JOURNAL OF CYBERSECURITY

Journal of Cybersecurity, 2025, tyaf005
https://doi.org/10.1093/cybsec/tyaf005

Research Paper

Software security in practice: knowledge and motivation

Hala Assal [1,*], Srivathsan G. Morkonda [2], Muhammad Zaid Arif [2], Sonia Chiasson [2]

[1]Department of Systems and Computer Engineering, 1125 Colonel By Drive, Ottawa, ON, K1S 5B6, Canada.
[2]School of Computer Science, Carleton University, 1125 Colonel By Drive, Ottawa, ON, K1S 5B6, Canada

*Corresponding author. Department of Systems and Computer Engineering, Carleton University, 1125 Colonel By Drive, Ottawa, ON, K1S 5B6, Canada. E-mail: Hala.Assal@carleton.ca

Received 7 October 2024; revised 12 December 2024; accepted 14 February 2025

Abstract

Developing secure software remains a challenge for developers despite the availability of security resources and secure development tools. Common factors affecting software security include the developer's security awareness and the rationales behind their development decisions with respect to security. In this work, we conducted interviews with software developers to examine how developers in organizations acquire security knowledge, and what factors motivate or prevent developers from adopting software security practices. Our analysis reveals that developers' security knowledge and motivations are intertwined aspects that are both important for promoting security in development teams. We identified a variety of learning opportunities used by developers and employers for increasing security awareness, including in-context learning activities preferred by developers. Based on our application of the self-determination theory, better security outcomes are expected when developers are internally driven toward security, rather than motivated by external factors; this aligns with our interpretation of participants' descriptions relating to security outcomes within their teams. Based on our analysis, we provide ideas on how to motivate developers to internalize security and improve their security practices.

Keywords: usable security; software security; software developers; interview; security knowledge; security motivation

Introduction

Software developers are not necessarily security experts, yet they are widely held responsible for developing secure applications. Many security initiatives and tools have been proposed to support the integration of security in the Software Development Lifecycle (SDLC) (e.g.[1–7]). Despite these efforts, vulnerabilities persist [8,9] and with the proliferation of software in all aspects of our lives, security vulnerabilities can have a devastating impact on users' livelihood (e.g. vulnerabilities in cars [10,11], in medical wearable devices [12], or in home appliances [13]). Formally, a software vulnerability can be defined as "*a weakness in an information system, system security procedure, internal control, or implementation that could be exploited by a threat source.*" [14]. Vulnerabilities could be unintentional or could be introduced to a system by a malicious developer; the latter is out of the scope of this paper.

Many reasons have been suggested for the prevalence of vulnerabilities. For example, a common problem for security is the *the unmotivated user property* [15], where security is generally not a primary goal for users; this concept also applies to software developers since security is rarely their primary objective [16–19]. While secure coding guidelines could be useful, developers are generally unaware of such guidelines [5], or are not mandated by their companies to use them [20–22]. Besides, developers might lack security knowledge necessary to prevent vulnerabilities [23–25]. And even when they do possess some security knowledge, developers may lack the ability [26] or expertise [25,27] to apply this knowledge to identify and address vulnerabilities in their applications. In this context, *security knowledge* refers to information that increases developers' software security awareness, and helps them avoid, identify, or fix security issues. Previous usable security research has focused on developers and the human factors of software security [16–19]. For example, Acar et al. [16] developed a research agenda that focuses on proposing and improving security tools and methodologies, as well as understanding how developers view and deal with software security.

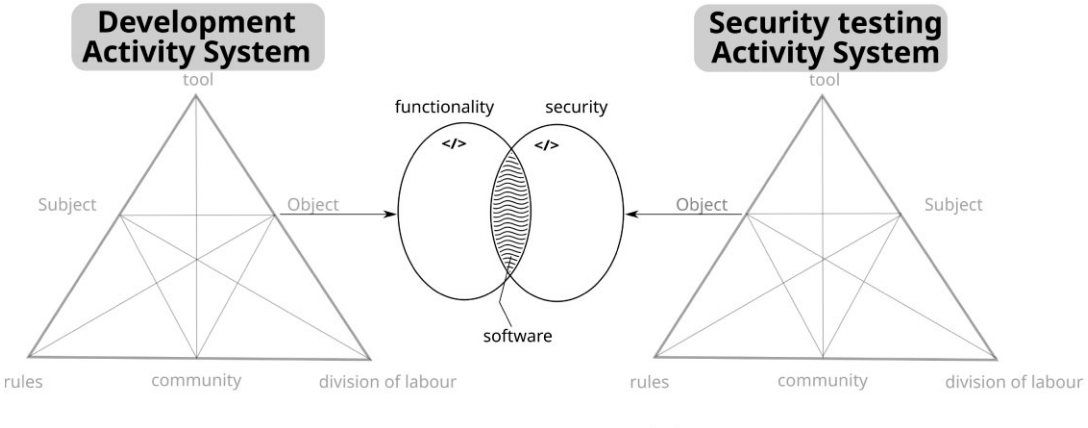


Figure 1. Third generation activity theory with two interacting activity systems [35].

We conducted an interview study with professional software developers. In an accompanying paper [28], we analyzed a portion of this data to explore real-life software security practices and identified factors that may influence these practices. In this current paper, we address different research questions and analyze the full dataset using the Grounded Theory methodology. Herein, we focus on how developers learn about security, and factors that motivate (or deter) developers from addressing software security. We presented early discussion about the motivations and amotivations related to software security at the 2018 SOUPS Workshop on Security Information Workers [29]). In this paper, we pursue the following research questions:

RQ1: How do developers acquire knowledge relating to software security?
RQ2: What are developers' motivations towards adopting software security practices?

Initially, we set out to explore developers' knowledge of software security and how they acquire this knowledge (*RQ*1). However, during preliminary data analysis, we found that even those with the necessary security knowledge may lack motivation to adopt software security practices. This led us to explore the second research question (*RQ*2).

Through this work, we identify opportunities and strategies for acquiring security knowledge (Section "Knowledge acquisition taxonomy"), and explore factors influencing developers' motivation toward adopting security practices (Section "Motivation for software security"). Our data analysis also revealed that security knowledge and motivation are two intertwined aspects that may influence security practices; motivation in itself is not enough if the developer lacks security knowledge and, as it turns out, security knowledge itself affects motivation. In addition, we found several factors that may induce developers' amotivation (i.e. lack of drive to engage) toward security, despite their security knowledge and belief of its importance. Thus, besides offering technical support for developers with security tools and libraries, our data shows the importance of having developers internalize software security practices and act with volition toward it (Section "Internalizing software security").

Related work and background

Developers may recognise the need for integrating security within the SDLC [30], yet they may be unable to follow security practices due to insufficient security knowledge or due to workplace factors such as a lack of security culture [28,31]. In this paper, we focus on how software developers gain security knowledge, as well as their motivations toward adopting software security practices.

We begin this section by providing a brief background on two theories that we use to explain the results (in sections "Knowledge acquisition taxonomy" and "Motivation for software security"), followed by a review of existing literature on factors affecting secure software development practices.

Theoretical background on activity theory

In the section "Multiple activity systems interacting within project teams," we use Activity Theory [32,33] to describe the interactions between the different teams working on developing a software product, their often conflicting objectives and perspectives, and how these multiple perspectives can enhance/impede the security of their software. Here, we provide a brief primer on Activity Theory.

Activity theory [32,33] can be defined as "*a philosophical and cross-disciplinary framework for studying different forms of human practices as development processes, with both individual and social levels interlinked at the same time*" [34]. Engeström proposed the "activity system model" [33,34] that describes a three-way relationship between a subject (e.g. a developer), their object (e.g. developing software) and their community (e.g. a development team).

The "third generation of activity theory" expands the original theory by considering two activity systems as the minimal unit of analysis [35]. It aims to understand discussions, perspectives, and interactions between multiple activity systems. Of particular relevance to this paper is the principle of *multi-voicedness* [35]. Rather than a homogeneous system, activity theory views an activity system as consisting of multiple perspectives, traditions, and interests [35,36]. This multi-voicedness is magnified when multiple activity systems interact as it brings multiple viewpoints closer.

As will be discussed in the section "Multiple activity systems interacting within project teams," in software development organizations, the development activity can be considered as one activity system while the security testing activity is a second independent activity system (Fig. 1). Each activity system involves different objectives, priorities (e.g. functionality for developers, and security for security testers), and perspectives.

Theoretical background on Self-Determination Theory

In the section "Motivation for software security," we use the Self-Determination Theory (SDT) to explain what motivates developers

and their teams to address software security, as well as explain reasons for lacking software security motivation.

SDT [37,38] is a cognitive framework for studying human motivation in learning environments such as classrooms and organizations. SDT identified distinct types of motivation, each with clear consequences on an individual's potential to thrive [39] in relation to learning, performance, personal experience, or well-being [38]. Behaviors are "autonomously" motivated when they are fully self-determined, whereas "controlled" behaviors are those driven by external or internal pressures or an obligation to act [38,39]; in our context, a developer who performs security activities out of self-interest characterizes fully autonomous behavior, whereas a developer who performs security activities only to comply with regulations represents controlled behavior. SDT uses the *autonomy-control continuum* to differentiate types of motivation with respect to their regulation [39], as described next (and later depicted in the section "Motivation for software security," Fig. 4).

Amotivation (at the far left of this continuum) is the lack of motivation to act, where the person does not act at all or acts without intent [38]. Amotivation has three forms. The first form is when people feel they cannot effectively achieve the desired outcome, e.g. because they are not competent to do it [38–40], such as when they lack security knowledge. The second form of amotivation occurs when the action lacks interest, relevance, or value to the person [38–40], such as when they do not perceive security to be their responsibility. Finally, the third form is when amotivation to a behavior is actually a defiance and motivation to oppose said behavior [39]. For example, a developer refuses to use a security tool because it requires modifying their existing workflows.

To the right of amotivation, the continuum presents the different types of motivations based on the actor's degree of autonomy when carrying out the activity; we describe these motivations in the context of software security practices. *External regulation* is the least autonomous and most external to the developer (e.g. motivated only by fear of losing business opportunities), whereas *introjected regulation* refers to motivations that result from self-pressure and ego. More autonomously, *identified regulation* occurs when the developer deems security as personally important. *Integrated regulation* is when the developer fully accepts the goal of the activity and acts toward it with volition (e.g. to protect users' privacy and security). On the rightmost end of the spectrum, the most autonomous is *intrinsic motivation*, where security activities are performed purely for the pleasure and satisfaction that result from the challenge they present to the developer. SDT suggests that more autonomous motivation styles are associated with positive outcomes, such as increased engagement, improved performance, encouraging creativity, more cognitive flexibility, and better learning [38,40].

Secure software development practices

Security within the SDLC

Security best practices recommend integrating security throughout the SDLC and starting from the early stages [28,41,42]. Assal and Chiasson [28] discuss practical strategies for integrating security in each stage of the SDLC, e.g. by identifying security requirements and performing threat modeling in the design stage, and integrating security in the post-development testing stage. The authors also point out the importance of considering security of third-party libraries and monitoring for vulnerabilities [28]. When security is a priority, it should influence decisions throughout the SDLC. For example, developers could consider more secure programming languages such as Rust that prevent certain classes of vulnerabilities [43]. However, previous work suggests that security tends to be considered only in certain development stages rather than integrated throughout the SDLC. Assal and Chiasson [28] found that development teams often prioritize security only in the latter stages of the SDLC such as during code reviews and post-development testing. They also found that deviation from security best practices (e.g. the principle of security-by-design) are influenced by workplace factors such as the organization's development hierarchy, and developers' security knowledge. Braz and Bacchelli [31] examined code review practices in specific organizations, and found that security considerations were largely absent during code reviews. They suggested that developers might be lacking motivation to consider security during code reviews because developers had insufficient security knowledge or because of insufficient employer-driven security training. Many studies about secure software development (e.g. [44–46]) have found that developers need to be explicitly reminded to consider security. For example, Braz et al. [44] suggested that simply prompting code reviewers to focus on security can increase detection of vulnerabilities. Interestingly, developers may also inadvertently (and possibly unknowingly) follow secure practices as an unintentional by-product of completing other tasks [47] but this haphazard approach is clearly unreliable. In summary, existing research has focused on how security practices are included in the SDLC; our work extends the literature by exploring what motivates developers with respect to software security, and identifies activities that can help improve developers' security knowledge.

Factors influencing adoption of secure development solutions

Other research has focused on understanding factors influencing the adoption of available tools that support secure software development. Besides developers' security expertise and skills [28,48], adoption tends to be impacted by the organization's overall culture toward security and by the context of the application being developed [21,22,49–51]. For example, developers (even those with security knowledge) may lack resources or the autonomy within their organization to improve their applications' security and privacy [50,52]. Relatedly, Danilova et al. [53] found that when security tools (e.g. static analysis tools) are incompatible with developers' preferences, workflows, or their roles within the organizations, the developers perceive the security warnings from these tools as unuseful. However, developers tend to be more receptive when the security warnings include code examples for vulnerabilities and solutions [54]. Similarly, Acar et al. [55] suggested that including working code samples and comprehensive Application Programming Interface (API) documentation would improve the adoption of cryptographic APIs and lead to more secure code. Fulton et al. [43] found that although developers perceived positive security benefits from the memory-safe programming language Rust, they were reluctant to adopt it due to its steep learning curve. These findings suggest that security adoption depends not only on developers' security knowledge or their willingness toward security but also on organizational norms and usability of security tools. Thus, it is important to consider these diverse factors when encouraging security adoption.

Security knowledge and skills

Sources of security knowledge

Previous research has explored the impact of specific knowledge sources used by developers for learning about software security including online developer forums and code documentation. Having security knowledge is often a key driver of security-related activities in development teams [56,57], but the quality of the information source can significantly impact whether developers succeed at

producing secure code. Developers tend to depend heavily on the online Question and Answer developer-focused forum Stack Overflow [58] as an easy-to-use programming resource, however, it often leads to insecure code [59,60]. A comparative study by Acar et al. [60] on Android application development found that relying on official Android documentation and textbooks led developers to produce more secure code while developers relying on Stack Overflow produced more functional but less secure code. Sadly, many security-related posts on Stack Overflow tend to remain unanswered or have answers that are not "accepted" (i.e. not marked as satisfactory by the user asking the question) [61], raising doubts about the efficacy of such forums for software security. Developers also rely on colleagues as sources of security knowledge. Through an ethnographic study, Tuladhar et al. [57] observed that developers acquire security knowledge when collaborating with other developers on the specific applications they develop. These studies show that developers rely on the knowledge resources that are available to them which may be of varying quality and may not necessarily align with security best practices. In the section "Knowledge acquisition taxonomy," we discuss security knowledge acquisition activities in detail.

Security awareness among developers

Other works have focused on understanding and improving developers' security awareness. Gasiba et al. [5] found that developers are generally not aware of secure coding guidelines, which aim to promote security knowledge. Votipka et al. [25] found that the lack of security knowledge was a primary contributor to the introduction of vulnerabilities regardless of development experience. In addition, even when developers are aware of security concepts, they may not know how to apply the concepts correctly in order to achieve the intended security goals [25]. Lopez et al. [30] suggested a different view that developers do possess basic security knowledge but they lack control of development activities that provide opportunities to extend their security knowledge. They recommended that developers combine online discussions about security on sites such as Stack Overflow with social interactions about security in their development teams to extend security knowledge and promote a security culture within the organization. There also exist studies that focused on improving developers' programming knowledge in general (i.e. not specific to security), for example, using "in-context" code annotations [62] or using dedicated knowledge-sharing platforms [63]; similar approaches could potentially also help improve developers' security knowledge but this has not yet been explored. Beyond those discussed above, our work identifies additional security knowledge sources and development activities that help increase developers' security knowledge, and proposes an overall taxonomy allowing for comparison.

Developer challenges and motivators

Challenges faced by developers

Building secure software is challenging, especially when developers without adequate security knowledge are expected to implement code that might affect the security of the software [17,60]. Mokhberi and Beznosov [19] reviewed previous studies focused on the challenges faced by developers in secure software development and suggested that organizational and human factors tend to have a larger impact than technological factors, e.g. developers favor security tool adoption when supported by a manager who prioritizes security and by an organization that provides access to security resources. Tahaei and Vaniea [18] surveyed existing literature on "developer-centered security" (a secure software development approach with emphasis on the needs of developers [64]) and discussed the importance of workplace context and organizational incentives when considering developers' security needs. A common challenge highlighted by both surveys relates to security often being considered a secondary function which causes developers to sacrifice security for other functional requirements [18,19]. Poller et al. [65] found that developers may be reluctant to change their security practices due to various organizational factors such as existing development procedures and assigned roles. These studies show that barriers to secure development practices are much more nuanced than simply a lack of usable security tools and that several important socio-organizational factors also need to be considered.

Understanding developer motivations

In an early study on security motivations within organizations, Woon and Kankanhalli [66] found that developers' intentions to perform security actions were aligned with the developers having positive attitudes and perceptions of the usefulness of security. More recent work on identifying developers' security motivations have focused on independent app developers. For example, Van der Linden et al. [47] examined the relationships between mobile app developers' security rationale and their security behavior, and observed that app developers were particularly concerned about security in specific scenarios such as when collecting personal data of users or when storing password hashes. Weir et al. [67] found that Android app developers tend to perform security-enhancing activities when they perceive a need for security and when they also have access to security experts. Ryan et al. [51] characterized developers into four security archetypes that explain developers' security behaviors. The authors suggested that developers' security behavior is primarily influenced by self-interest and environmental support. Through our interviews and data analysis herein, we found additional factors that fit into a wide spectrum of motivators and amotivators described in the section "Motivation for software security." While there has been a considerable focus on the challenges faced by independent developers, there has been little focus on understanding what motivates or demotivates developers with respect to security within organizational contexts. Our study begins to fill this research gap.

Interventions for motivating developers

Other work has proposed interventions to motivate developers toward adopting security practices, including on-the-job training activities [5,6,68], cybersecurity games [69,70] and in-context education [26,68]. Weir et al. [6] demonstrated that a series of low-cost facilitated workshops can motivate developers to adopt effective security practices. Relatedly, Lopez et al. [69] designed guidelines for engaging with developers about security; they recommended using scenario-based games and workshops to motivate discussions about security. Gasiba et al. [5] suggested various training activities within development teams in order to raise awareness about secure coding guidelines. They also designed a cybersecurity game [70] targeted to developers in industrial environments. To support developers in the correct use of cryptographic APIs, Gorski et al. [71] proposed designing security warnings using concise messages and including code examples for different use cases. Thomas et al. [68] recommended tailoring training activities to focus on the specific types of security issues developers are likely to encounter in their code, and to address developers' weakness in security knowledge. Ford et al. [72] recommended matching development tasks based on developer personalities, which may also be useful for security-related tasks. The variety of interventions considered in these studies suggest that there is no one-size-fits-all solution. In this paper, we take a more holistic view by describing a number of activities organizations can adopt

Table 1. Participant demographics.

	Participant					Company and team	
ID	Gender	Age	Years	Title	SK	Company size	Team size
P1	F	30	1	Software engineer	4	Large enterprise	20
P2	M	34	15	Software engineer	5	Large enterprise	12
P3	M	33	10	Software engineer	4	Large enterprise	10
P4	M	38	21	Software developer	4	SME	7
P5	M	34	12	Product manager	5	Large enterprise	7
P6	F	26	3	Software engineering analyst	3	Large enterprise	12
P7	M	33	4	Senior web engineer	4	SME	3
P8	M	34	5	Software developer	3	Large enterprise	20
P9	M	33	8	Software engineer	2	SME	5
P10	M	37	20	Principal software engineer	5	SME	10
P11	M	38	15	Senior software developer	2	SME	8
P12	M	26	3	Software developer	2	SME	4
P13	F	27	5	Junior software developer	4	Large enterprise	7

Years: years of experience in development.
SK: self-rating of security knowledge 1 (no knowledge) to 5 (expert). SME: Small–medium enterprise.

based on their needs and constraints to promote security-enhancing activities.

Study design and methodology

In this section, we present the study design, data analysis methodology, participant demographics, and study limitations.

Interview study design

We designed and conducted an IRB-approved semi-structured interview study with professional software developers. The interview addressed five main topics: general development activities, attitude toward security, security knowledge, security processes, and software testing activities (see Appendix A for the interview script). We recruited participants through posts on software development forums and relevant social media groups, as well as through announcements among professional contacts. Participants received a $20 Amazon gift card as compensation. Participants completed a demographics questionnaire before their one-on-one interviews. Interviews were conducted in-person ($n = 3$) or remotely, e.g. through phone calls or videoconferencing ($n = 10$), lasted approximately one hour, and were audio recorded and later transcribed for analysis. A total of 3 waves of data collection took place, each followed by preliminary analysis and preliminary conclusions [73]. We concluded recruitment upon saturation (i.e. when new data did not provide more insights) as per Glaser and Strauss's [73] recommendation. In total, we recruited 13 professional software developers for our study.

Participant demographics

Our 13 participants answered our interview questions in the context of 15 companies; two participants reflected on their current and previous companies. We did not have multiple participants from the same company. Participants self-identified their roles and the products with which they are involved. We then classified their responses using Forward and Lethbridge's [74] software taxonomy. Participants in our dataset worked on various types of applications: web applications and services like e-finance, online productivity, online booking, website content management, and social networking, as well as software like embedded software, kernels, design and engineering software, support utilities, and information management and support systems. The organizations mentioned in the dataset were all based in North America. All participants included in the study hold university degrees, have had courses in software programming, and are employed as developers with an average of 9.35 years experience ($Md = 8$). The recruitment did not prioritize any specific software development methodology. Participants reported following a Waterfall model or variations of the Agile methodology. See Table 1 for participant demographics.

Data analysis approach

We used Strauss and Corbin's Grounded Theory methodology [75] to analyze our interviews. Our analysis took into consideration the different stages of the SDLC, and how security was (or was not) included in each stage. For example, during our analysis of participants' code review processes, we paid particular attention to whether/how security was addressed during these reviews. This included, analyzing whether security was prioritized during the reviews or if it was addressed in an ad-hoc manner, the reviewers' security expertise, how security considerations were raised and addressed, and how receptive different team members were to discussing and addressing .0s. We performed open-coding through examining the answer to each question in the interview script and assigning codes describing the main themes or ideas discussed. The main researcher performed the open-coding, however, codes were discussed with a second researcher whenever a new code was created. Open coding was done using Atlas.ti (https://atlasti.com/interview-analysis-tools) on 600 unique excerpts and resulted in a total of 170 open codes. We italicize and use a different `font` for our codes when reporting.

For example, "`Learning from peers`" is an open code that we created when participants indicated that they acquire security knowledge through interaction with their colleagues. In the following quote, P8 explains how all his security knowledge came from colleagues in the company where he works. He said, "*I guess up until now, any knowledge I have got of [software security] has just been purely from peers, or anytime we bring in a new employee and they have more knowledge about it. That's where I kinda learn it from.* "

Following open coding, we performed axial coding by looking for patterns, relationships, and connections between the open codes. We wrote each of the codes on a Post-It note, grouped similar ones, and looked for relationships, such as categorical or causality relationships. Even though this process was possible using Atlas.ti, we preferred using Post-It notes to allow us to be more im-

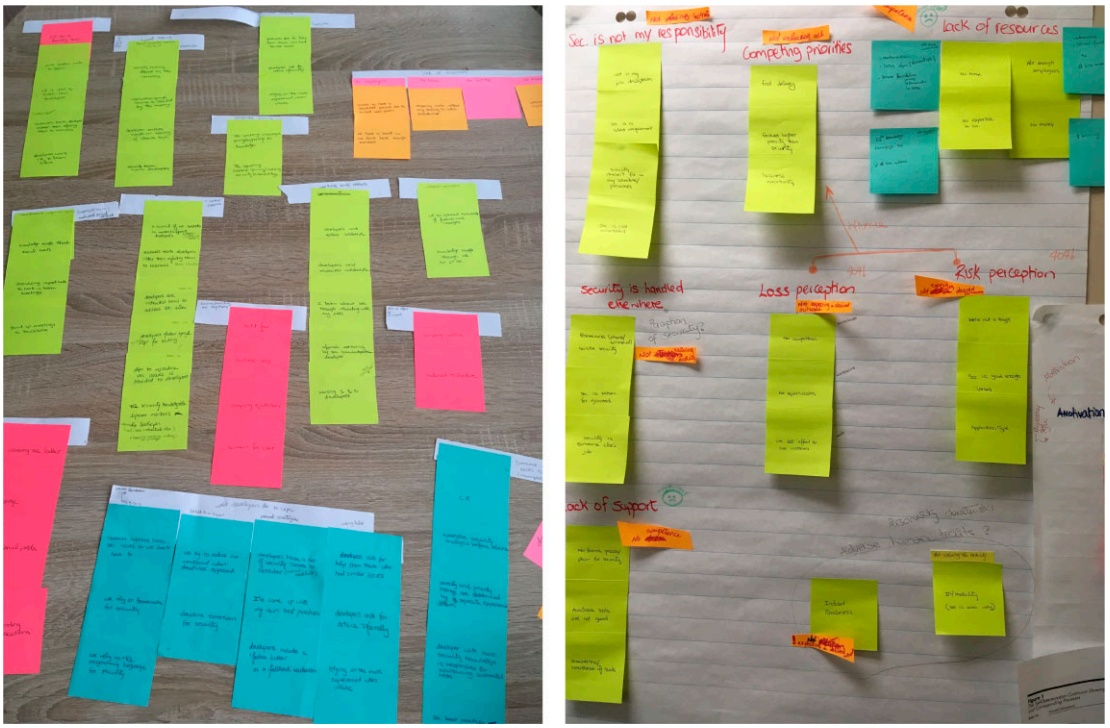


Figure 2. Axial coding process. Left: first round of axial coding, right: looking for relationships and connections.

mersed in the data, have an overview of the codes and categories, and have the ability to move them around as needed, as shown in Fig. 2.

The last step in coding was selective coding, where we worked toward integrating and refining the categories, and identifying a *core category* that represents the overall theme of the research [75]. To achieve this, we examined the categories, while referring back to the interview scripts (raw data), abstracting the main issue and asking ourselves: "*what comes through although it might not be said directly?*" [75].

Through being immersed in the data and as the analysis continued, a core category relating to the concept of internalizing and accepting security activities and behaviors began to emerge. We will discuss this further in the section "Internalizing software security."

Knowledge acquisition taxonomy

Through our analysis of the interview data, we identified different opportunities for acquiring and sharing security knowledge. We found that these opportunities can be grouped based on distinct characteristics (discussed below), which we organize into a taxonomy. In Table 2, we present the knowledge acquisition taxonomy: a taxonomy of the activities described by our participants that we have identified as opportunities for knowledge acquisition. We represent the type of learning associated with the activity horizontally across the table, and the initiator of the activity (i.e. the initiator of the learning opportunity) vertically.

Types of learning

We classified the learning opportunities identified in our data according to their level of formality. "Formal" learning is always organized and structured, has learning objectives, and is always intentional from the learner's perspective [76–79]. Conversely, "informal" learning is neither organized nor structured, does not have specific objectives, and happens as a by-product of other activities [76,77,79]. "Semi-formal" learning opportunities identified in our data fall between these two, where learning lacks one or more aspects of formal learning while being more organized and structured than informal learning [76,77,79].

Activity initiator

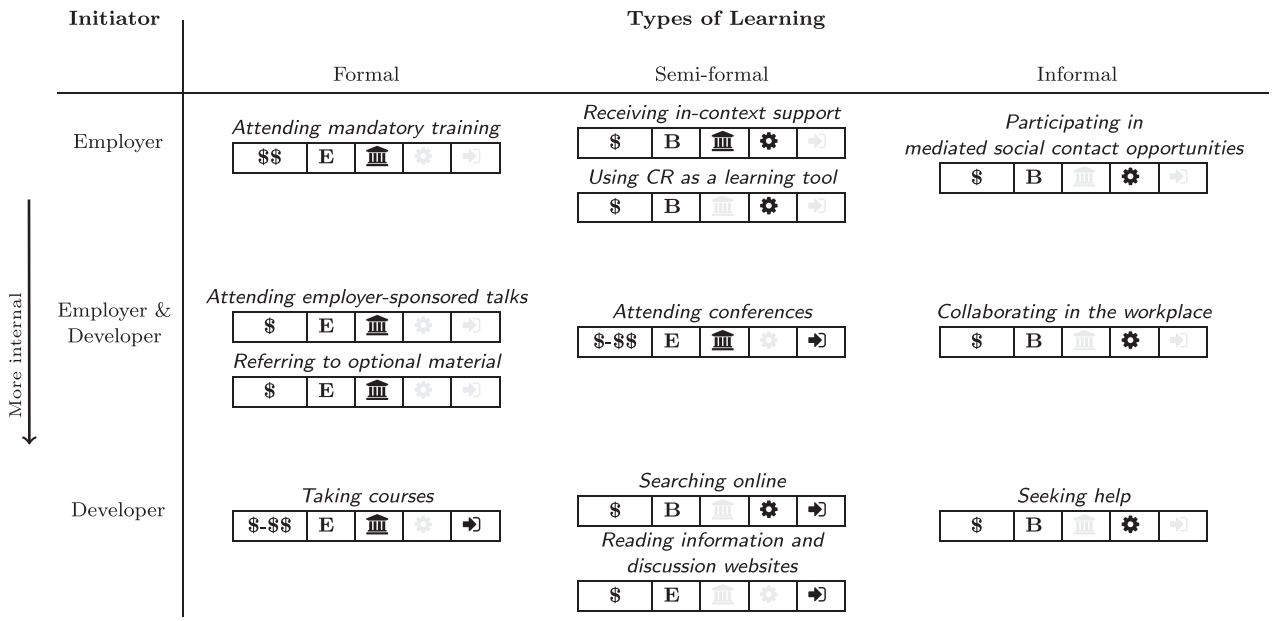
This is the entity with the motivation to start the activity, thus the security learning opportunity. This can be the employer (e.g. when the employer organizes mandatory activities that the developer attends for compliance), or the developers (i.e. in activities that the developer is self-motivated to initiate without direct encouragement or mandate). Some activities are initiated by *both* the employer and the developer, e.g. *optional* activities the employer sets up in which developers can choose to participate, even though they would not have initiated the activity on their own. Thus, activity initiation is along an *employer—developer* spectrum, where initiation tends to be more internal to, and self-motivated by, the developer as we move toward the developer end of the spectrum. We note that developers disinterested in security may still perform activities in the taxonomy's third row (i.e. developer-initiated activities) as part of completing job-related tasks; however, our analysis shows that they are likely to procrastinate with respect to these activities. We discuss developer motivations in more detail in the section "Motivation for software security."

Features

We have identified five different features for each learning opportunity/activity.

- *Relative cost (*$, $$*)*. The symbol $ indicates that the activity is relatively low cost and $$ indicates higher cost. Obviously, the cost one company finds reasonable may be expensive for another.

Table 2. Knowledge acquisition taxonomy. The taxonomy presents security knowledge acquisition opportunities and features associated with each opportunity. See inline for their description.



$** - lower cost; **$$ - higher cost.
E - learning is the developer's explicit objective; **B** - learning is a by-product of another activity.
🏛 - high subject-matter expertise (lower expertise when grayed out).
⚙ - activity is part of the SDLC (not part of the SDLC when grayed out).
➔ - knowledge source is external to the company (source is internal when grayed out).

Thus, this characteristic should be used to compare activities relative to each other, rather than, e.g. finding the most reasonably priced activity.

- *Fit in the developer's objective (***E**, **B***).* We use **E** to indicate that learning is the developer's explicit objective for the activity, whereas **B** indicates that learning is a by-product of another activity.
- *Source expertise (🏛).* A 🏛 indicates that the source of information has high subject-matter expertise. It is grayed-out (🏛) when the source of information varies in their level of expertise. Note that advancement in knowledge can occur through discussions and sharing of interpretations, even if the teacher does not have higher expertise than the learner [80].
- *Fit in the SDLC (⚙).* A ⚙ indicates that the activity is performed as part of the developer's tasks (thus part of the SDLC). It is grayed-out (⚙) when the activity is *not* part of the SDLC.
- *Knowledge source (➔).* A ➔ indicates that knowledge is flowing to the company from external sources. It is grayed-out (➔) when knowledge is shared *within* the company.

We built this taxonomy based on our analysis of the interview data. Though other activities may exist that are not included in the taxonomy, the taxonomy allows for exploring and reasoning about activities that induce learning in the context of software security. We next describe and provide context for each activity presented in the taxonomy. Table B1 in Appendix B shows which participants discussed each activity categorized in the taxonomy; it highlights that all opportunities were discussed by multiple individuals.

Formal learning

For all formal learning opportunities identified, learning is explicit (**E**) from the developer's perspective. In addition, the source of information, be it an instructor in a training session or an author of a training manual, is assumed to have high subject-matter expertise (🏛).

Attending mandatory training ($$E 🏛 ⚙ ➔)

This formal learning refers to one-time or regularly scheduled training activities that employers require developers to complete as part of their job. Formal security training is usually expected as the first step for a secure SDLC [7]. Most participants mentioned that they attended mandatory security training when they started their job but, in most cases, the training focused on general security topics (e.g. passwords and phishing), and on best practices while using company resources or sharing company code. Three participants reported attending mandatory on-boarding training that explicitly included aspects of software security. In addition to attendance, P1 mentioned her company requiring that developers successfully pass exams relating to the training topics.

Beyond initial mandatory training, P5's company also requires attendance at regularly scheduled training sessions. P5 explains, "*We kept doing [mandatory training], and it's been quite effective. So as a result of it being effective, we scaled down the frequency at which it needs to be done. But it's not because it's less important, it's just because people started to get it more.*" Linking training frequency to security outcomes could lead developers to find the training more useful and be more attentive to it. The rationale may be that when developers are more attentive, they can spend less time on additional

training and they can get go back to their development work more quickly. In the section "Internalizing software security," we discuss how valuing security can impact developers' performance and promote software security.

Attending employer-sponsored talks ($E 🏛 ⚙ 🡒)
Our data analysis shows that employer-sponsored talks are typically technical and have specific learning objectives (e.g. introducing a new security API). This type of learning opportunity is initiated by both the employer (for hosting the talk) and the developer (for deciding to attend). These talks are usually given by employees sharing expertise with colleagues, thus knowledge sources are internal to the company. P3 mentioned that his company sometimes invites external experts as well.

Referring to optional material ($E 🏛 ⚙ 🡒)
Some companies provide optional security knowledge resources (e.g. reading material, or video lectures) prepared by in-house experts; similar to the employer-sponsored talks above, these activities are initiated by both the employer (for offering the material) and the developer (for seeking the material). P3 explained, "*they do have an infrastructure, so that people can easily find actual courses taught by colleagues at this very institution. [...] They have a series of sort of self seminars, so these are like slideshows or online videos that we can look at.*" P1 mentioned that their material is accompanied by an assessment test that allows the developer to self-test their knowledge and the company to recognize the developer's level of security knowledge.

Taking courses ($-$$E 🏛 ⚙ 🡒)
Some participants explained that they took courses or even acquired a graduate degree to increase or maintain their security knowledge. Their companies did not require or even encourage them to do so, thus the initiation of this learning opportunity is internal to the developer. P11 explained, "*For me, I like taking courses [...] I mean [...] something [that] shows me all the types of vulnerabilities I need to really be thinking about when I am working on my applications.*" The cost of this activity varies; an online course is likely less expensive than an on-site course, and both are less expensive than a graduate degree. Knowledge here is flowing to the company from an external source (the institute offering the course or graduate degree).

Semi-formal learning

The semi-formal learning opportunities listed here vary in their degree of structure, the presence of learning goals, and the experience of the information source. Learning may be the developer's explicit objective (**E**) or a by-product (**B**) of development activities.

Receiving in-context support ($B 🏛 ⚙ 🡒)
Some participants mentioned learning about security through different forms of in-context support received while working on their tasks (⚙). Thus, learning from these activities is a by-product (**B**) of the development task. For example, participants discussed that if a security tester identifies an issue in their code, the tester would then provide the developer with the specific steps to follow to reproduce the issue, as well as an explanation of the issue and possible fixes. One participant also mentioned that his company pairs developers with more senior colleagues to disseminate security knowledge across the development team. P9 explained that the intention is "*to make sure that the junior [developer] doesn't feel, you know, like they are left alone on the issue or they are frustrated or stuck. [They] have somebody to kinda guide them through the work that they are doing step by step.*" On the other hand, P9's previous company took this even further and formed a "security council" from in-house experts to provide security guidance. The council keeps developers updated on relevant issues to consider during implementation, and developers are expected to consult this council, e.g. when they need advice. P9 explained, "*If there is anything that we flag up as 'ok this might have security implications', then it goes to them to say 'ok, do you guys find anything? [Do] you have any comments on the design? Is there anything maybe we didn't think of?'*"

Using CR as a learning tool ($B 🏛 ⚙ 🡒)
Code review is a typical SDLC step where implemented code is inspected by reviewers (possibly including the author of the code); these reviewers typically involve developers, but can also involve testers, security experts, and other project team members. During code review, the reviewers collectively aim to find or address code issues, which is typically primarily focused on functionality issues, but can also address security bugs [28]. Similar to the in-context learning activities described above, code reviews allow developers to gain security knowledge while working on their tasks (⚙), and security learning is a by-product (**B**) of the development task. P1 explained that upon finding a security issue, reviewers take this opportunity to teach the developer about its implications and how to fix it. This can happen face-to-face or through documented code review feedback. She said, "*[Reviewers] just come right away to your cubicle and explain [to] you [...] because they feel [that] going and taking a book and reading it would be, mmm, so much painful. So, they just come over to you and draw on the board and explain what you did and what you should not do.*" In some cases, the reviewer is not necessarily more experienced than the developer, however, the discussion that arises during the review session can lead to better insights on code security. P1 also mentioned that junior developers can act as mock reviewers to learn about the process and types of issues to avoid in their code. One of the factors to rate the success of a code review session discussed by participants is by determining whether the review resulted in information sharing among reviewers and developers. P5 explained, "*A good review is one where the development team gets a better understanding of the security of the application, and the security team gets a better understanding of how applications are constructed and how to interact with the development teams.*"

Attending conferences ($-$$E 🏛 ⚙ 🡒)
Some participants mentioned that they sometimes attend academic conferences to keep up with new technologies and new security attacks and defences. There is no mandate from their employers to attend such events, although it is encouraged. Employers may reimburse their developers for conference registration fees and/or other expenses. P11 explained, "*they do offer umm, they will pay for us to go. Like if you want go to a conference that's, you know, in town, they'll pay for the fee to go to the conference.*"

Thus, attending conferences is an activity initiated by developers and encouraged by employers, where learning is an explicit goal (**E**). Conference presenters are often considered subject-matter experts (🏛), and likely external to the company (🡒). The cost of this activity varies depending on the conference registration fees, and whether it includes travel and accommodation expenses.

Searching online ($B 🏛 ⚙ 🡒)
Several online resources are available to help developers in their job-related tasks, however such resources vary in structure and credibility. These range from personal blogs and knowledge markets (e.g.

Stack Overflow [58]), to more official resources [e.g. National Vulnerability Database [81] and Common Vulnerabilities and Exposures (CVEs) [82]). We found that when participants use online resources to fix a security issue, this helps them learn about that particular security issue while working on their task. Thus, learning from this activity is considered part of the SDLC (⚙) and is a by-product of the activity (**B**). P3 explained, "*frankly, you know, Google search engine. I basically search things online. So, when there's something particular that I need to look into, I basically look it up online and see what the internet says.*" He also explained that he would prefer using "*official resources or more reputable sources*," rather than a blog.

Reading information and discussion websites ($E 🏛 ⚙ 🡒**)**
This activity involves the same resources as in the `Searching Online` activity described above. However, we categorize this into a distinct activity as participants' motives for accessing these resources are different. Contrary to `Searching Online`, participants in this activity access the internet resources with the explicit goal of learning about security. Thus, learning in this activity is explicit (**E**) and not part of the SDLC (🡒).

Participants explained that they use internet resources to stay up-to-date on security vulnerabilities. For example, P9 explained his strategy, "*[I follow a] couple of blogs, just general websites as well that might point out some new vulnerability. If I want to go in depth on something, then, you know, we can read about its CVE for that thing.*" Our participants' recounts of their use of discussion websites indicates that they did not actively participate in discussions, rather they were passive learners reading about the security topic and the available discussion.

Informal learning

Informal learning activities identified herein fall under "learning by experience" [76]. For all these activities, learning is a by-product (**B**) of the development tasks performed as part of the SDLC (⚙).

Participating in mediated social contact opportunities ($B 🏛 ⚙ 🡒**)**
Some participants gained security knowledge while participating in group activities enabled by their employers. For example, some participants mentioned that they discuss work impediments during team meetings, including security issues they face and how these could be addressed. Others mentioned that they work in open-plan offices which often stimulates discussions. P10 said, "*We all sit relatively close together, so if someone finds something, they might just sort of say 'OK, does anyone know about this?' 'Why are we doing it this way?'*" P10 explained that they value these general discussions as they allow developers to stay informed about security vulnerabilities and prevent vulnerabilities in their code.

Collaborating in the workplace ($B 🏛 ⚙ 🡒**)**
We found that participants learned about security during collaborations with members of other teams through interactions and discussions between teams. In our interviews, participants described multiple instances of different teams working together, e.g. testers working with developers to better understand the purpose of the code, and thus being able to better analyze potential vulnerabilities. P2 explained, "*Usually, if [the testers] think there's a problem, they really wouldn't go ahead and publish the bug like this; they would work with us. They'd be like, 'do I understand this correctly? Is this the correct behaviour?' [...] So, it's a process before the bug actually gets submitted.*" Such collaboration with other teams provides mutual benefits; it allows for "*information sharing*" (P11) between the different teams, can help bridge the knowledge gap [68], and reduce conflicts. Conversely, poor communication and a disconnect between teams who are working together leads to tension; this can result in conflicts between the teams and poor security outcomes. P8 exemplifies this through his development team's frustration with the testing team, "*[The relationship between the testing and development team is] bad. [chuckles] I mean, usually, you just pass them the code and then, they run through test cases and, you know, if they fail, they'll just come back say 'fail'. And then they don't..., because people who [are] doing the testing, they have zero knowledge about the code itself.*" Such disconnect may be due to conflicting goals between teams (e.g. functionality for developers, and security for security testers), and may require building a shared sense of responsibility between the teams. The section "Multiple activity systems interacting within project teams" discusses workplace collaborations in more detail.

Seeking help ($B 🏛 ⚙ 🡒**)**
Participants also described multiple instances where they turned to their colleagues for help. This is different from `Collaborating in the Workplace`, as help seeking here is informal and random, occurring only when the developer needs help while working on their tasks, rather than, e.g. a follow-up on testing results. In addition, the developer seeking help is usually the main beneficiary of the knowledge shared. P4 explained, "*if I need advice from someone, I would usually ask, you know; 'I'm looking at this thing here, how would you go with doing it?' We kind of just talk back and forth, it's usually pretty free form and open.*" Our participants also mentioned that they sometimes seek help to answer more specific questions relating to specific security issues. For example, P1 explained that if she cannot fix a security issue in her code, she asks a teammate who faced a similar issue how they fixed it. Although this activity is initiated by the developer, it is sometimes performed even by unmotivated developers albeit after procrastinating. P2 explained, "*In my experience [some devs] would delay [asking for help]. They'd work on things for months and then over coffee they'd be telling me what they're looking at and I'd break it in 2 minutes [...]. And they would be like [chuckle] 'okay, let's do this again'.*"

In summary, all the activities (Formal, Semi-Formal, and Informal) discussed herein improve developers' security knowledge, though participants' favored in-context learning activities (Semi-Formal and Informal). In-context learning may be preferable since it fits within developers' existing objectives, thus allowing developers to learn about security while working on their routine development tasks. In other words, casual and social exchanges of security information during regular work activities are more meaningful, and the encouragement of a culture of security knowledge sharing is beneficial. Participants also found (Formal) security training activities useful, however it is important to adapt the frequency of training according to the needs of the trainees and to incorporate the training activities into their regular workload.

Motivation for software security

In this section, we focus on *RQ2*, specifically what motivates developers to adopt or forgo security practices and tasks.

Through our analysis (see Fig. 3), we identified different motivations to adopt software security practices, as well as several factors that may induce developers' amotivation despite their knowledge and belief of the importance of security. Classifying the motivations as either intrinsic or extrinsic was too simplistic (e.g. the extrinsic motivations we identified varied in their driving forces from an external

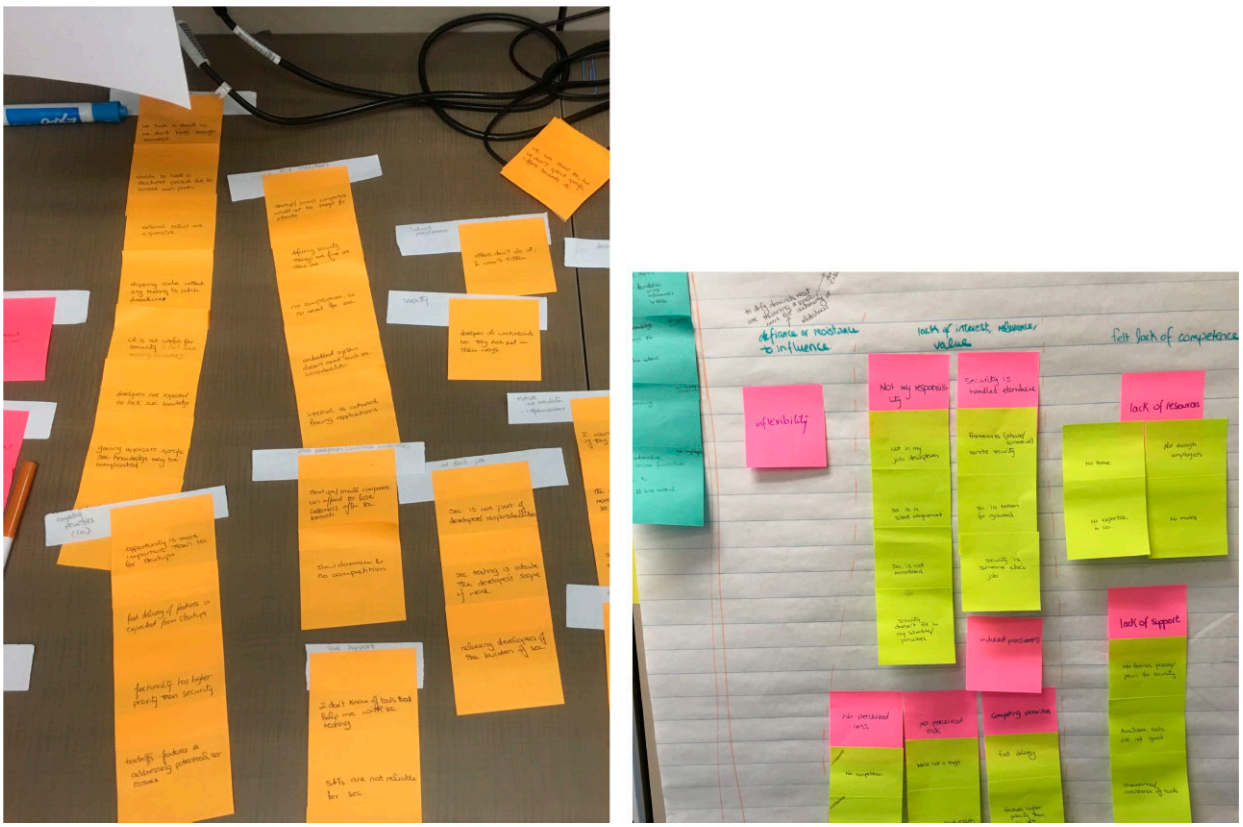


Figure 3. Analyzing motivations and amotivations for software security. Left: looking for patterns, right: identified patterns in amotivation.

mandate to the developer's sense of responsibility), thus we use SDT's *autonomy-control continuum* [37,39] to present our results. This continuum, explained in the section "Theoretical background on Self-Determination Theory," describes human motivation in learning environments using motivators ranging from extrinsic forces (e.g. compliance to external requirements) to intrinsic forces (e.g. self-interest). Table C1 in Appendix C, presents qualitative analysis codes corresponding to the software security (a)motivations found in our data, explains each code, and presents a corresponding sample quotation.

Figure 4 presents (a)motivations identified in our study on the self-determination continuum. At the leftmost end of the continuum (colored orange), we present amotivations that led participants (or their teams) to neglect software security. To the right, we present software security motivations. As we move toward the right, activity regulation increases in autonomy. Motivations under "external regulation" and "introjected regulation" are colored separately (yellow), because these motivations are not truly internalized (i.e. these motivations do not come from within the actor) and are contingent on their perceived outcomes (e.g. they are performed to comply with regulations or to maintain self-esteem). Motivations under "identified regulation" and "integrated regulation" (light blue) are more internally driven, and along with "intrinsic motivation" (dark blue), they present the most favorable types of software security motivations.

Note that security practice herein refers to practices developers *can* do to address security in their code, which may differ from security best practices such as performing static analysis or threat modeling [28]. We focus on these practices since our goal is to understand developers' security motivations (i.e. willingness to address security issues) rather than compare with best practices.

Amotivation

Through our data analysis, we identified three main reasons for participants' neglect of software security.

Amotivation—perceived lack of competence

Our analysis revealed that a `lack of resources` and a `lack of support` are two factors that led to a perceived lack of competence to address software security. Some participants indicated that their teams do not have the necessary budget, time, people-power, or expertise, to properly address security in their SDLC. We also found that this lack of trust in their ability to address security occurs when teams do not have a security plan in place, when security tools are nonexistent or lacking, and when developers are unaware of the availability of such tools. For example, P4 said, "*I wish I knew of [security] tools, but unfortunately I don't really know of any tool. So, I would probably be happy to say I would like to use some tools, but I don't know of any. I kinda wish I did, but I don't.*"

Amotivation—lack of interest, relevance, or value

This type of amotivation comes from the lack of interest, relevance, or perceived value of performing security tasks. The lack of relevance happens when security is not considered to be one of the developer's everyday duties (`not my responsibility`), or when security is viewed as another entity's responsibility (`security is handled elsewhere`), such as by another team or team member. Our analysis shows that when this is the general attitude within a team, it can have detrimental effects such as `induced passiveness`. It could lead developers (even those who believe in the importance of addressing security) to become demotivated toward security and rather focus on their 'more valuable' existing duties. For example, P9 said, "*I don't*

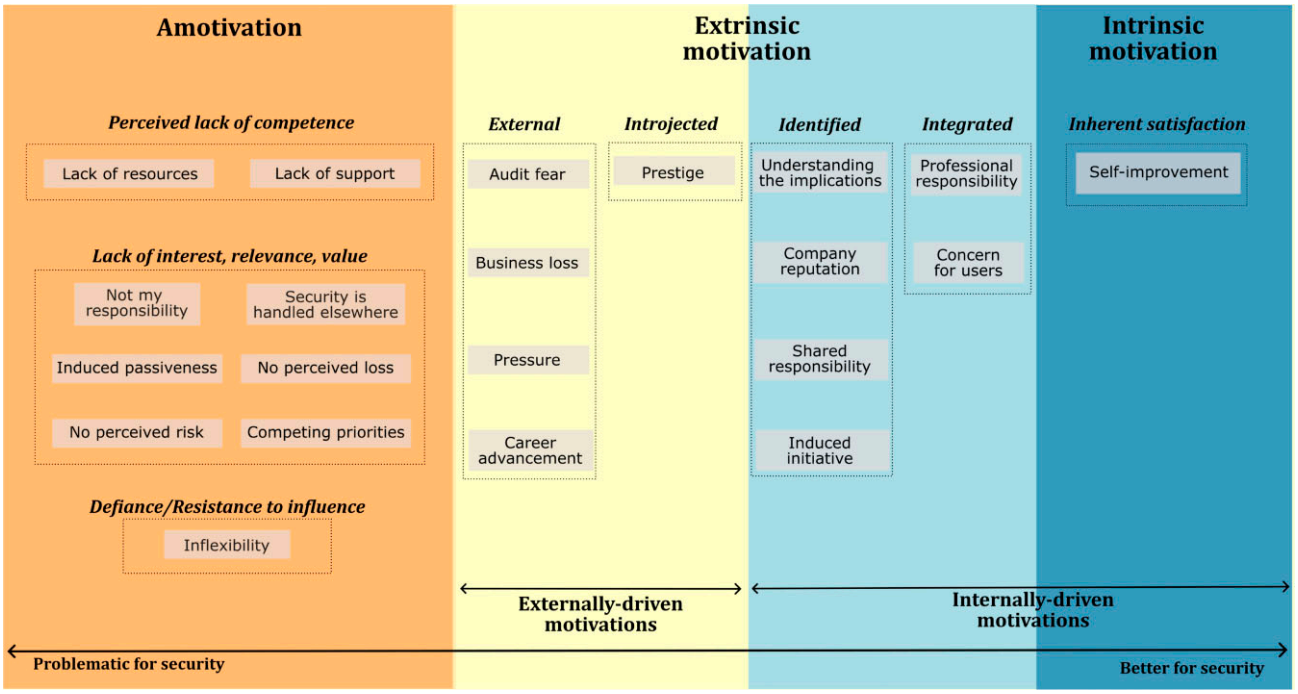


Figure 4. The self-determination continuum of software security. Amotivations (orange) are presented at the leftmost end of the continuum (problematic for security). Towards the right, motivations are presented in increasing order of activity regulation (better for security). In the middle, extrinsic motivations are shown in two different blocks to highlight the externally driven motivations (yellow) and the internally driven motivations (light blue). Intrinsic motivations, at the rightmost end (blue), are always internally driven and are more favorable for security.

really trust [my team members] to run any kind of like source code scanners or anything like that. I know I'm certainly not going to."

Additionally, our analysis shows different reasons why security efforts lack value for some participants in our dataset. First, we found that some participants are influenced by the optimistic bias [83], thinking that attackers would not be interested in their applications, or that they are not in an organization big enough to be a target for attacks. Thus, as they see `no perceived risk`, security efforts lack value. P7 said, "*For a small company, nobody will usually attack or compromise the vulnerabilities in your system. If something really bad happens, usually, you don't really get enough [bad] reputation as well.*" We also found that when there are no perceived negative consequences to the individuals or to the business from the lack of security (`no perceived loss`), then security efforts lack value. For example, when developers are not held responsible for security issues found in their code, they would rather spend their time on aspects for which they will be held responsible. P7 explained, "*[If] I made a bad security decision, nobody would blame me as much as if I made a decision that [led] to a [functionality] bug in the system. So the priority of security is definitely lower than introducing bugs in the system.*" Moreover, as different tasks compete for resources (e.g. the developer's time in the previous quote), when security has no perceived value, those tasks deemed more valuable are prioritized.

Amotivation—defiance/resistance to influence

The final amotivation we identified is `inflexibility`. We found that some developers ignored security, not because it is difficult to comply, but rather because it conflicts with their perception of the proper way of coding, or their personal coding practices. P9 explained how one of his team members resists using a framework in the proper way, despite having "*gotten into so many arguments*" with his manager, "*I can tell he is very self-absorbed with his own thoughts, and he thinks that what he says is somehow the truth, even if it doesn't necessarily pan out that way.*"

Extrinsic and intrinsic motivations

Externally driven motivations—external and introjected

Our analysis shows that addressing security can be driven by the desire to be recognized as the security expert or to receive acknowledgement (`prestige`), which helps in maintaining self-esteem and self-worth. P1 explained, "*When [somebody] clicks your name [on the employee website] and checks, it shows a badge that you're 'security certified', which gives you a good feeling.*"

In addition, we found three external motivations that are driven by the desire to avoid negative consequences associated with a lack of security: an overseeing entity finding non-compliance with regulations (`audit fear`), losing market share or market value due to a security breach (`business loss`), and being monitored and pressured by managers (`pressure`). P2 explained, "*We have a safety audit [conducted by an auditing organization]; all these guys they actually send auditors to us every, I don't know, however many months [..], and they look at the process. They, you know, scan every single check-in, every single review, [...] and they say 'oh, no! You haven't done that, you lose your certification.' [If] we lose our certification, [then] we have no company, we have no customers.*" Another external security motivation identified in our data is receiving rewards in the form of `career advancement` [e.g. "*promotions or mov[ing] throughout the scales and employment bands*"(P5)].

Internally driven motivations—identified and integrated

As shown in Fig. 4, there are three types of internally driven motivation, ordered from left to right in the figure by the degree of internalization. Those to the right are considered most favorable [38].

We first discuss those falling in the extrinsic motivation category, specifically motivations with Identified and Integrated regulation (cf. section "Theoretical background on Self-Determination Theory"). Discussion of intrinsic motivation follows later.

For extrinsic motivation, `professional responsibility` and `concern for users` are two motivations where the action is not performed for its inherent enjoyment, but rather to fulfill what the developer views as their responsibility to their profession and to safeguard users' privacy and security. These motivations are a type of integrated regulation, where the actor recognizes the importance of the tasks involved and thus accepts the goals associated with the tasks. For example, P3 said, "*I would not feel comfortable with basically having something used by end users that I didn't feel was secure, or I didn't feel respective of privacy, umm so I would try very hard to not compromise on that.*"

In addition, we identified motivations where participants view the goal of addressing security as personally important (i.e. identified regulation). For example, our analysis shows that `understanding the implications` of ignoring or dismissing security increased security awareness and motivated developers and their teams to integrate security in their SDLCs. P4 explained, "*I know for me personally when I realized just how catastrophic something could be, just by making a simple mistake, or not even a simple mistake, just overlooking something simple. uhh it changes your focus.*" This was especially true when the understanding came through practical examples of how the developer's code could lead to a security issue or through experiencing a real security issue at work. Caring about the `company reputation` and recognizing how it could be negatively affected in case of a security breach is another example of identified regulation motivation. Participants were also motivated to participate in security activities when security was a `shared responsibility` for the project team rather than the responsibility of one individual alone. This could in turn have a snowball effect and motivate additional team members to recognize the importance of security since their colleagues do (`induced initiative`). For example, P7 said, "*When you see your colleagues actually spending time on something, you might think that 'well, it's something that's worth spending time on', but if you worked in a company that nobody just touches security then you might not be motivated that much.*"

Internally driven motivations—inherent satisfaction

We classify `self-improvement` as (the only) intrinsic motivation to security. It is driven by the developer's own interest in security and the self-satisfaction of producing issue-free code. For example, P1 said, "*And sometimes I will challenge [myself], that 'okay, this time I'm going to submit [my code] for a review where nobody will give me a comment', though that never happened, but still...*"

Internalizing software security

During selective coding (the last coding stage in Grounded Theory), we recognized that our themes from the previous coding stage could be connected by a central theme about internalizing security—the driving force behind participants' security practices being their own will as opposed to external factors. In our data, we saw varied motivation toward security and varying degrees of internalization. For example, some participants spoke of the importance of security tasks and how they personally value these tasks, while others were indifferent to security and indicated that security tasks are only performed to satisfy an external driving force. Through our analysis, we found that participants who were internally motivated toward software security were more accepting and willing to adopt security practices.

We developed a human-oriented model that describes the process of internalizing software security based on our analysis (see Fig. 5). The end goal in this model is developers' internalization of software security, where they act toward security with autonomy and volition. This can apply to developers with varying level of motivations, including developers who are initially amotivated. In other words, the process of internalizing can begin at any point in the continuum of amotivations and motivations (shown in Fig. 4). Internally motivated actions are often associated with positive outcomes, such as increased engagement in the activity, improved performance, more cognitive flexibility, and better learning [38,40]. All of these outcomes would be useful and important for software security given its complexity and how it is commonly de-prioritized in software development in practice. As shown in Fig. 5, the two levers that influence this internalization are "(perceived) competence" and "relatedness"; the first refers to the developer's software security abilities and their perceptions thereof, while the latter is the developer's sense of connection to their project team. Naturally, all the software security learning opportunities discussed in Table 2 (indicated by the purple box in Fig. 5) help improve developers' (perceived) competence. Additionally, when these activities involve collaboration with other team members (indicated by the pink box in Fig. 5), this can improve relatedness through increasing the developer's bond with other members and their sense of belonging to the project team. With improving (perceived) competence and improving relatedness, developers' autonomy to act toward software security improves, thus encouraging internalization of software security. Internalization is a continuous process, with the learning opportunities identified in our data acting as the enduring impetus for improving competence and relatedness, and in turn creating two feedback loops fueling developers' internalization. We describe this process next.

Improving performance–valuing security loop

Acquiring software security knowledge and expertise (e.g. through company-organized security training or through receiving in-context support during development tasks) improves developers' competence and their confidence in their ability to address security in their code [i.e. *(perceived) competence*]. To encourage security learning among amotivated developers, employers could rely on activities initiated by the *employer* and activities initiated by the *employer and developer* (i.e. activities in the top two rows of the table in Fig. 5), with the aim of gradually moving these developers toward more internally driven motivations. Acquiring security knowledge drives the *improving performance—valuing security* loop (center left section of Fig. 5), as described next.

When developers' security knowledge improves, this improves (1) their *competence* (i.e. their awareness of the negative consequences of ignoring security, and their ability to address security) and (2) their *perceived competence* (i.e. their confidence in applying their knowledge). This leads to developers *valuing security* and thus encourages them to adopt security practices with *improved autonomy* (i.e. their actions are more internalized). In turn, with *improved autonomy*, developers persevere in their security practices (e.g. through critical thinking, asking for help from security experts) which results in *improving performance*. As developers observe positive outcomes from their security efforts, this further improves their *perceived competence*. And with each cycle through the loop, developers are likely to engage in further knowledge acquisition activities.

For example, a developer takes a course on Cross-Site Scripting (XSS) vulnerabilities [84], which improves their competence and per-

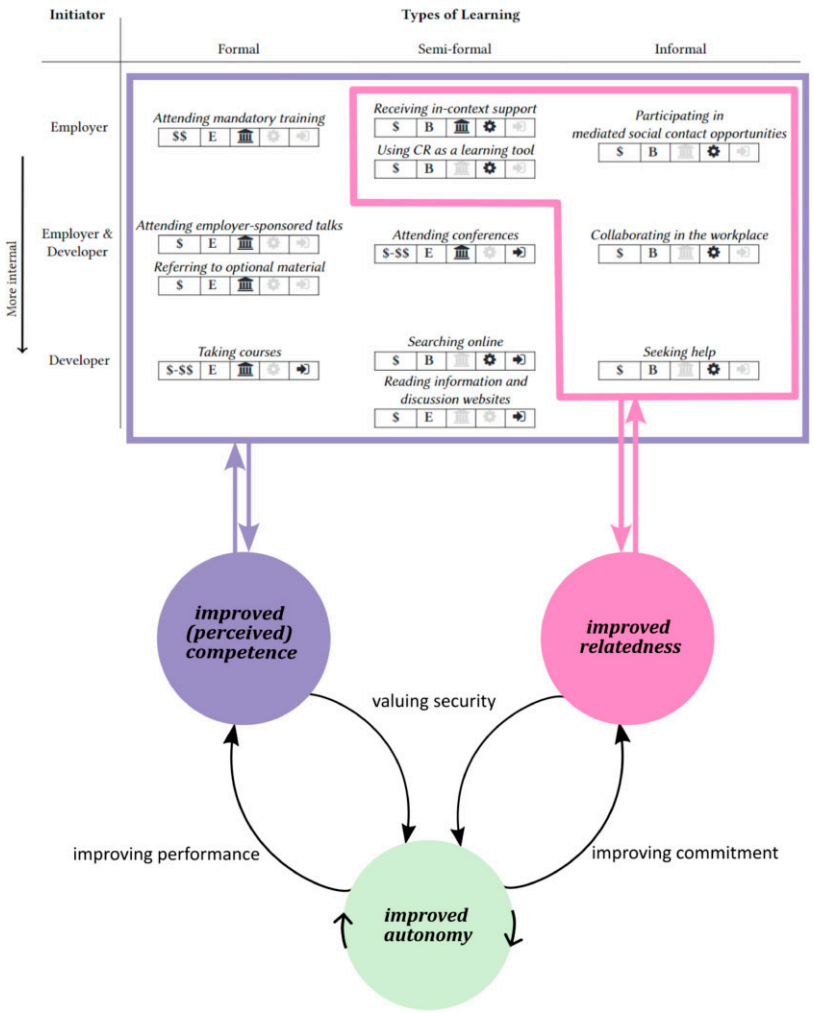


Figure 5. Internalizing software security model. The model shows the process by which the security knowledge activities (from Table 2) contribute to the continuous internationalization of security and to improving the developers' autonomy toward software security.

ceived competence. This helps the developer value security as they `understand the implications` (cf. Fig. 4) of having this vulnerability in their code, which could allow attackers to steal sensitive user data (`concern for users`). Equipped with this new security knowledge and confidence in their abilities, the developer is more internally driven toward addressing XSS vulnerabilities in their code (*improved autonomy*). This autonomy leads the developer to spend the time and effort on protecting their applications against XSS attacks, thus improving their performance and leading to *improved competence*. And the loop continues.

Improving commitment–valuing security loop

Many security learning activities we have identified involve collaboration among team members and across different teams within the company (shown within the pink box in Fig. 5). In addition to improving security knowledge, these activities help individuals develop a sense of relatedness to their project team and its security goals, driving the *improving commitment—valuing security* loop (center right section of Fig. 5). We describe next.

Collaboration encourages individuals to recognize the project team's overall goals beyond their own individual goals (e.g. functionality for developers or security for security engineers) and to feel connected to their team (*improved relatedness*). When the project team prioritizes security, this leads to individual developers also *valuing security*. Working toward this shared priority, each individual seeks to contribute to the team with volition (i.e. *improved autonomy*) and thus, becomes more internally driven toward adopting security practices. Improved autonomy leads to *improving commitment* from the developer toward their team and its security goals, which in turn leads to *improved relatedness* to the project team. With each cycle through the loop, the developer engages in further collaborative knowledge acquisition activities.

For example, a developer and a tester collaborate on fixing an XSS vulnerability in the developer's code, which includes discussing how this issue affects the team's software security goals. As the developer works together with the tester and feels that "*people are watching out for [her]*" [P10], the developer deepens her `sense of belonging` (cf. Fig. 4) to the team, which leads to *improved relatedness*. The developer will thus come to value security and view it as a `shared responsibility` (cf. Fig. 4), aligning with the objectives of the project team to which she belongs. Consequently, the developer is more internally driven to address the XSS vulnerability in her code (*improved autonomy*). With improved autonomy and willingness to contribute to her team, the developer becomes

increasingly committed (*improving commitment*) to this shared security goal. This in turn reinforces the developer's relatedness to her team and promotes further collaborative knowledge acquisition activities.

An ongoing process of internalization

As developers' (perceived) competence and relatedness increase, they gain and deepen their sense of belonging and their sense of responsibility, to their team, company, and society. Thus, developers go into a continuous process of internalizing the externally driven (or amotivated) security activities (i.e. their motivations move toward the right-side of Fig. 4), a process of active learning and self-growth [39]. And consequently, as developers' security practices are internalized, they can perform these practices with better performance, which improves the security of their applications [38,40].

Factors affecting internalization

Our previous analysis related to security practices [28] describes factors that affect security practices in development teams. Here, we discuss a subset of these factors that could help expedite, or hinder, the process of internalizing software security practices.

Prior security knowledge

The duration it takes to fully internalize software security (cf. Fig. 5) would likely be influenced by the developer's existing security knowledge and awareness. Under the same conditions, developers who already have prior background in security or have some awareness of its implications would likely internalize and accept security more readily than those who do not. In addition, developers who are recurrently exposed to security learning opportunities (e.g. through in-context learning activities) may be more likely to internalize security.

Company culture

As our analysis revealed, the attitude toward software security by the developer's team, supervisors, and those up in the company hierarchy has a substantial effect on the developer's motivation to learn about and internalize security. Developers whose teams view security as a shared goal and responsibility are more likely to adopt this view. Such teams typically follow a security plan. This can be both a cause and an effect of internalizing security; they develop a security plan to motivate security, and their motivation to security improves their security plan (recall that internalization is an ongoing process). Likewise, developers who recognize that security is valued by senior management can be inspired to take an interest in security. For example, P12 recounted, "*As I was working with [my CTO], he was telling me [about] all these different kinds of possible attack vectors that may happen, such as, what happens if the attacker gets access to the actual heap of the program, the memory [...] So stuff like that, I've never had to experience before [...] So, it was really, really interesting.*"

Resource availability

Resource availability can substantially affect motivation, both for individual developers and for the entire development team [28]. For example, when time is limited, developers preferred to prioritize their primary tasks; only those who are highly motivated to focus on security would ask for a deadline extension. In cases where the extension request was denied, security would either be deferred or the team would have to assign (or hire) extra personnel. However, with a limited budget, this may not always be possible.

Limitations and future work

Our sample size follows the norms this type of qualitative research methodology; we stopped recruitment only after reaching data saturation [73] (i.e. when new data did not add new insights relating to our research questions). A future study could use quantitative methods to assess the prevalence of the different aspects of our internalization model.

Like all interview-based research, our data is self-reported, which may not necessarily fully represent developers' reality. To address this, we assured participants that their responses would be anonymous and that they may skip any questions they are not comfortable answering, thus minimizing social desirability bias. To avoid priming participants on security, our study description did not mention "security" and the interview focused on all aspects of software development, which would naturally include security.

In this paper, we focus on software security in the professional context, so all our participants are employed as software developers. Thus, our results may not be generalizable to other types of developers (e.g. open-source developers). Additionally, all our participants' organizations are based in North America, hence our results may not be representative of organizations in other regions. Future work could focus on security knowledge acquisition opportunities and security (de)motivators of software developers in different contexts and locations. Another future research direction includes comparing the different security learning activities and examining their effectiveness so that organizations could prioritize those activities most likely to be effective for their context.

Future work could also examine how to integrate collaborative knowledge sharing opportunities into developers' existing communication methods and platforms. For example, it may be useful to introduce a security-focused branch within platforms like Stack Overflow [58] to help developers find and share security knowledge. We also suggest that future work should consider a deeper analysis of inter-team collaboration and how security knowledge flows across teams (i.e. how multiple activity systems interact during knowledge sharing, see the section "Multiple activity systems interacting within project teams" for further discussion).

Discussion and conclusion

In this paper, we focus on human aspects of software security in practice, specifically how developers acquire security knowledge (RQ1) and their motivations toward software security (RQ2). We now revisit the research questions and discuss our findings to provide further insights.

RQ1: Security knowledge acquisition

We created a Knowledge Acquisition Taxonomy (Table 2) categorizing the different security learning activities identified in our data. Formal learning was the least commonly reported learning type among our participants (cf. Table B1 in Appendix B). This included activities such as regularly scheduled training sessions, and learning resources (e.g. video lectures) provided by employers and often prepared by in-house security experts. In contrast, the most common security learning opportunities resulted as a by-product of the developer's tasks and responsibilities. Such tasks involved collaboration with the developer's teammates (e.g. pair programming) or with members from other teams within the organization. For example, during code review tasks, participants mentioned discussing security issues relating to their software with security testers, which increased participants' security awareness. In addition, this type of collaboration also al-

lows other teams (such as security testing teams) to learn about issues relating to software development. Collaborative activities such as these can bring together multiple perspectives (*multi-voicedness*, cf. the section "Theoretical background on activity theory") and can help bridge knowledge gaps between teams. This type of knowledge sharing can also motivate developers to internalize security, which in turns improves their perceived competence in addressing security issues, as discussed in the section "Internalizing software security."

RQ2: Software security motivations

We identified varying motivations (including amotivations) toward software security; we presented these using the autonomy-control spectrum of the Self-Determination Theory [37] (Section "Motivation for software security"). We found that developers tend to ignore security when they do not perceive value in focusing their efforts toward it. In particular, when security is not among their primary responsibilities, our participants tend to focus on the other considerations for which they will be held responsible. Such security devaluation can have dangerous consequences as it could demotivate even developers who consider security to be important. In contrast, we found that a team culture promoting security as a shared objective and responsibility motivates developers to value security and strive to produce secure code. By belonging to this culture, the developer is driven to internalize software security and thus acts toward it of their own will, e.g. out of personal interest or out of their concern for users. This further strengthens their bond to their team and its objectives. While developers may adopt security practices due to external motivators (e.g. to avoid negative consequences such as audit failures and loss of reputation), internally driven motivations are more favorable for security as per SDT, e.g. for improved performance, encouraging creativity, and fostering learning (Section "Theoretical background on Self-Determination Theory"). We developed a human-oriented model to describe the process of internalizing software security in the section "Internalizing software security."

Multiple activity systems interacting within project teams

Our internalization model (in Fig. 5) described how collaborative knowledge acquisition opportunities are beneficial for improving both security knowledge and motivations toward software security. We reflect on these collaborative opportunities identified in our data through the lens of Activity Theory (recall the section "Theoretical background on activity theory"). We look at a software development project team (consisting of a development team, a testing team, etc.) as a system of multiple interacting activity systems (see Fig. 1). We focus specifically on the interaction between the development team, represented by the *development activity system*, and the security testing team, represented by the *security testing activity system*. Each team considers the software from a different perspective and has its own background, points of views, and objectives (*multi-voicedness* [35]). For example, a development team would focus mainly on functionality, whereas a security testing team focuses on security.

Some project teams attempt to benefit from their multi-voicedness through communication, negotiations, and resolving conflicts. This allows the activity system to become more interconnected [35]. For example, when the development and security testing teams collaborate and harmonize their perspectives and objectives, this could lead to increasing security knowledge within both teams and to satisfying both teams' objectives: a functioning software for the development team and a secure software for the security testing team. Collaborative opportunities identified in our knowledge acquisition taxonomy (Table 2) can facilitate such inter-connectivity. These opportunities are also highlighted in the right section of Fig. 5. For example, when a developer and a security tester work together to fix a vulnerability (`Collaborating in the workplace`), we see knowledge acquisition happening across multiple activity systems.

P5 recounts that to support collaboration, the security testers "*have full access to the development team, so they can coordinate as much as they want.*" This collaboration allows testers to have a good understanding of the features they are testing, write better tests, and minimizes conflicts between testers and developers. P2 provided an example of how successful collaboration between developers and testers helped avoid overlooking a serious security issue due to a gap in the tester's understanding of the system. He said, "*a [tester] [...] was testing some memory issue in the kernel. While he was doing the test, he wanted to access kernel memory from the user process. So, his test actually succeeds to get to the kernel memory. He's [thinking] 'if I can get to the kernel memory, everything is fine, continue on.' So I look at the test and I'm like 'dude, [...] the test actually did discover a flaw, but you didn't tell me about it. This is a false negative.' [...] So, this [happened] because of the lack of understanding.*" In contrast, a lack of collaboration or breakdowns in the developer-tester relationship can be detrimental. For example, P8 described the relationship between the developers and the testing team as "*bad*" due to a lack of collaboration and a lack of testers' understanding of the software functionality.

Developers also benefit from such collaboration as it helps them stay updated on the constantly evolving security issues (e.g. [12,85,86]). Given that security is not the developer's primary objective, as evidenced by our data and previous work [16–19,24], it is unrealistic to expect that developers will be able to remain informed about these issues on top of their development tasks. In addition, security information is often presented in a manner that is unusable to developers [87,88]. Thus, collaborating with those with higher security expertise gives developers an opportunity to stay updated on new security issues, and it could also lead to improving performance and motivation toward adopting security practices.

Practical use for the knowledge acquisition taxonomy

The knowledge acquisition taxonomy presented in the section "Knowledge acquisition taxonomy" describes different security knowledge acquisition opportunities. Our data shows that implicit learning, especially when it is part of the SDLC, can be more effective than other types of learning, and can have a positive impact on software security. For example, our findings show that developers are more willing to engage in learning about security when it is combined with their existing tasks. This finding supports previous research recommending teaching developers about security in context [57,68].

Practitioners (e.g. employers, team leads) can use this taxonomy to induce software security learning opportunities within their organizations. With improved security knowledge, developers are more prepared to address software security, and, as detailed in the section "Internalizing software security," they may be more motivated and willing to do so.

Which activities should my organization adopt?

When choosing activities, practitioners should take into consideration the different features of each activity and developers' initiative. Based on our analysis, we have identified the following three main aspects to consider when deciding on activities to promote security knowledge.

Initiative. If the developer (being the learner) is motivated to learn about security, then all the activities listed in the taxonomy are suitable. However, in case of an amotivated developer, it is unlikely that they would initiate an explicit learning activity. Thus, the employer could instead look into initiating (or at least partly initiating) learning opportunities. Ergo, activities listed in the first, and perhaps the second row, of the taxonomy (Table 2) may be suitable. In addition, learning opportunities that are a by-product of the developer's main tasks (**B** ⚙) avoid competing for the developer's time and may be better received by such developers, especially those opposed to mandatory explicit learning (e.g. mandatory training).

Experience. Organizations that lack in-house security expertise should avoid activities that require high internal security expertise. They could consider activities without that requirement (i.e. those marked with 🏛), or activities with security expertise external to the company (i.e. those marked with 🏛 and ➔). When relying on external expertise, it is important to check the source's credibility and to encourage developers to use credible sources; developers often rely on external resources that are not necessarily ideal for security [60,89].

Budget. The available budget that the employer is willing to allocate for promoting security is another deciding aspect. Fortunately, most learning opportunities derived from our interviews are relatively low cost. However, as the affordability of an activity varies between companies, employers would need to decide on the most useful activity that fits their budget. In addition, some employers may wish to invest in more expensive learning opportunities, such as offering security courses to their developers, or by hiring external security experts to provide in-context support to developers. Although the latter did not come up in our interviews, it has been reported elsewhere [90].

Expanding the taxonomy

This taxonomy is not exhaustive, thus employers could map their own activities onto the taxonomy to compare and determine whether they are a good fit for them and their developers. For example, organizations sometimes use Capture The Flag (CTF) events to support security learning [91,92]. CTFs [93] are competitions where teams work together to solve coding challenges; developers thus learn through hands-on experience while socializing and competing on the challenges [91,92]. Within the taxonomy, CTFs would be considered as semi-formal learning opportunities, and placed in the *Employer & Developer* initiator row on Table 2, alongside conferences. Since the primary reasons for participating in CTFs are socialization, prizes, and bragging rights [91,92], learning is a by-product (**B**). CTFs are also not part of the SDLC (⚙), and the security expertise of team members can vary based on their individual experiences and knowledge (🏛). Knowledge could be from external (➔) or internal sources (➔), depending on the setup and location of the CTF.

Motivating developers toward adopting security practices

Finding the best way to motivate developers is not a trivial task. Even though external rewards and punishment may help induce external motivation, previous research in other domains [38,94–96] suggests that these approaches can have detrimental consequences, such as negatively influencing conceptual learning and problem solving. For example, externally motivated developers might gain limited knowledge about individual security issues rather than gaining an understanding of the underlying security concepts that may be necessary for solving new security issues. Thus, relying solely on external motivations may contribute to the poor performance and inadequate security practices identified in previous work (e.g. [28]). Of the participants who had external motivations for security (e.g. audits), those who also had internal motivations had better security practices than those who did not.

Based on our findings, internal motivations are more favorable for practicing software security compared to other motivation types. Thus, guided by our analysis, we built a security internalization model (Section "Internalizing software security") to explain how software security practices can be transformed to be internally motivated, rather than an external chores. Such transformation occurs by recognizing the value of incorporating software security and believing in one's ability to have an impact on the security of the software being built. To improve chances of success, security tasks should be accompanied by improving the team's morale when it comes to security. Based on our data, this can be through adopting a security culture, supporting developers in these tasks, providing positive encouragement, and allowing teams to see value and identify with such tasks. In addition, the internalization of security can be reinforced by promoting developers' security awareness which helps to improve their confidence in addressing security issues. Thus, we recommend that development teams focus on both increasing security awareness *and* improving developers' motivation to achieve positive security outcomes.

The role of Artificial Intelligence

Use of Artificial Intelligence in software development

Since our participant interviews, there have been significant advancements in Artificial Intelligence (AI) tools such as ChatGPT (https://chatgpt.com/). These tools are now used by developers [97,98] as a resource for explaining concepts and providing instant feedback [99], and to generate code or automate routine tasks [100]. However, developers relying on AI-generated code could introduce more security vulnerabilities compared to those not using AI, and yet have false confidence in the code's security [101]. In fact, Khoury et al. [102] identified various security vulnerabilities in AI-generated code, including code injection, buffer overflow, and XSS vulnerabilities. This could be the result of the AI models being trained on data obtained from the internet, which may include unverified or insecure code samples [101,103,104]. The security implications of adopting AI tools in software development remains unclear and warrants further investigation.

Impact of AI on security knowledge

Developers may learn about security from interactions with AI tools, e.g. when debugging code or searching for information. However, their learning may be limited to the particular solution they seek (e.g. how to request user input) without gaining an understanding of relevant security concepts (e.g. code injection attacks). Additionally, information generated by AI tools may be inaccurate, outdated, or incomplete [105]; and without contextual details relating to the developer's work (e.g. their application's threat model), AI tools' output may be of limited applicability [106]. The quality of AI-generated information is also greatly dependent on the specifics of the developer's prompt; minor prompt variations could substantially change the output [107]. Developers with limited security expertise may thus face challenges when using AI tools to obtain security information. On the other hand, developers who rely heavily on AI tools may perceive these tools as sufficient for their needs and become reluctant to engage in security knowledge acquisition activities (Section "Knowledge acquisition taxonomy").

Moreover, developers' increasing reliance on AI as a primary source of development-related information [108] may lead to reduced communication between team members. This raises concerns for software security as informal discussions within the workplace create opportunities for developers to gain relevant security knowledge (as discussed in the section "Internalizing software security"). Organizations may thus need to focus on adopting and creating opportunities that foster communication between project team members, such as mediated social contact activities or hosting CTFs.

Impact of AI on security motivation

Internalizing security (i.e. being internally motivated to adopt security practices, cf. the section "Internalizing software security") requires sustained relatedness to the project team and its security goals. As discussed above, developers' increasing reliance on AI tools could reduce opportunities for interaction between team members, which would lower their sense of relatedness to their team. As the developer becomes disconnected from their team, it becomes harder for her to value security or to view it as a shared responsibility, thus hindering the internalization of software security. Overreliance on AI tools could also lead to amotivation should the developer defer responsibility for software security to the AI tool. While AI tools come with promises for improved software development processes, further research is needed to fully understand their impact on security motivations.

Integration within the taxonomy

Despite the potential shortcomings discussed above, developers may use AI tools as a source for acquiring security knowledge. However, until security considerations become a priority for such tools, developers need to thoroughly review the generated output to ensure the information is reliable and secure [106,109]. For completeness, we now discuss how using AI tools like ChatGPT could fit within our taxonomy to allow for comparison with other knowledge acquisition activities (cf. the section "Practical use for the knowledge acquisition taxonomy"). The cost for this activity varies based on the type of the AI tool used *($-$$)* . Learning is typically a byproduct of the activity (**B**), and is integrated into the SDLC (⚙). Expertise is questionable due to potential inaccuracies (🏛) and the source of information is external to the organization (➔). In Table 2, this activity would be considered as a semi-formal learning opportunity, and would be placed in either the Employer & Developer row if the usage of AI tools is encouraged by the employer, or the Developer row if the usage is purely the developer's decision.

Conclusion

With an increasing number of security-sensitive software applications, it is essential for software developers to be aware of software security and stay motivated in order to address security in their applications. In this paper, we explicate security knowledge acquisition, developer motivation toward security and offer a framework for internalizing security. Our novel taxonomy can help practitioners recognize existing developer activities that may lead to advancing their security knowledge. In addition, it could help employers explore different learning opportunities and decide on the best methods to promote security knowledge within their organization. We envision this work to lead to increase in security awareness and in developers motivated toward improving the security of their software applications.

Author contributions

Hala Assal (Conceptualization, Formal analysis, Funding acquisition, Methodology, Supervision, Visualization, Writing–original draft, Writing–review & editing), Srivathsan G. Morkonda (Visualization, Writing–original draft, Writing–review & editing), Muhammad Zaid Arif (Writing–original draft, Writing–review & editing), Sonia Chiasson (Conceptualization, Formal analysis, Funding acquisition, Methodology, Supervision, Writing–original draft, Writing–review & editing)

Conflict of interest: There are no conflicts of interest.

Funding

H.A. acknowledges her NSERC Discovery Grant (RGPIN-2021-03808). S.C. acknowledges NSERC for funding of an Arthur B. McDonald Fellowship (SMFSA-566403-2022) and a Discovery Grant (RGPIN-2023-04653).

Appendix A: Interview script

The following questions represent the main themes discussed during the interviews. We may have probed for more details depending on participants' responses.

- What type of development do you do?
- What are your main priorities when doing development? (In order of priority)
- Do your priorities change when a deadline approaches?
- What about security? Is it something you worry about?
- Which are the best methods in your opinion for ensuring the security of software applications?
- How does security fit in your priorities?
- Which resources do you use to gain security knowledge?
- Do you get training (formal, or self-learning) to gain better knowledge of software security? How often?
- Which software security best practices are you familiar with?
- Are there any obligations by your supervisor/employer for performing security testing?
- What methods do you use to try to ensure the security of applications?
- Do you perform testing on your (or someone else's) applications/code?
- Do you perform code reviews?
- How would you describe the relation between the development and the testing team?
- Can you think of a story of security issue that was frustrating and how you dealt with it?

Appendix B: Distribution of participants' security learning opportunities

Table B1. Distribution of participants mentioning learning opportunities fitting in each cell of the knowledge acquisition taxonomy.

Initiator (More internal ↓)	Types of Learning: Formal	Types of Learning: Semi-formal	Types of Learning: Informal
Employer	*Attending mandatory training* P1, P2, P5	*Receiving in-context support* P1, P4, P6, P7, P9, P10 *Using CR as a learning tool* P1, P4, P6, P9	*Participating in mediated social contact opportunities* P2, P6, P7, P9, P11, P10, P11
Employer & Developer	*Attending employer-sponsored talks* P2, P3 *Referring to optional material* P1, P3, P9	*Attending conferences* P2, P3, P5, P9	*Collaborating in the workplace* P1, P2, P3, P5, P6, P7, P9
Developer	*Taking courses* P4, P6, P9	*Searching online* P3, P8, P9 *Reading information and discussion websites* P4, P7, P9, P10	*Seeking help* P1, P2, P4, P7, P8, P9, P10

Appendix C: Motivations and amotivations for software security

Table C1. Motivations and amotivations of software security.

Code	Description	Example Quote
	Amotivation - Felt lack of competence	
Lack of resources	The shortage in resources, e.g. budget and human power, needed to perform security tasks	"*We don't have that much manpower to explicitly test security vulnerabilities, [..] we don't have those kind of resources. But ideally if we did have [a big] company size, I would have a team dedicated to find exploits, um, that sorta thing. But unfortunately we don't.*"
Lack of support	The inadequate security tools and processes, or the lack thereof	"*We don't have any formal process of like a code review, sitting down and talking about security risks*"
	Amotivation - Lack of interest, relevance, value	
Not my responsibility	Security is not part of my duties	"*Developers are similar to me, they don't care that much about security or it's not part of their day to day job, therefore they don't pay much attention to the security aspect of the code.*"
Security is handled elsewhere	Security is another entity's responsibility	"*I usually don't as a developer go to the extreme of testing vulnerability in my feature, that's someone else's to do.*"
Induced passiveness	The surrounding environment causes passiveness toward security	"*I don't really trust them [my team members] to run any kind of like source code scanners or anything like that. I know I'm certainly not going to.*"
No perceived loss	The lack of competition, expected repercussions, and loss	"*I can introduce a big security issue and I definitely won't be blamed that much for it*"
No perceived risk	The company or application type is perceived as not a valuable target for attacks	"*For a small company, nobody will usually attack or compromise the vulnerabilities in your system. If something really bad happens, usually, you don't really get enough [bad] reputation as well.*"
Competing priorities	Other tasks compete for resources and are prioritized over security	"*I have security issues that are frustrating, but I haven't been able to deal with them yet. [...] It's not something that we've been able to deal with yet, just cause of priorities with everything else.*"
	Amotivation - Defiance/Resistance to influence	
Inflexibility	The resistance to new technology and being set in one's way	"*[My team is] using a framework and these guys, they used the framework incorrectly, they didn't like how certain part of this coding framework works and has been designed, so they decided to do things completely different than it [...] And I am sure it's gonna result in a security risk down the line.*"

Table C1. Continued

Code	Description	Example Quote
	Extrinsic Motivation - External	
Audit fear	The presence of an overseeing and supervising entity	"*One of the main reasons that they did [address security] was audits. I think they had to comply with certain security regulation standard, basically every quarter or so they're being checked for compliance, therefore they had the make sure the auditors can't find any issue during the penetration test.*"
Business loss	Losses that a business can incur, e.g. losing customers, due to security issues	"*We ended up ignoring security until we got a decent customer base where we were actually concerned that if our product was compromised, we will lose these customers.*"
Pressure	Continuous pressure by superiors	"*If they find a security issue, then you will be in trouble. Everybody will be at your back, and you have to fix it as soon as possible.*"
Career advancement	Software security efforts and knowledge move employees up in the hierarchy	"*When it comes time to do promotions or move throughout the scales and employment bands, the people with the higher knowledge on everything move up and the people who don't necessarily, like, didn't take those security training seriously, [...] they sort of stay in the same range.*"
	Extrinsic Motivation - Introjected	
Prestige	Acknowledgement and preserving self-image	"*Whenever somebody wants to find about you, then they go and check you in the employee website. Then, when they click your name and check, it shows a badge that you're security certified, which gives you a good feeling.*"
	Extrinsic Motivation - Identified	
Understanding the implications	Recognizing and understanding the potential implications of ignoring security	"*Just understanding the implications, I guess, of what could happen [would motivate developers be more security-oriented]. I know for me personally when I realized just how catastrophic something could be, just by making a simple mistake, or not even a simple mistake, just overlooking something simple. uhh it changes your focus.*"
Company reputation	The company and its employees care about their reputation and how customers perceive the company	"*We need to know safe secure coding techniques, we need to know what paths the attackers might take, and have you fixed everything on your code and your code doesn't have any vulnerabilities. [...] because finally, it is going to go under your logo.*"
Shared responsibility	The responsibility of software security is shared among different teams within the project team	"*[If we find a vulnerability,] we try not to say, 'you personally are responsible for causing this vulnerability'. I mean, it's a team effort, people looked at that code and they passed on it too, then it's shared, really.*"
Induced initiative	Opportunities may exist that lead developers to take the software security initiative	"*When you see your colleagues actually spending time on something, you might think that 'well, it's something that's worth spending time on', but if you worked in a company that nobody just touches security then you might not be motivated that much.*"
	Extrinsic Motivation - Integrated	
Professional responsibility	Feeling responsible as a professional	"*I would hesitate to release anything that's not functional and I also hesitate to release anything that had security concerns.*"
Concern for users	Caring about users' privacy and security	"*I would not feel comfortable with basically having something used by end users that I didn't feel was secure, or I didn't feel respective of privacy, umm so I would try very hard to not compromise on that.*"
	Intrinsic Motivation	
Self-improvement	The interest in, and self-satisfaction from, improving one's implementation	"*And sometimes I will challenge [myself], that 'okay, this time I'm going to submit [my code] for a review where nobody will give me a comment', though that never happened, but still...*"

REFERENCES

1. Assal H, Chiasson S, Biddle R. Cesar: visual representation of source code vulnerabilities. In: *2016 Symposium on Visualization for Cyber Security (VizSec)*, Baltimore, MD, USA: IEEE, 2016, 1–8.
2. Backes M, Rieck K, Skoruppa M,. *et al.* Efficient and flexible discovery of PHP application vulnerabilities. In: *2017 European Symposium on Security and Privacy (EuroS&P)*, Paris, France: IEEE, 2017, 334–49.
3. Chess B, McGraw G. Static analysis for security. *IEEE Secur Priv* 2004;**2**:76–9. https://doi.org/10.1109/MSP.2004.111
4. Smith J, Johnson B, Murphy-Hill E,. *et al.* Questions developers ask while diagnosing potential security vulnerabilities with static analysis. In: *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering. ESEC/FSE 2015*. New York, NY: ACM, 2015, 248–59.
5. Espinha Gasiba T, Lechner U, Pinto-Albuquerque M,. *et al.* Is secure coding education in the industry needed? An investigation through a large scale survey. In: *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*, Madrid: ES, 2021, 241–52.
6. Weir C, Becker I, Noble J,. *et al.* Interventions for long-term software security: creating a lightweight program of assurance techniques for developers. *Software Pract Exp* 2020;**50**: 275–98. https://doi.org/10.1002/spe.2774
7. Microsoft Corp. Microsoft Security Development Lifecycle. https://www.microsoft.com/en-us/securityengineering/sdl/practices. (April 2024, date last accessed).
8. Mandiant. Mandiant Unveils M-Trends 2023 Report, Delivering Critical Threat Intelligence Directly from the Frontlines. 2023. https://www.mandiant.com/company/press-releases/m-trends-2023. (June 2024, date last accessed).
9. NIST. CVSS Severity Distribution Over Time. https://nvd.nist.gov/general/visualizations/vulnerability-visualizations/cvss-severity-distribution-over-time. (September 2024, date last accessed).

10. Greenberg A. Hackers Remotely Kill a Jeep on the Highway—With Me in It. 2015. https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/. (May 2017, date last accessed).
11. Gitlin JM. Hackers discover that vulnerabilities are rife in the auto industry. 2023. https://arstechnica.com/cars/2023/01/hackers-discover-that-vulnerabilities-are-rife-in-the-auto-industry/. (June 2024, date last accessed).
12. Radcliffe J. Hacking Medical Devices for Fun and Insulin: Breaking the Human SCADA System. 2011. https://media.blackhat.com/bh-us-11/Radcliffe/BH_US_11_Radcliffe_Hacking_Medical_Devices_WP.pdf. (February 2017, date last accessed].
13. Pance L. Hackers Turn Smart Fridges into Cryptocurrency Miners, Causing Global Kitchen Meltdown. 2024. https://techreport.com/news/hackers-turn-smart-fridges-mining-rigs/ (June 2024, date last accessed).
14. Blank R, Gallagher P. NIST Special Publication 800-30 Revision 1: Guide for Conducting Risk Assessments. 2012. NIST Technical Series Publication, https://csrc.nist.gov/pubs/sp/800/30/r1/final. (June 2024, date last accessed].
15. Whitten A, Tygar JD. Why Johnny Can't Encrypt: A Usability Evaluation of PGP 5.0.In: *USENIX Security Symposium*. Vol. **348**. 1999.
16. Acar Y, Fahl S, Mazurek ML. You are Not Your Developer, Either: A Research Agenda for Usable Security and Privacy Research Beyond End Users. In: *2016 Cybersecurity Development (SecDev)*, Boston, MA, USA: IEEE, 2016, 3–8.
17. Green M, Smith M. Developers are Not the Enemy!: The Need for Usable Security APIs. *IEEE Secur Priv* 2016;**14**:40–6. https://doi.org/10.1109/MSP.2016.111
18. Tahaei M, Vaniea K. A Survey on Developer-Centred Security. In: *2019 European Symposium on Security and Privacy Workshops (EuroS&PW)*, Stockholm, Sweden: IEEE, 2019, 129-38.
19. Mokhberi A, Beznosov K. SoK: Human, Organizational, and Technological Dimensions of Developers' Challenges in Engineering Secure Software. In: *Proceedings of the 2021 European Symposium on Usable Security*. New York, NY, USA: ACM, 59–75.
20. Witschey J, Xiao S, Murphy-Hill E. Technical and Personal Factors Influencing Developers' Adoption of Security Tools. In: *Proceedings of the 2014 ACM Workshop on Security Information Workers, SIW '14*. New York, NY, USA: ACM, 2014, 23–26.
21. Xiao S, Witschey J, Murphy-Hill E. Social Influences on Secure Development Tool Adoption: Why Security Tools Spread. In: *Proceedings of the 17th ACM Conference on Computer Supported Cooperative Work and Social Computing, CSCW '14*. New York, NY, USA. ACM, 2014, 1095–106.
22. Xie J, Lipford HR, Chu B. Why do programmers make security errors? In: *2011 Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, Pittsburgh, PA: IEEE, 2011, 161–64.
23. Marshall BK. Passwords Found in the Wild for January 2013. http://blog.passwordresearch.com/2013/02/. (April 2017, date last accessed).
24. Wurster G, van Oorschot PC. The Developer is the Enemy. In: *Proceedings of the 2008 New Security Paradigms Workshop, NSPW '08*. New York, NY: ACM, 2008, 89–97.
25. Votipka D, Fulton KR, Parker J,. *et al*. Understanding Security Mistakes Developers Make: Qualitative Analysis from Build It, Break It, Fix It. In: *Proceedings of the 29th USENIX Conference on Security Symposium*, USENIX Association, 2020, 109–26.
26. Oliveira D, Rosenthal M, Morin N,. *et al*. It's the Psychology Stupid: How Heuristics Explain Software Vulnerabilities and How Priming Can Illuminate Developer's Blind Spots. In: *Proceedings of the 30th Annual Computer Security Applications Conference, ACSAC '14*. New York, NY: ACM, 2014, 296–305.
27. Baca D, Petersen K, Carlsson B,. *et al*. Static Code Analysis to Detect Software Security Vulnerabilities - Does Experience Matter?In: *2009 International Conference on Availability, Reliability and Security*, Fukuoka, Japan: IEEE, 2009, 804–10.
28. Assal H, Chiasson S. Security in the Software Development Lifecycle. In: *Fourteenth Symposium on Usable Privacy and Security (SOUPS 2018)*. Baltimore, MD: USENIX Association, 2018.
29. Assal H, Chiasson S. Motivations and Amotivations for Software Security. In: *SOUPS Workshop on Security Information Workers (WSIW)*. Baltimore, MD: USENIX Association, 2018.
30. Lopez T, Tun TT, Bandara AK,. *et al*. Taking the Middle Path: Learning About Security Through Online Social Interaction. *IEEE Software* 2020;**37**:25–30. https://doi.org/10.1109/MS.2019.2945300
31. Braz L, Bacchelli A. Software Security during Modern Code Review: The Developer's Perspective. In: *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2022*. New York, NY: ACM 2022, 810–21.
32. Engeström Y. Learning by expanding. *Center for Activity Theory and Developmental Work Research*.Orienta-konsultit 1987.
33. Engeström Y, Miettinen R, Punamäki RL. *Perspectives on Activity Theory*. Cambridge: Cambridge University Press, 1999.
34. Kuutti K. Activity theory as a potential framework for human-computer interaction research. *Context and Consciousness: Activity Theory and Human-Computer Interaction* 1996;17–44
35. Engeström Y. Expansive Learning at Work: Toward an activity theoretical reconceptualization. *Journal Educ Work* 2001;**14**:133–56. https://doi.org/10.1080/13639080020028747
36. O'Connor K. In: *Activity Theory*. John Wiley and Sons, 2015. https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118611463.wbielsi188.
37. Deci E, Ryan RM. *Intrinsic Motivation and Self-Determination in Human Behavior*. US: Springer, 1985.
38. Ryan RM, Deci EL. Self-determination theory and the facilitation of intrinsic motivation, social development, and well-being. *Am Psychol* 2000;**55**:68. https://doi.org/10.1037/0003-066X.55.1.68
39. Ryan RM, Deci EL. *Self-determination theory: Basic psychological needs in motivation, development, and wellness*. New York, NY: Guilford Publications, 2017.
40. Vallerand RJ, Blssonnette R. Intrinsic, Extrinsic, and Amotivational Styles as Predictors of Behavior: A Prospective Study. *J Pers* 1992;**60**:599–620. https://doi.org/10.1111/j.1467-6494.1992.tb00922.x
41. Fulton KR, Votipka D, Abrokwa D,. *et al*. Understanding the How and the Why: Exploring Secure Development Practices through a Course Competition. In: *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. New York, NY: ACM, 2022.
42. Lipner S. The Trustworthy Computing Security Development Lifecycle. In: *20th Annual Computer Security Applications Conference*. IEEE, 2004, 2–13.
43. Fulton KR, Chan A, Votipka D,. *et al*. Benefits and Drawbacks of Adopting a Secure Programming Language: Rust as a Case Study. In: *Proceedings of the Seventeenth USENIX Conference on Usable Privacy and Security, SOUPS'21*. USENIX Association, 2021.
44. Braz L, Aeberhard C, Çalikli G,. *et al*. Less is More: Supporting Developers in Vulnerability Detection during Code Review. In: *Proceedings of the 44th International Conference on Software Engineering, ICSE '22*. New York, NY: ACM, 2022, 1317–29.
45. Danilova A, Naiakshina A, Rasgauski A,. *et al*. Code Reviewing as Methodology for Online Security Studies with Developers: A Case Study with Freelancers on Password Storage. In: *Proceedings of the Seventeenth USENIX Conference on Usable Privacy and Security, SOUPS'21*. USENIX Association, 2021.
46. Naiakshina A, Danilova A, Gerlitz E,. *et al*. "If you want, I can store the encrypted password": A Password-Storage Field Study with Freelance Developers. In: *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems, CHI '19*. New York, NY: ACM, 2019, 1–12.
47. van der Linden D, Anthonysamy P, Nuseibeh B,. *et al*. Schrödinger's Security: Opening the Box on App Developers' Security Rationale. In: *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*. New York, NY: ACM, 2020.
48. Haney JM, Lutters WG. Skills and Characteristics of Successful Cybersecurity Advocates. In: *Workshop on Security Information Workers, Symposium on Usable Privacy and Security (SOUPS)*. Santa Clara, CA: USENIX Association, 2017.

49. Nurgalieva L, Frik A, Doherty G. A Narrative Review of Factors Affecting the Implementation of Privacy and Security Practices in Software Development. *ACM Comput Surv* 2023;**55**:1–27. https://doi.org/10.1145/3589951
50. Kathrin Bednar SS, Langheinrich M. Engineering Privacy by Design: Are engineers ready to live up to the challenge? *Inform Soc* 2019;**35**:122–42. https://doi.org/10.1080/01972243.2019.1583296
51. Ryan I, Roedig U, Stol KJ. Understanding Developer Security Archetypes. In: *2021 IEEE/ACM 2nd International Workshop on Engineering and Cybersecurity of Critical Systems (EnCyCriS)*, Madrid, Spain, 2021, 37–40.
52. Spiekermann S, Korunovska J, Langheinrich M. Inside the Organization: Why Privacy and Security Engineering Is a Challenge for Engineers. *Proc IEEE*, 2019;**107**:600–15. https://doi.org/10.1109/JPROC.2018.2866769
53. Danilova A, Naiakshina A, Smith M. One Size Does Not Fit All: A Grounded Theory and Online Survey Study of Developer Preferences for Security Warning Types. In: *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering, ICSE '20*. 2020, 136–48.
54. Tahaei M, Vaniea K, Beznosov K,. *et al*. Security Notifications in Static Analysis Tools: Developers' Attitudes, Comprehension, and Ability to Act on Them. In: *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems, CHI '21*, New York, NY: ACM, 2021.
55. Acar Y, Backes M, Fahl S,. *et al*. Comparing the Usability of Cryptographic APIs. In: *2017 IEEE Symposium on Security and Privacy (SP)*, San Jose, CA, USA, 2017, 154–71.
56. Oyetoyan TD, Cruzes DS, Jaatun MG. An Empirical Study on the Relationship between Software Security Skills, Usage and Training Needs in Agile Settings. In: *2016 11th International Conference on Availability, Reliability and Security (ARES)*, Salzburg, Austria: IEEE, 2016.
57. Tuladhar A, Lende D, Ligatti J,. *et al*. An Analysis of the Role of Situated Learning in Starting a Security Culture in a Software Company. In: *Proceedings of the Seventeenth USENIX Conference on Usable Privacy and Security, SOUPS'21*. USENIX Association, 2021.
58. Stack Overflow - Where Developers Learn, Share, and Build Careers. https://stackoverflow.com. (January 2018, date last accessed].
59. Fischer F, Böttinger K, Xiao H,. *et al*. Stack Overflow Considered Harmful? The Impact of Copy&Paste on Android Application Security. In: *2017 IEEE Symposium on Security and Privacy (SP)*, San Jose, CA, USA, 2017, 121–36.
60. Acar Y, Backes M, Fahl S,. *et al*. You Get Where You're Looking for: The Impact of Information Sources on Code Security. In: *2016 IEEE Symposium on Security and Privacy (SP)*, San Jose, CA, USA, 2016, 289–305.
61. Díaz Ferreyra NE, Vidoni M, Heisel M,. *et al*. Cybersecurity Discussions in Stack Overflow: A Developer-Centred Analysis of Engagement and Self-Disclosure Behaviour. *Soc Netw Anal Mining* 2024;**14**:16.
62. Horvath A, Liu MX, Hendriksen R,. *et al*. Understanding How Programmers Can Use Annotations on Documentation. In: *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems, CHI '22*. New York, NY: ACM, 2022.
63. Arab M, LaToza TD, Liang J,. *et al*. An Exploratory Study of Sharing Strategic Programming Knowledge. In: *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems, CHI '22*. New York, NY: ACM, 2022.
64. Pieczul O, Foley S, Zurko ME. Developer-centered Security and the Symmetry of Ignorance. In: *Proceedings of the 2017 New Security Paradigms Workshop, NSPW 2017*. New York, NY: ACM, 2017, 46–56.
65. Poller A, Kocksch L, Türpe S,. *et al*. Can Security Become a Routine?: A Study of Organizational Change in an Agile Software Development Group. In: *Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing, CSCW '17*. New York, NY: ACM, 2017, 2489–503.
66. Woon IMY, Kankanhalli A. Investigation of IS professionals' intention to practise secure development of applications. *Int J Hum-Comput Stud* 2007;**65**:29–41. https://doi.org/10.1016/j.ijhcs.2006.08.003
67. Weir C, Hermann B, Fahl S. From Needs to Actions to Secure Apps? The Effect of Requirements and Developer Practices on App Security. In: *Proceedings of the 29th USENIX Conference on Security Symposium*. USENIX Association, 2020.
68. Thomas TW, Tabassum M, Chu B,. *et al*. Security During Application Development: An Application Security Expert Perspective. In: *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems, CHI '18*. New York, NY: ACM, 2018, 262:1–262:12.
69. Lopez T, Sharp H, Tun T,. *et al*. Talking About Security with Professional Developers. In: *2019 IEEE/ACM Joint 7th International Workshop on Conducting Empirical Studies in Industry (CESI) and 6th International Workshop on Software Engineering Research and Industrial Practice (SER&IP)*. 2019, 34–40.
70. Gasiba T, Lechner U, Pinto-Albuquerque M. "CyberSecurity Challenges for Software Developer Awareness Training in Industrial Environments". In: *Innovation Through Information Systems*. 2021, 370–87.
71. Gorski PL, Acar Y, Lo Iacono L,. *et al*. Listen to Developers! A Participatory Design Study on Security Warnings for Cryptographic APIs. In: *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems, CHI '20*. New York, NY: ACM, 2020, 1–13.
72. Ford D, Zimmermann T, Bird C,. *et al*. Characterizing Software Engineering Work with Personas Based on Knowledge Worker Actions. In: *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, Toronto, ON, Canada, 2017, 394–403.
73. Glaser BG, Strauss AL. *The discovery of grounded theory: strategies for qualitative research*. Aldine, 1967.
74. Forward A, Lethbridge TC. A Taxonomy of Software Types to Facilitate Search and Evidence-based Software Engineering. In: *Proceedings of the 2008 Conference of the Center for Advanced Studies on Collaborative Research: Meeting of Minds, CASCON '08*. New York, NY: ACM, 2008, 179–91.
75. Strauss AL, Corbin JM. Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory. Sage Publications, Inc., 1998.
76. Organisation for Economic Co-operation and Development (OECD). Recognition of Non-formal and Informal Learning - Home. http://www.oecd.org/edu/skills-beyond-school/recognitionofnon-formalandinformallearning-home.htm. (January 2018, date last accessed).
77. Dib CZ. Formal, non-formal and informal education: concepts/applicability. *AIP Conf Proc* 1988;**173**:300–15. https://doi.org/10.1063/1.37526
78. Eraut M. Non-formal learning and tacit knowledge in professional work. *Brit J Educ Psychol* 2000;**70**:113–36. https://doi.org/10.1348/000709900158001
79. Eshach H. Bridging In-school and Out-of-school Learning: Formal, Non-Formal, and Informal Education. *J Sci Educ Technol* 2007;**16**:171–90. https://doi.org/10.1007/s10956-006-9027-1
80. Stahl G. Conceptualizing the Intersubjective Group. *Int J Comput Supp Collab Learn*. 2015;**10**:209–17. https://doi.org/10.1007/s11412-015-9220-4
81. NIST. National Vulnerability Database. https://nvd.nist.gov. (March 2017, date last accessed).
82. CVE - Common Vulnerability Exposures. https://cve.mitre.org. (January 2018, date last accessed).
83. Rhee HS, Ryu YU, Kim CT. Unrealistic optimism on information security management. *Comput Secur* 2012;**31**:221–32. https://doi.org/10.1016/j.cose.2011.12.001
84. OWASP. Cross Site Scripting (XSS). https://owasp.org/www-community/attacks/xss/. (July 2024, date last accessed).
85. Howard M, Lipner S. *The security development lifecycle: SDL, a process for developing demonstrably more secure software*, Redmond, Wash: Microsoft Press, 2006.
86. Sophy J. 43 Percent of Cyber Attacks Target Small Business. 2016. https://smallbiztrends.com/2016/04/cyber-attacks-target-small-business.html. (February 2017, date last accessed).
87. Nafees T, Coull N, Ferguson I,. *et al*. Vulnerability Anti-Patterns: A Timeless Way to Capture Poor Software Practices (Vulnerabilities). In:

Pattern Languages of Programs Conference. USA: The Hillside Group, 2017.
88. Nafees T, Coull N, Ferguson RI,. *et al*. Idea-Caution Before Exploitation: The Use of Cybersecurity Domain Knowledge to Educate Software Engineers Against Software Vulnerabilities. In: Bodden E, Payer M, Athanasopoulos E, (eds.). *Engineering Secure Software and Systems*. Cham: Springer International Publishing, 2017, 133–42.
89. Fischer F, Böttinger K, Xiao H,. *et al*. Stack Overflow Considered Harmful? The Impact of Copy Paste on Android Application Security. In: *2017 IEEE Symposium on Security and Privacy (SP)*, San Jose, CA, USA, 2017, 121–36.
90. van Zadelhoff M. Cybersecurity Has a Serious Talent Shortage. Here's How to Fix It. https://hbr.org/2017/05/cybersecurity-has-a-serious-talent-shortage-heres-how-to-fix-it. (January 2018, date last accessed).
91. OWASP. OWASP CTF Project. https://www.owasp.org/index.php/Category:OWASP_CTF_Project. (January 2018, date last accessed].
92. Harmon TD. Cyber Security Capture The Flag (CTF): What Is It?. https://blogs.cisco.com/perspectives/cyber-security-capture-the-flag-ctf-what-is-it. (January 2018, date last accessed).
93. CTFtime. CTF? WTF? https://ctftime.org/ctf-wtf/. (January 2018, date last accessed).
94. Gagné M, Deci EL. Self-determination theory and work motivation. *J Organ Behav* 2005;**26**:331–62. https://doi.org/10.1002/job.322
95. Bear GG, Slaughter JC, Mantz LS,. *et al*. Rewards, praise, and punitive consequences: Relations with intrinsic and extrinsic motivation. *Teach Teach Educ* 2017;**65**:10–20. https://doi.org/10.1016/j.tate.2017.03.001
96. Selart M, Nordström T, Kuvaas B,. *et al*. Effects of Reward on Self-regulation, Intrinsic Motivation and Creativity. *Scand J Educ Res* 2008;**52**:439–58. https://doi.org/10.1080/00313830802346314
97. Shani I, Staff GitHub. Survey reveals AI's impact on the developer experience. 2023. https://github.blog/2023-06-13-survey-reveals-ais-impact-on-the-developer-experience/. (August 2024, date last accessed).
98. Overflow S. 2024 Developer Survey. https://survey.stackoverflow.co/2024. (August 2024, date last accessed).
99. Tubino L, Adachi C. Developing feedback literacy capabilities through an ai automated feedback tool. *Ascilite Publ* 2022; e22039. https://doi.org/10.14742/apubs.2022.39
100. Schulte-Althoff M. What's to Automate? A Task Analysis of AI-enabled Start-ups. *Proc 56th Hawaii Int Conf System Sci* 2023.
101. Perry N, Srivastava M, Kumar D,. *et al*. Do Users Write More Insecure Code with AI Assistants?In: *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, CCS '23*. New York, NY: ACM, 2023, 2785–99.
102. Raphael Khoury and Anderson R. Avila and Jacob Brunelle and Baba Mamadou Camara. How Secure is Code Generated by ChatGPT? 2023. https://arxiv.org/pdf/2304.09655.
103. Pearce H, Ahmad B, Tan B,. *et al*. Asleep at the Keyboard? Assessing the Security of GitHub Copilot's Code Contributions. In: *2022 IEEE Symposium on Security and Privacy (SP)*, 2022, San Francisco, CA, USA, 754–68.
104. Negri-Ribalta C, Geraud-Stewart R, Sergeeva A,. *et al*. A systematic literature review on the impact of AI models on the security of code generation. *Front Big Data* 2024;**7**.
105. OWASP. LLM Top 10 for LLMs v1.1. 2024. https://genai.owasp.org/resource/llm-top-10-for-llms-v1-1/.
106. Klemmer JH, Horstmann SA, Patnaik N,. *et al*. Using AI Assistants in Software Development: A Qualitative Study on Security Practices and ConcernsIn: *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, Salt Lake City, UT, USA, 2024.
107. Mastropaolo A, Pascarella L, Guglielmi E,. *et al*. On the Robustness of Code Generation Techniques: An Empirical Study on GitHub Copilot. In: *Proceedings of the 45th International Conference on Software Engineering, ICSE '23*, Melbourne, Victoria, Australia: IEEE, 2023, 2149–60.
108. Vizard M. Survey Surfaces Widespread Reliance on Generative AI Among Developers. 2024. https://devops.com/survey-surfaces-widespread-reliance-on-generative-ai-among-developers/. (August 2024, date last accessed).
109. Silva CAGd, Felipe NR, Rafael VdM,. *et al*. ChatGPT: Challenges and Benefits in Software Programming for Higher Education. *Sustainability* 2024;**16**:1–23.