. *Europe's J. Psychol.* 16, 3 (2020), 352–356.
[39] Steven A. Brieger. 2019. Social identity and environmental concern: The importance of contextual effects. *Environ. Behav.* 51, 7 (2019), 828–855.
[40] Rupert Brown. 2020. The social identity approach: Appraising the Tajfellian legacy. *British J. Soc. Psychol.* 59, 1 (2020), 5–25. DOI: https://doi.org/10.1111/bjso.12349
[41] Marcel Bruch, Eric Bodden, Martin Monperrus, and Mira Mezini. 2010. IDE 2.0: Collective intelligence in software development. In *FSE/SDP Workshop on Future of Software Engineering Research*. ACM, 53–58.
[42] Evan A. Byrne and Raja Parasuraman. 1996. Psychophysiology and adaptive automation. *Biol. Psychol.* 42, 3 (1996), 249–268.
[43] Gadiel Sznaier Camps, Nicolas Bohm Agostini, and David Kaeli. 2019. Discovering programmer intention behind written source code. In *18th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 432–437.
[44] OWASP Foundation. 2020. *Category: Vulnerability Scanning Tools - OWASP*. Retrieved January 2020 from https://www.owasp.org/index.php/Category:Vulnerability_Scanning_Tools.
[45] Mengsu Chen, Felix Fischer, Na Meng, Xiaoyin Wang, and Jens Grossklags. 2019. How reliable is the crowdsourced knowledge of security implementation? In *Proceedings of the 41st International Conference on Software Engineering*. IEEE Press, 536–547.
[46] Robert B. Cialdini. 1985. *Influence*. Scott, Foresman and Company, Glenview, IL.
[47] Robert B. Cialdini, Carl A. Kallgren, and Raymond R. Reno. 1991. A focus theory of normative conduct: A theoretical refinement and reevaluation of the role of norms in human behavior. In *Advances in Experimental Social Psychology*. Vol. 24. Academic Press, San Diego, CA, 201–234.
[48] Robert B. Cialdini and Melanie R. Trost. 1998. Social influence: Social norms, conformity and compliance. In *Handbook of Social Psychology*. Vol. 2, D. T. Gilbert, S. T. Fiske, G. Lindzey (Eds.). McGraw-Hill, Boston, MA, 151–192).
[49] Andrew Conway, Chris Jarrold, Michael Kane, Akira Miyake, and John Towse. 2008. *Variation in Working Memory*. Oxford University Press.
[50] The MITRE Corporation. 2020. Cybersecurity Resources. Retrieved January 2020 from https://www.mitre.org/capabilities/cybersecurity/overview/cybersecurity-resources.
[51] Ricardo Couceiro, Gonçalo Duarte, João Durães, João Castelhano, Catarina Duarte, Cesar Teixeira, Miguel Castelo Branco, Paulo Carvalho, and Henrique Madeira. 2019. Pupillography as indicator of programmers' mental effort and cognitive overload. In *49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 638–644.
[52] Kevin Crowston and Ericka Eve Kammerer. 1998. Coordination and collective mind in software requirements development. *IBM Syst. J.* 37, 2 (1998), 227–245.

[53] Shirley Cruz, Fabio Q. B. da Silva, and Luiz Fernando Capretz. 2015. Forty years of research on personality in software engineering: A mapping study. *Comput. Hum. Behav.* 46 (2015), 94–113.
[54] CVE - Common Vulnerabilities and Exposures (CVE). 2020. Retrieved January 2020 from https://cve.mitre.org/.
[55] Meredyth Daneman and Patricia A. Carpenter. 1980. Individual differences in working memory and reading. *J. Mem. Lang.* 19, 4 (1980), 450.
[56] Rogério de Lemos, Holger Giese, Hausi A. Müller, Mary Shaw, Jesper Andersson, Luciano Baresi, Basil Becker, Nelly Bencomo, Yuriy Brun, Bojan Cukic, Ronald J. Desmarais, Schahram Dustdar, Gregor Engels, Kurt Geihs, Karl M. Göschka, Alessandra Gorla, Vincenzo Grassi, Paola Inverardi, Gabor Karsai, Jeff Kramer, Marin Litoiu, Antónia Lopes, Jeff Magee, Sam Malek, Serge Mankovskii, Raffaela Mirandola, John Mylopoulos, Oscar Nierstrasz, and Ma. 2010. Software engineering for self-adaptive systems: A second research roadmap. In *Software Engineering for Self-Adaptive Systems (Dagstuhl Seminar Proceedings, Vol. 10431)*, Rogério de Lemos, Holger Giese, Hausi A. Müller, and Mary Shaw (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany. Retrieved from http://drops.dagstuhl.de/opus/volltexte/2011/3156/.
[57] A. P. Dijksterhuis and Henk Aarts. 2010. Goals, attention, and (un) consciousness. *Ann. Rev. Psychol.* 61 (2010), 467–490.
[58] Wenliang Du and Ronghua Wang. 2008. SEED: A suite of instructional laboratories for computer security education. *J. Educ. Resour. Comput.* 8, 1 (2008), 3.
[59] John Duncan, Hazel Emslie, Phyllis Williams, Roger Johnson, and Charles Freer. 1996. Intelligence and the frontal lobe: The organization of goal-directed behavior. *Cog. Psychol.* 30, 3 (1996), 257–303.
[60] Manuel Egele, David Brumley, Yanick Fratantonio, and Christopher Kruegel. 2013. An empirical study of cryptographic misuse in Android applications. In *ACM SIGSAC Conference on Computer & Communications Security*. ACM, 73–84.
[61] Hewlett Packard Enterprise. 2015. Awareness is only the first step: A framework for progressive engagement of staff in cyber security. Retrieved from https://www.riscs.org.uk/wp-content/uploads/2015/12/Awareness-is-Only-the-First-1713Step.pdf.
[62] Rino Falcone and Cristiano Castelfranchi. 2001. The human in the loop of a delegated agent: The theory of adjustable social autonomy. *IEEE Trans. Syst., Man, Cyber.-Part A: Syst. Hum.* 31, 5 (2001), 406–418.
[63] Felix Fischer, Konstantin Böttinger, Huang Xiao, Christian Stransky, Yasemin Acar, Michael Backes, and Sascha Fahl. 2017. Stack overflow considered harmful? The impact of copy&paste on Android application security. In *IEEE Symposium on Security and Privacy (SP)*. IEEE, 121–136.
[64] Jose Fonseca, Marco Vieira, and Henrique Madeira. 2007. Testing and comparing web vulnerability scanning tools for SQL injection and XSS attacks. In *13th Pacific Rim International Symposium on Dependable Computing (PRDC07)*. IEEE, 365–372.
[65] Michael Frese and Dieter Zapf. 1994. Action as the core of work psychology: A German approach. *Handb. Industr. Organiz. Psychol.* 4, 2 (1994), 271–340.
[66] Krzysztof Z. Gajos, Daniel S. Weld, and Jacob O. Wobbrock. 2010. Automatically generating personalized user interfaces with Supple. *Artif. Intell.* 174, 12-13 (2010), 910–950.
[67] Vaibhav Garg and Jean Camp. 2013. Heuristics and biases: Implications for security design. *IEEE Technol. Soc. Mag.* 32, 1 (2013), 73–79.
[68] Patrice Godefroid. 2007. Random testing for security: Blackbox vs. whitebox fuzzing. In *2nd International Workshop on Random Testing, co-located with the 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE'07)*. ACM, 1–1.
[69] Peter Leo Gorski, Luigi Lo Iacono, Dominik Wermke, Christian Stransky, Sebastian Möller, Yasemin Acar, and Sascha Fahl. 2018. Developers deserve security warnings, too: On the Effect of Integrated Security Advice on Cryptographic {API} Misuse. In *14th Symposium on Usable Privacy and Security (SOUPS'18)*. 265–281.
[70] Jerold L. Hale, Brian J. Householder, and Kathryn L. Greene. 2002. The theory of reasoned action. *Persuas. Handb.: Devel. Theor. Pract.* 14 (2002), 259–286.
[71] Charles Haley, Robin Laney, Jonathan Moffett, and Bashar Nuseibeh. 2008. Security requirements engineering: A framework for representation and analysis. *IEEE Trans. Softw. Eng.* 34, 1 (2008), 133–153.
[72] Julie M. Haney and Wayne G. Lutters. 2017. Skills and characteristics of successful cybersecurity advocates. In *Symposium on Usable Privacy and Security (SOUPS'17)*.
[73] Julie M. Haney, Mary Theofanos, Yasemin Acar, and Sandra Spickard Prettyman. 2018. "We make it a big deal in the company": Security mindsets in organizations that develop cryptographic products. In *14th Symposium on Usable Privacy and Security (SOUPS'18)*. 357–373.
[74] S. Alexander Haslam. 2001. *Psychology in Organizations: The Social Identity Approach*. London: Sage, London.
[75] J.-M. Hoc. 2014. *Psychology of Programming*. Academic Press.
[76] Thomas Hofer. 2010. *Evaluating Static Source Code Analysis Tools*. Technical Report. EPFL, Switzerland.

[77] Yao-Wen Huang, Shih-Kun Huang, Tsung-Po Lin, and Chung-Hung Tsai. 2003. Web application security assessment by fault injection and behavior monitoring. In *12th International Conference on World Wide Web*. ACM, 148–159.
[78] Luigi Lo Iacono and Peter Leo Gorski. 2017. I do and I understand. not yet true for security APIs. So sad. In *2nd European Workshop on Usable Security*. DOI: https://doi.org/10.14722/eurousec
[79] William James. 2007. *The Principles of Psychology*. Vol. 1. Cosimo, Inc., 2007.
[80] Jolanda Jetten, Catherine Haslam, and S. Haslam Alexander. 2012. *The Social Cure: Identity, Health and Well-being*. Psychology Press.
[81] Philip N. Johnson-Laird and Ruth M. J. Byrne. 1993. Precis of deduction. *Behav. Brain Sci.* 16, 2 (1993), 323–333.
[82] Russell L. Jones and Abhinav Rastogi. 2004. Secure coding: Building security into the software development life cycle. *Inf. Syst. Secur.* 13, 5 (2004), 29–39.
[83] Jan Jurjens. 2006. Security analysis of crypto-based Java programs using automated theorem provers. In *21st IEEE/ACM International Conference on Automated Software Engineering (ASE'06)*. IEEE, 167–176.
[84] Daniel Kahneman and Amos Tversky. 1996. On the reality of cognitive illusions. *Psychol. Rev.* 103, 3 (1996), 582–591.
[85] Kaarina Karppinen, Lyly Yonkwa, and Mikael Lindvall. 2009. Why developers insert security vulnerabilities into their code. In *2nd International Conference on Advances in Computer-human Interactions*. IEEE, 289–294.
[86] Tara Kennedy, Glenn Regehr, Jay Rosenfield, S. Wendy Roberts, and Lorelei Lingard. 2004. Exploring the gap between knowledge and behavior: A qualitative study of clinician action following an educational intervention. *Acad. Med.* 79, 5 (2004), 386–393.
[87] Iacovos Kirlappos, Simon Parkin, and M. Angela Sasse. 2014. Learning from "Shadow Security": Why understanding non-compliance provides the basis for effective security. In *Proceedings Workshop on Usable Security*. Retrieval https://discovery.ucl.ac.uk/id/eprint/1424472/.
[88] Barbara Kitchenham and Stuart Charters. 2007. Guidelines for performing systematic literature reviews in software engineering. Citeseer. Retrieval info https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.117.471&rep=rep1&type=pdf.
[89] Agata Kołakowska. 2016. Towards detecting programmers' stress on the basis of keystroke dynamics. In *Federated Conference on Computer Science and Information Systems (FedCSIS)*. IEEE, 1621–1626.
[90] Anja Kollmuss and Julian Agyeman. 2002. Mind the gap: Why do people act environmentally and what are the barriers to pro-environmental behavior? *Environ. Educ. Res.* 8, 3 (2002), 239–260.
[91] Stefan Krüger, Sarah Nadi, Michael Reif, Karim Ali, Mira Mezini, Eric Bodden, Florian Göpfert, Felix Günther, Christian Weinert, Daniel Demmler, et al. 2017. CogniCrypt: Supporting developers in using cryptography. In *32nd IEEE/ACM International Conference on Automated Software Engineering*. IEEE Press, 931–936.
[92] Raula Gaikovina Kula, Daniel M. German, Ali Ouni, Takashi Ishio, and Katsuro Inoue. 2018. Do developers update their library dependencies? *Empir. Softw. Eng.* 23, 1 (2018), 384–417.
[93] Patrick C. Kyllonen and Raymond E. Christal. 1990. Reasoning ability is (little more than) working-memory capacity?! *Intelligence* 14, 4 (1990), 389–433.
[94] Per Lenberg, Robert Feldt, and Lars Göran Wallgren. 2015. Behavioral software engineering: A definition and systematic literature review. *J. Syst. Softw.* 107 (2015), 15–37.
[95] Timothy C. Lethbridge. 2000. Priorities for the education and training of software engineers. *J. Syst. Softw.* 53, 1 (2000), 53–71.
[96] Peng Li and Baojiang Cui. 2010. A comparative study on software vulnerability static analysis techniques and tools. In *IEEE International Conference on Information Theory and Information Security (ICITIS)*. IEEE, 521–524.
[97] Simon Y. W. Li, Ann Blandford, Paul Cairns, and Richard M. Young. 2008. The effect of interruptions on postcompletion and other procedural errors: An account based on the activation-based goal memory model. *J. Experim. Psychol.: Appl.* 14, 4 (2008), 314.
[98] Tong Li, Jennifer Horkoff, and John Mylopoulos. 2018. Holistic security requirements analysis for socio-technical systems. *Softw. Syst. Model.* 17, 4 (2018), 1253–1285.
[99] Tamara Lopez, Thein Tun, Arosha Bandara, Levine Mark, Bashar Nuseibeh, and Helen Sharp. 2019. An anatomy of security conversations in stack overflow. In *IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Society (ICSE-SEIS)*. IEEE, 31–40.
[100] Tamara Lopez, Thein T. Tun, Arosha Bandara, Mark Levine, Bashar Nuseibeh, and Helen Sharp. 2018. An investigation of security conversations in stack overflow: Perceptions of security and community involvement. In *1st International Workshop on Security Awareness from Design to Deployment*. ACM, 26–32.
[101] Kai-Uwe Loser and Martin Degeling. 2014. Security and privacy as hygiene factors of developer behavior in small and agile teams. In *IFIP International Conference on Human Choice and Computers*. Springer, 255–265.
[102] Shinsuke Matsumoto, Yasutaka Kamei, Akito Monden, Ken-ichi Matsumoto, and Masahide Nakamura. 2010. An analysis of developer metrics for fault prediction. In *6th International Conference on Predictive Models in Software Engineering*. ACM, 18.

[103] Gary McGraw. 2006. *Software Security: Building Security in*. Vol. 1. Addison-Wesley Professional.
[104] Jennifer C. McVay and Michael J. Kane. 2009. Conducting the train of thought: Working memory capacity, goal neglect, and mind wandering in an executive-control task. *J. Experim. Psychol.: Learn., Mem., Cogn.* 35, 1 (2009), 196.
[105] Susan Michie, Maartje M. Van Stralen, and Robert West. 2011. The behaviour change wheel: A new method for characterising and designing behaviour change interventions. *Implement. Sci.* 6, 1 (2011), 42.
[106] Sarah Nadi, Stefan Krüger, Mira Mezini, and Eric Bodden. 2016. Jumping through hoops: Why do Java developers struggle with cryptography APIs? In *38th International Conference on Software Engineering*. ACM, 935–946.
[107] Alena Naiakshina, Anastasia Danilova, Eva Gerlitz, and Matthew Smith. 2020. On conducting security developer studies with CS students: Examining a password-storage study with CS students, freelancers, and company developers. In *CHI Conference on Human Factors in Computing Systems*. 1–13.
[108] Alena Naiakshina, Anastasia Danilova, Eva Gerlitz, Emanuel von Zezschwitz, and Matthew Smith. 2019. "If you want, I can store the encrypted password": A password-storage field study with freelance developers. In *CHI Conference on Human Factors in Computing Systems*. ACM, 140.
[109] Alena Naiakshina, Anastasia Danilova, Christian Tiefenau, Marco Herzog, Sergej Dechand, and Matthew Smith. 2017. Why do developers get password storage wrong?: A qualitative usability study. In *ACM SIGSAC Conference on Computer and Communications Security*. ACM, 311–328.
[110] Stephan Neuhaus, Thomas Zimmermann, Christian Holler, and Andreas Zeller. 2007. Predicting vulnerable software components. In *14th ACM Conference on Computer and Communications Security*. ACM, 529–540.
[111] Fergus G. Neville. 2015. Preventing violence through changing social norms. *Oxford Textbook of Violence Prevention: Epidemiology, Evidence and Policy*, P. Donnelly and C. Ward (Eds.) Oxford University Press, 239–244.
[112] Duc Cuong Nguyen, Dominik Wermke, Yasemin Acar, Michael Backes, Charles Weir, and Sascha Fahl. 2017. A stitch in time: Supporting Android developers in writingsecure code. In *ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1065–1077.
[113] Dennis Nigbur, Evanthia Lyons, and David Uzzell. 2010. Attitudes, norms, identity and environmental behaviour: Using an expanded theory of planned behaviour to predict participation in a kerbside recycling programme. *British J. Soc. Psychol.* 49, 2 (2010), 259–284.
[114] Donald A. Norman. 1981. Categorization of action slips. *Psychol. Rev.* 88, 1 (1981), 1.
[115] Daniela Oliveira, Marissa Rosenthal, Nicole Morin, Kuo-Chuan Yeh, Justin Cappos, and Yanyan Zhuang. 2014. It's the psychology, stupid: How heuristics explain software vulnerabilities and how priming can illuminate developer's blind spots. In *Computer Security Applications Conference*. ACM, 296–305.
[116] Daniela Seabra Oliveira, Tian Lin, Muhammad Sajidur Rahman, Rad Akefirad, Donovan Ellis, Eliany Perez, Rahul Bobhate, Lois A. DeLong, Justin Cappos, and Yuriy Brun. 2018. *API* Blindspots: Why experienced developers write vulnerable code. In *14th Symposium on Usable Privacy and Security (SOUPS'18)*. 315–328.
[117] OWASP. (Accessed on: January, 2020). Source Code Analysis Tools | OWASP. Retrieved from https://owasp.org/www-community/Source_Code_Analysis_Tools.
[118] OWASP Foundation. 2020. The Open Source Foundation for Application Security. Retrieved from https://owasp.org/.
[119] OWASP Secure Coding Practices - Quick Reference Guide. 2020. Retrieved January, 2020 from https://www.owasp.org/index.php/OWASP_Secure_Coding_Practices_-_Quick_Reference_Guide.
[120] Harold E. Pashler. 1999. *The Psychology of Attention*. The MIT Press.
[121] Steven Pemberton. 1996. Programmers are humans too. *ACM SIGCHI Bull.* 28, 1 (1996), 96.
[122] Thomas F. Pettigrew. 2018. The emergence of contextual social psychology. *Personal. Soc. Psychol. Bull.* 44, 7 (2018), 963–971.
[123] Shari Lawrence Pfleeger, M. Angela Sasse, and Adrian Furnham. 2014. From weakest link to security hero: Transforming staff security behavior. *J. Homel. Secur. Emerg. Manag.* 11, 4 (2014), 489–510.
[124] Olgierd Pieczul, Simon Foley, and Mary Ellen Zurko. 2017. Developer-centered security and the symmetry of ignorance. In *New Security Paradigms Workshop*. 46–56.
[125] Frank Piessens. 2019. The Cyber Security Body of Knowledge, Software Security Knowledge Area Issue 1.0. (2019). Retrieval Info: https://www.cybok.org/media/downloads/cybok_version_1.0.pdf.
[126] Andreas Poller, Laura Kocksch, Katharina Kinder-Kurlanda, and Felix Anand Epp. 2016. First-time security audits as a turning point?: Challenges for security practices in an industry software development team. In *CHI Conference Extended Abstracts on Human Factors in Computing Systems*. ACM, 1288–1294.
[127] Andreas Poller, Laura Kocksch, Sven Türpe, Felix Anand Epp, and Katharina Kinder-Kurlanda. 2017. Can security become a routine? A study of organizational change in an agile software development group. In *ACM Conference on Computer Supported Cooperative Work and Social Computing*. 2489–2503.
[128] Irum Rauf, Elena Troubitsyna, and Ivan Porres. 2019. A systematic mapping study of API usability evaluation methods. *Comput. Sci. Rev.iew* 33 (2019), 49–68.

[129] Irum Rauf, Dirk van der Linden, Mark Levine, John Towse, Bashar Nuseibeh, and Awais Rashid. 2020. The impact of social considerations on app developers' choices. In *42nd International Conference on Software Engineering Workshops (ICSEW'20)*.
[130] Stephen Reicher, Russell Spears, and S. Alexander Haslam. 2010. The social identity approach in social psychology. *Sage Ident. Handb.* (2010), 45–62.
[131] Allecia E. Reid, Robert B. Cialdini, and Leona S. Aiken. 2010. Social norms and health behavior. In *Handbook of Behavioral Medicine*. Springer, New York, NY.
[132] Katharina Reinecke and Abraham Bernstein. 2011. Improving performance, perceived usability, and aesthetics with culturally adaptive user interfaces. *ACM Trans. Comput.-Hum. interact.* 18, 2 (2011).
[133] Christelle Robert, Erika Borella, Delphine Fagot, Thierry Lecerf, and Anik De Ribaupierre. 2009. Working memory and inhibitory control across the life span: Intrusion errors in the Reading Span Test. *Mem. Cogn.* 37, 3 (2009), 336–345.
[134] Tobias Roehm, Rebecca Tiarks, Rainer Koschke, and Walid Maalej. 2012. How do professional developers comprehend software? In *34th International Conference on Software Engineering (ICSE)*. IEEE, 255–265.
[135] Mazeiar Salehie and Ladan Tahvildari. 2009. Self-adaptive software: Landscape and research challenges. *ACM Trans. Auton. Adapt. Syst.* 4, 2 (2009), 14.
University of Oulu. Retrieval https://core.ac.uk/download/pdf/344910619.pdf.
[136] Tommi Sallinen. 2020. Secure Coding Intention via Protection Motivation Theory Based Survey. University of Oulu. Retrieval https://core.ac.uk/download/pdf/344910619.pdf.
[137] Jerome H. Saltzer and Michael D. Schroeder. 1975. The protection of information in computer systems. *Proc. IEEE* 63, 9 (1975), 1278–1308.
[138] Luciano Sampaio and Alessandro Garcia. 2016. Exploring context-sensitive data flow analysis for early vulnerability detection. *J. Syst. Softw.* 113 (2016), 337–361.
[139] M. Angela Sasse and Awais Rashid. 2019. Human Factors Knowledge Area, Software Security Knowledge Area Issue 1.0. (2019). Retrieval https://www.cybok.org/media/downloads/Human_Factors_issue_1.0.pdf.
[140] Johann M. Schumann. 2001. *Automated Theorem Proving in Software Engineering*. Springer Science & Business Media.
[141] Anuj K. Shah and Daniel M. Oppenheimer. 2008. Heuristics made easy: An effort-reduction framework. *Psychol. Bull.* 134, 2 (2008), 207.
[142] Paschal Sheeran and Thomas L. Webb. 2016. The intention–behavior gap. *Soc. Personal. Psychol. Compass* 10, 9 (2016), 503–518.
[143] Michael Siegrist and George Cvetkovich. 2000. Perception of hazards: The role of social trust and knowledge. *Risk Anal.* 20, 5 (2000), 713–720.
[144] Alberto Sillitti, Giancarlo Succi, and Jelena Vlasenko. 2012. Understanding the impact of pair programming on developers' attention: A case study on a large industrial experimentation. In *34th International Conference on Software Engineering*, Martin Glinz, Gail C. Murphy, and Mauro Pezzè (Eds.). IEEE Computer Society, 1094–1101. DOI: https://doi.org/10.1109/ICSE.2012.6227110
[145] Daniel J. Simons and Christopher F. Chabris. 1999. Gorillas in our midst: Sustained inattentional blindness for dynamic events. *Perception* 28, 9 (1999), 1059–1074.
[146] Eliot R. Smith and Gün R. Semin. 2004. Socially situated cognition: Cognition in its social context. Retrieval https://psycnet.apa.org/record/2005-01913-002.
[147] Justin Smith, Brittany Johnson, Emerson Murphy-Hill, Bill Chu, and Heather Richter Lipford. 2015. Questions developers ask while diagnosing potential security vulnerabilities with static analysis. In *10th Joint Meeting on Foundations of Software Engineering*. ACM, 248–259.
[148] Justin Smith, Brittany Johnson, Emerson Murphy-Hill, Bei-Tseng Chu, and Heather Richter. 2018. How developers diagnose potential security vulnerabilities with a static analysis tool. *IEEE Trans. Softw. Eng.* 45, 9 (2018), 877–897.
[149] Joanne R. Smith and Winnifred R. Louis. 2008. Do as we say and as we do: The interplay of descriptive and injunctive group norms in the attitude–behaviour relationship. *British J. Soc. Psychol.* 47, 4 (2008), 647–666.
[150] Joanne R. Smith and Winnifred R. Louis. 2009. Group norms and the attitude–behaviour relationship. *Soc. Personal. Psychol. Compass* 3, 1 (2009), 19–35.
[151] Erin Treacy Solovey, Francine Lalooses, Krysta Chauncey, Douglas Weaver, Margarita Parasi, Matthias Scheutz, Angelo Sassaroli, Sergio Fantini, Paul Schermerhorn, Audrey Girouard, et al. 2011. Sensing cognitive multitasking for a brain-based adaptive user interface. In *SIGCHI Conference on Human Factors in Computing Systems*. ACM, 383–392.
[152] Source Code Analysis Tools - OWASP. n.d. Retrieved January, 2020 from https://www.owasp.org/index.php/Source_Code_Analysis_Tools.
[153] Mohammad Tahaei and Kami Vaniea. 2019. A survey on developer-centred security. In *IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE, 129–138.
[154] Carmen Tanner. 1999. Constraints on environmental behaviour. *J. Environ. Psychol.* 19, 2 (1999), 145–157.

[155] Blair Taylor and Shiva Azadegan. 2008. Moving beyond security tracks: Integrating security in cs0 and cs1. In *ACM SIGCSE Bulletin*, Vol. 40. ACM, 320–324.
[156] David R. Thomas. 2006. A general inductive approach for analyzing qualitative evaluation data. *Amer. J. Eval.* 27, 2 (2006), 237–246.
[157] Tyler W. Thomas, Madiha Tabassum, Bill Chu, and Heather Lipford. 2018. Security during application development: An application security expert perspective. In *CHI Conference on Human Factors in Computing Systems*. ACM, 262.
[158] J. N. Towse, M. Levine, M. Petre, A. Bandara, T. Lopez, A. Rashid, I. Rauf, H. Sharp, T. Tun, D. van der Linden, and B. Nuseibeh. 2020. The case for understanding secure coding as a psychological enterprise. (2020). Manuscript submitted for publication.
[159] Endel Tulving. 1993. What is episodic memory? *Curr. Direct. Psychol. Sci.* 2, 3 (1993), 67–70.
[160] Thein Than Tun, Mu Yang, Arosha K. Bandara, Yijun Yu, Armstrong Nhlabatsi, Niamul Khan, Khaled M. Khan, and Bashar Nuseibeh. 2018. Requirements and specifications for adaptive security: concepts and analysis. In *IEEE/ACM 13th International Symposium on Software Engineering for Adaptive and Self-managing Systems (SEAMS)*. IEEE, 161–171.
[161] John C. Turner, Michael A. Hogg, Penelope J. Oakes, Stephen D. Reicher, and Margaret S. Wetherell. 1987. *Rediscovering the Social Group: A Self-categorization Theory*. Basil Blackwell.
[162] John C. Turner, Penelope J. Oakes, S. Alexander Haslam, and Craig McGarty. 1994. Self and collective: Cognition and social context. *Personal. Soc. Psychol. Bull.etin* 20, 5 (1994), 454–463.
[163] Jay J. Van Bavel and Andrea Pereira. 2018. The partisan brain: An identity-based model of political belief. *Trends Cogn. Sci.* 22, 3 (2018), 213–224.
[164] Dirk van der Linden, Pauline Anthonysamy, Bashar Nuseibeh, Thein T. Tun, Marian Petre, Mark Levine, John Towse, and Awais Rashid. 2020. Schrödinger's security: Opening the box on app developers' security rationale. In *42nd International Conference on Software Engineering (ICSE)*.
[165] Dirk van der Linden, Emma Williams, Joseph Hallett, and Awais Rashid. 2020. The impact of surface features on choice of (in) secure answers by Stackoverflow readers. *IEEE Trans. Softw. Eng.* 1, 1 (2020), 1–1. DOI: 10.1109/TSE.2020.2981317
[166] Axel Van Lamsweerde. 2004. Elaborating security requirements by construction of intentional anti-models. In *26th International Conference on Software Engineering*. IEEE, 148–157.
[167] Axel Van Lamsweerde and Emmanuel Letier. 1998. Integrating obstacles in goal-driven requirements engineering. In *20th International Conference on Software Engineering*. IEEE, 53–62.
[168] Dirk van Moorselaar and Heleen A. Slagter. 2020. Inhibition in selective attention. *Ann. New York Acad. Sci.* 1464, 1 (2020), 204.
[169] Samuel M. Waldron, John Patrick, Phillip L. Morgan, and Sophia King. 2007. Influencing cognitive strategy by manipulating information access. *Comput. J.* 50, 6 (2007), 694–702.
[170] Charles Weir, Ingolf Becker, James Noble, Lynne Blair, M. Angela Sasse, and Awais Rashid. 2020. Interventions for software security: Creating a lightweight program of assurance techniques for developers. *Softw.: Pract. Exper.* 50, 3 (2020), 275–298.
[171] Charles Weir, Lynne Blair, Ingolf Becker, James Noble, Angela Sasse, and Awais Rashid. 2019. Interventions for Software security: Creating a lightweight program of assurance techniques for developers. In *41st International Conference on Software Engineering*, Helen Sharpe and Michael Whalen (Eds.). IEEE.
[172] Charles Weir, Awais Rashid, and James Noble. 2016. How to Improve the Security Skills of Mobile App Developers: Comparing and Contrasting Expert Views. In *2nd Workshop on Security Information Workers, WSIW@SOUPS 2016, Denver, CO, USA, June 22, 2016*. https://www.usenix.org/conference/soups2016/workshop-program/wsiw16/presentation/weir.
[173] Charles Weir, Awais Rashid, and James Noble. 2017. I'd like to have an argument, please: Using dialectic for effective app security In *EuroUSEC 2017 Internet Society*. Retrieval https://research-information.bris.ac.uk/en/publications/id-like-to-have-an-argument-please-using-dialectic-for-effective.
[174] Rodrigo Werlinger, Kirstie Hawkey, David Botta, and Konstantin Beznosov. 2009. Security practitioners in context: Their activities and interactions with other stakeholders within organizations. *Int. J. Hum.-comput. Stud.* 67, 7 (2009), 584–606.
[175] Michael Whitney, Heather Lipford-Richter, Bill Chu, and Jun Zhu. 2015. Embedding secure coding instruction into the IDE: A field study in an advanced CS course. In *46th ACM Technical Symposium on Computer Science Education*. 60–65.
[176] James A. Whittaker and Richard Ford. 2006. How to think about security. *IEEE Secur. Priv.* 4, 2 (2006), 68–71.
[177] Craig Williams, Helen M. Hodgetts, Candice Morey, Bill Macken, Dylan M. Jones, Qiyuan Zhang, and Phillip L. Morgan. 2020. Human error in information security: Exploring the role of interruptions and multitasking in action slips. In *International Conference on Human-computer Interaction*. Springer, 622–629.

[178] Jim Witschey, Shundan Xiao, and Emerson Murphy-Hill. 2014. Technical and personal factors influencing developers' adoption of security tools. In *ACM Workshop on Security Information Workers*. ACM, 23–26.
[179] Jim Witschey, Olga Zielinska, Allaire Welk, Emerson Murphy-Hill, Chris Mayhorn, and Thomas Zimmermann. 2015. Quantifying developers' adoption of security tools. In *10th Joint Meeting on Foundations of Software Engineering*. ACM, 260–271.
[180] Claes Wohlin. 2014. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *18th International Conference on Evaluation and Assessment in Software Engineering*. 1–10.
[181] Claes Wohlin and Rafael Prikladniki. 2013. Systematic literature reviews in software engineering. *Inf. Softw. Technol.* 55, 6 (2013), 919–920.
[182] Irene M. Y. Woon and Atreyi Kankanhalli. 2007. Investigation of IS professionals' intention to practise secure development of applications. *Int. J. Hum.-comput. Stud.* 65, 1 (2007), 29–41.
[183] Shundan Xiao, Jim Witschey, and Emerson Murphy-Hill. 2014. Social influences on secure development tool adoption: Why security tools spread. In *17th ACM Conference on Computer Supported Cooperative Work & Social Computing*. ACM, 1095–1106.
[184] Jing Xie, Bill Chu, Heather Richter Lipford, and John T. Melton. 2011. ASIDE: IDE support for web application security. In *27th Annual Computer Security Applications Conference*. ACM, 267–276.
[185] Jing Xie, Heather Richter Lipford, and Bill Chu. 2011. Why do programmers make security errors? In *IEEE Symposium on Visual Languages and Human-centric Computing (VL/HCC)*. IEEE, 161–164.
[186] Limin Yang, Xiangxue Li, and Yu Yu. 2017. VulDigger: A just-in-time and cost-aware tool for digging vulnerability-contributing changes. In *IEEE Global Communications Conference*. IEEE, 1–7.
[187] Xin-Li Yang, David Lo, Xin Xia, Zhi-Yuan Wan, and Jian-Ling Sun. 2016. What security questions do developers ask? A large-scale study of stack overflow posts. *J. Comput. Sci. Technol.* 31, 5 (2016), 910–924.
[188] Thomas Zimmermann, Nachiappan Nagappan, and Laurie Williams. 2010. Searching for a needle in a haystack: Predicting security vulnerabilities for Windows Vista. In *3rd International Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 421–428.

Received September 2020; revised April 2021; accepted June 2021