An Analysis of the Role of Situated Learning in Starting a Security Culture in a Software Company

Anwesh Tuladhar, Daniel Lende, Jay Ligatti, and Xinming Ou, *University of South Florida*

https://www.usenix.org/conference/soups2021/presentation/tuladhar

This paper is included in the Proceedings of the Seventeenth Symposium on Usable Privacy and Security.

August 9–10, 2021

978-1-939133-25-0

An Analysis of the Role of Situated Learning in Starting a Security Culture in a Software Company

Anwesh Tuladhar Daniel Lende Jay Ligatti Xinming Ou
University of South Florida, Tampa, FL, USA
Email: {atuladhar, dlende, ligatti, xou} @usf.edu

Abstract

We conducted an ethnographic study of a software development company to explore if and how a development team adopts security practices into the development lifecycle. A PhD student in computer science with prior training in qualitative research methods was embedded in the company for eight months. The researcher joined the company as a software engineer and participated in all development activities as a new hire would, while also making observations on the development practices. During the fieldwork, we observed a positive shift in the development team's practices regarding secure development. Our analysis of data indicates that the shift can be attributed to enabling all software engineers to see how security knowledge could be applied to the specific software products they worked on. We also observed that by working with other developers to apply security knowledge under the concrete context where the software products were built, developers who possessed security expertise and wanted to push for more secure development practices (security advocates) could be effective in achieving this goal. Our data point to an interactive learning process where software engineers in a development team acquire knowledge, apply it in practice, and contribute to the team, leading to the creation of a set of preferred practices, or "culture" of the team. This learning process can be understood through the lens of the situated learning framework, where it is recognized that knowledge transfer happens within a community of practice, and applying the knowledge is the key in individuals (software engineers) acquiring it and the community (development team) embodying such knowledge in its practice. Our data show that enabling a situated learning environment for security gives rise to security-aware software engineers. We discuss the roles of management and security advocates in driving the learning process to start a security culture in a software company.

1 Introduction

A wide range of research has addressed how to best establish secure development practices within a software development team/company. The standard approach is to use a secure development model to formulate a suitable secure software development lifecycle (S-SDLC) [10,19,36,43]. Despite the success of S-SDLC in its originating company [19], successful establishment of secure development practices has remained more difficult for the software industry at large. In general, some companies are unwilling to place code security on a level playing field with business considerations such as time-to-market of the product. There are also companies that make an effort to deliver secure code through the adoption of secure development life cycles but have not been able to do so effectively. Reasons for such failures have been posited as lack of security knowledge in developers, lack of available resources, lack of usable security, improper use of security APIs, and so on [1, 12, 33, 35]. Use of security tools is often suggested as a way to help alleviate such problems by catching developer mistakes before they land in the product. However the adoption of security tools into development itself can remain an issue [34]. Some studies allude to the notion of security mindset or security culture within the company as the influential factor in driving these secure development practices [5, 17, 42]. But what is a security culture? What are the benefits it provides and how can a company start to develop such a culture?

We conducted an extensive ethnographic study in a software company in order to understand how and why secure coding practices are (or are not) integrated into software development processes. We embedded a computer science PhD student with training in qualitative methods as part of the

USENIX Symposium on Usable Privacy and Security (SOUPS) 2021.
August 8–10, 2021, Virtual Conference.

company's software development team. Approximately two months prior to the researcher joining, the company had initiated the process of implementing a secure development lifecycle, with upper-management declaring that security should receive greater consideration. This new emphasis on security provided us an opportune moment to conduct research into whether and how such push for security may result in concrete positive changes in the development processes. Our research was helped by the company empowering a recently hired software engineer who was also trained in security, to push for secure development practices within the development team. The main contributions of our work are as follows.

1. We identify an important factor in establishing secure development practices in a software company – the role of situated learning [20] that forms an integral part of software engineers' work. Rather than assuming structured processes on their own can solve security problems, we examine the context for learning about security within the development environment and analyze how it shapes the workflows followed by individual software engineers. Our analysis of data shows that what was driving the positive shift in the development team's security awareness can be explained by the learning dynamics existent therein, in particular the software engineers being able to identify the applicability of the security knowledge within the context of the everyday work they perform. The situated learning dynamics could drive the team into a set of agreed-upon knowledge and the associated practices, becoming the "preferred practices" for dealing with specific security concerns. We hence identify a way to start a secure coding culture in a development team.

2. Our data also indicate that the presence of a security expert working within the development team is instrumental in driving the situated learning cycle for security. It appears that when such security experts are part of the development team, and their actions foster the learning process, the adoption of secure coding practices become more readily accepted by the team. In particular, we find it important that security knowledge be offered within the context of the team's concrete work.

2 Fieldwork

Our fieldwork was conducted at a software development company headquartered in the United States with offices throughout the world. The researcher was embedded in a development team responsible for two security-related products developed by the company, referred to as P1 and P2 in this paper. Due to the ongoing COVID-19 pandemic, the mode of work varied between work-from-home and on-premise. The company followed local government guidelines; mandated work-from-home when stay-at-home order was in effect, and provided the flexibility of either work-from-home or on-premise otherwise. For on-premise work, the company followed all safety precautions by reorganizing the office setup to socially distance cubicles and providing masks and hand sanitizers. All meetings were held through video conferencing even when on premise. The advantages of being on premise were ease of access to the test environment and ability to start impromptu discussions and meetings when necessary.

2.1 The Development Team

The main participants were five software engineers (SWEs) in the development team, two network engineers, two support engineers, two sales/customer relations representatives, one quality assurance engineer (QAE), one graphic designer, and one vice president (VP), who also oversaw the management of the two products. All SWEs had at least 1.5 years' experience within the company, with two having more than five years' experience. The QAE joined during the fieldwork.

2.2 Research Methodology

We employed the qualitative research method of participant observation [7, 29]. In this method, the researcher spends an extended amount of time in the field taking part in day to day activities and practices. This method allows researchers to obtain in-depth understanding of practices such as software development that take place over long periods of time.

The participant observer in this research was a computer science PhD student with prior training in qualitative research methods as well as ample industry experience. This experience allowed the researcher to integrate quickly into the daily work. The researcher spent three days per week at the company for a duration of eight months. Although the researcher's background was in security, the researcher was not limited to security-related tasks and participated in all activities a regular SWE at the company would, such as sprint planning, scrum meetings, bug fixes, feature design/implementation/testing, and code reviews.

Our data analysis utilized the general inductive approach [30], augmented by specific techniques for qualitative data analysis such as analytic notes and comparative analysis [6]. The researcher maintained descriptive field notes on daily activities and interactions. After three months of data collection, the research team met weekly to reflect on the observations made so far. The team went over the events of the past week and discussed the events concerning software development practices, security practices, and the relevant interactions. For the security incidents encountered, we separately kept track of the process of identification, technical details of the issue, and the progress made towards mitigating them. The researcher coded the raw field data based on the patterns and themes that emerged during these discussions. Any unanswered questions and/or missing information during these discussions then guided the future observations in the field. The weekly iteration of data collections followed by in-depth discussions led to the refinement of the emerging categories used for coding (see Appendix for the final set of codes used in our research). Then, as broader themes were conceptualized, the researcher started to write analytic notes summarizing each theme and documenting ideas and analysis of each along with supporting data from exploration of code, tickets, and other relevant sources in addition to the raw fieldwork data. Multiple themes emerged and evolved throughout this process. After the end of the fieldwork, the research team continued further analysis of data through extensive discussions to draw out the major implications of our observations to secure software development practices.

The study was reviewed and approved by the Institutional Review Board (IRB). The researcher explained the study goals to participants and obtained verbal informed consent from them. Field notes were anonymized, as well as discussions during weekly research meetings.

3 Software Development Processes and Challenges Facing Secure Development

Approximately two months prior to the researcher joining the development team, management instructed the team to employ secure software development lifecycle (S-SDLC). This provided an invaluable opportunity for the research team to examine whether and how secure development practices can take hold in a software development team when there is buy-in from the top. In this section, we describe our observations of the company's overall software development processes and challenges facing secure development throughout our fieldwork. In section 4 we focus our discussion on observations and analysis of the shift in the development processes as a result of the management push for S-SDLC.

The company adopted a sprint-based agile development model. An issue-tracking tool was used for planning and tracking the development progress throughout a sprint. We describe this process below.

3.1 Sprint Planning

Everyone in the team was free to create a ticket for any work that was not already tracked and would be added to the *backlog* queue. However, only the lead SWEs could approve the ticket for development. In addition, the VP and customer facing specialists could add feature tickets based on company vision and customer requests/feedback. Each week the VP and the lead sales representative, along with the lead SWEs had a *prioritization meeting* where the new tickets were discussed, approved/rejected for future development, with the approved tickets ranked based on priority. For each sprint, the SWEs and QAE conducted a *sprint planning* meeting where the highest priority tickets from the backlog were discussed and assigned *story points* representing the estimated complexity/amount of work. Story points for each ticket were agreed upon by the whole team using SCRUM poker [41], where each SWE and QAE anonymously assigned story points based on their understanding of the required work. When the assigned scores varied widely, a discussion was held to allow each SWE/QAE to explain the reasoning for their scores and SCRUM poker was re-done, until everyone converged on a common score. A total of 60-70 story points were targeted for each sprint, which allowed for a small number of additional high-priority tickets or unforeseen issues to be included in the sprint at a later time.

3.2 Development Workflow

A short 20-30 minute scrum meeting was held every morning to provide brief updates on the progress from the previous day, any issues/roadblocks encountered, and goals for the current day. The meeting was led by the lead SWE and included all SWEs, the QAE, the graphic designer, and the VP. Additional meetings were called by individuals as required to discuss ticket requirements, design issues, knowledge transfer for codebase, or testing strategies. We next discuss the stages of development and the challenges facing secure development in the context of each stage.

3.2.1 Design

A high-level design discussion was held during the sprint planning. For simple tickets, the SWE assigned took the responsibility of finalizing the design. For more complex tickets, discussions were held with the appropriate team members. In some cases, a wiki page with suggested alternatives was requested before such discussions.

Including security as a part of the design consideration presented the following challenges.

Challenges regarding security knowledge of SWEs. During the design stage the main focus was to achieve functional correctness and performance considerations when applicable. When the features dealt with sensitive information, security became a necessity. Yet, secure design practices were not always the focus and instead assumed protection through "security functions" such as authentication and authorization. For example, one SWE when asked if the input attributes should be validated:

> *"I'm not sure I'd worry too much about that. This form is authorized for admin only, so customers won't be changing this attribute themselves. Trying to validate that they've provided a valid <redacted> attribute feels kind of complicated..."*

Secure design must determine relevant threats (through threat modeling) and consider all aspects of the software, but it is a tall order to require all SWEs to have such knowledge and skills.

Challenges in understanding contextual knowledge by a security expert. One of the software engineers in the team had a significant background in security (called SecSWE hereafter). He was able to identify the relevant security risks and propose secure design solutions. However, although he had been working with the team for more than 1.5 years, the knowledge of the minutiae of how the product operated was lacking and the proposed solutions were not always directly applicable for the product at hand. Since there was no one-size-fits-all solution to security problems, secure design required in-depth knowledge of both security and the product.

3.2.2 Implementation

SWEs picked up tickets from the sprint plan to implement. Again, the first priority of SWEs was to implement the functional requirements of the tickets. We observed the following secure development challenges in this stage.

Challenges regarding security knowledge in SWEs. Even with security considerations in the design phase, the actual implementation of code could expose vulnerabilities if the SWE was not capable of defensive programming and unaware of secure development practices. The main challenge is that the SWE needs to be able to identify the potential security risks in the code that he/she is writing. Other factors such as reliance on frameworks, or incorrect use of frameworks or APIs can also lead to insecure implementation. Such lack of knowledge in SWEs cannot be simply compensated by the presence of a security expert in the team. Usually in such cases the identification of security issues shifts further down the development process during code analysis or security testing and presents additional challenges – the issue might be missed altogether, fixing the security could require significant code changes or even design changes, or there might not be enough time in the sprint to fix the issues which might lead to the ticket being excluded from the sprint.

Challenges in applying security knowledge in practice. In certain cases SWEs were able to identify potential security risks and also had the necessary knowledge to resolve them, but chose not to do so. We concluded this based on our data where we observed that during discussions regarding security issues, often times some SWEs were able to propose the solutions, but did not apply them in practice. This could be attributed to multiple reasons such as lack of time, reliance on security functions (such as authentication and authorization), or security of code considered "invisible" to the customer compared to the feature itself.

3.2.3 Continuous Integrations/Code Analysis

Once the implementation was complete, the source code was pushed to the remote repository where automated builds were carried out by the continuous integration (CI) pipelines. These pipelines executed unit/integration tests and code analysis tools such as SonarQube [37] (later Black Duck [44]) on the feature branch.

Challenges on fully utilizing available tools. We found that the available tools were not utilized to their full capabilities. SonarQube was not maintained and mostly only relied upon for simply lint and code quality checks. The team was asked to set up Black Duck, a tool that analyzes the use of third-party open-source libraries in the codebase and provides information on licenses and known security vulnerabilities, into the CI pipeline as a part of the secure development effort. During discussions on the initial results of the scan, one SWE remarked: *"We use quite a few out-dated packages. I would be surprised if this tool didn't report any issues."* Black Duck was setup on management's request, and the scans were initially enabled by default in the CI pipeline. But the resulting build failures in the Black Duck stage prevented SWEs from merging in code. The tool was then disabled by default and a separate ticket was filed to track and address the vulnerabilities discovered.

3.2.4 Developer Testing

Before a ticket was assigned for code review, SWEs made sure that all automated tests were passing, deployed the updated product on a test environment, and performed their own testing. These tests were usually targeted towards functionality rather than security. Once functionality was verified, the ticket was updated with the steps to replicate the test plan and assigned for code review with the creation of a pull request.

3.2.5 Code Review

Two SWEs were assigned for code review. Usually one SWE provided thorough review while the other would just sanity-check the code. Depending on the complexity of the feature, the reviewers may perform quick functionality tests on top of going through the code changes. Any missing pieces, mistakes, inconsistency or departure from existing best practices in the coding pattern were set up as tasks to be addressed before the ticket was marked as "done." We observed the following challenges in this stage.

Challenges in consistently performing code review. Occasionally, when the changes were required urgently, code review was essentially skipped with the SWE just describing the changes made to others and asking if anyone objected to the approach. This could lead to potentially identifiable issues propagating to the production code base.

Challenges in thoroughly performing security review. Although SWEs provided good feedback during code review, the suggested changes were based on internal best practices and patterns followed in other similar modules in the product. However, a thorough security code review requires more in-depth security knowledge and experience which was lacking in the SWEs. SecSWE however was able to provide specific security-related feedback. A potential API misuse of a crypto-library (bouncycastle [39]) was identified by SecSWE during code review. While addressing this comment, it was discovered that the API misuse could have caused memory leaks leading to out of memory conditions.

3.2.6 Post Development Testing

The QAE, who was hired during our fieldwork, prepared thorough test plans for each ticket in the sprint and carried them out on the test build. We observed the following challenges at this stage.

Challenges in acquiring contextual knowledge by QAE. Although the QAE had years of prior experience, he was new to the team and the products, and hence required assistance to set up test environments and understand the specifics of the product before he could create strategic test plans. The QAE also had a security background and showed interest to learn and practice security-oriented testing along with SecSWE. But he expressed lack of time and in-depth knowledge about the product as reasons not to do so at that time.

3.3 Product Release

At the end of a sprint, a build of the product including all implementations in the sprint was deployed within the company for up to a week; then release notes were written and the product was released. The customers were required to opt-in for the updates, after which the support team executed the remote update procedures.

4 A Shift in Secure Development Practice

Shortly after the researcher joined, one SWE from each product team was assigned to be a member of a "virtual" application security engineering team and tasked to help drive security improvements for the product. This was part of the secure software development lifecycle (S-SDLC) effort that was kicked off before we joined. The designated SWE performed security-related tasks in addition to the normal sprint work. SecSWE was assigned this role for his team.

4.1 Little Impact at First

During the first three months of the fieldwork, the only security-related work fell into two new categories of tickets created as part of the S-SDLC efforts.

- *CSF* tickets: security-related tasks guided by the NIST Cybersecurity Framework (CSF) [38].
- *ASVS* tickets: compliance with OWASP Application Security Verification Standard (ASVS) framework [40], for web facing application components.

These tickets were not included in the sprint plan. SecSWE and another developer (SWE1) were tasked to work on these tickets alongside the sprint work. Both SecSWE and SWE1 worked on these tickets individually, and the only updates about this work was provided briefly during morning scrum. These tickets were referred to as "burning cycles" and often the updates on these efforts carried little information:

- *"I knocked off a couple of CSF tickets."*
- *"I talked with <management personnel> about some CSF work and what is expected."*
- *"I will be catching up on some neglected CSF work and write up some wikis."*
- *"My changes are in PR. I will next work on ASVS tickets while I wait for reviews."*
- *"I am working on a P2 ticket and also doing some ASVS audits."*

When talking to SecSWE on how the security work is going, he remarked:

> *"I don't know. It takes a lot of work for this ASVS stuff, looking at all the code, testing, researching... I feel like we are putting all of this effort and time on this but nothing is being done about it you know."*

Although significant effort was put on resolving the CSF and ASVS tickets, we did not observe any impact on the development workflow as a whole.

4.2 Making Progress

During the third month of the fieldwork, SecSWE started to work on threat models for both products. He first shared the initial threat model for P1 with the team for feedback which garnered greater visibility on the security work in the development team as a whole and initiated discussions on the communication patterns between the different microservices in the product. SecSWE also documented the security issues in order to facilitate the pending discussions for the threat modeling work.

Prior to the threat modeling work, two security tickets had also been logged: 1) The researcher discovered that the same key pair was reused for all customers when P1 was setup as a high availability (HA) pair. 2) On further investigation, SecSWE discovered another instance of key reuse problem in establishing connections to the cloud server. The threat modeling work also initiated discussions and feedback from other SWEs concerning these issues.

Another key mismanagement issue was discovered where a private key was exposed in a publicly accessible server. The initial response from other SWEs was that this server, while Internet facing, was not advertised to the public as it was mainly used to distribute software updates. Discussions on the potential misuse cases of this issue in particular garnered positive interest in security work with the lead SWE remarking: *"I am excited about the work SecSWE is doing."*

Security Scrum Poker. With several security tickets logged, SecSWE suggested to have a meeting specifically to discuss these tickets before the next sprint. Prior to the meeting, everyone was asked to review the tickets and corresponding wiki pages for discussion. SecSWE also introduced the DREAD risk assessment scheme [21] and the security scrum poker (akin to scrum poker) in order to assess the estimated risk of the discovered vulnerabilities. The goal was for the entire team to converge on a risk score for each ticket, discuss the rationale behind the scores in case of mismatch to clarify everyone's understanding of the issue, and ultimately use the risk scores to prioritize the security tickets.

4.2.1 Putting Security into Development Context Made Security into Development Practice

Three security scrum pokers were held during the fieldwork. In the first meeting, two SWEs tended to score lower than the others. As with scrum poker, in case of mismatch the SWEs were asked to explain the rationale behind their scores. This brought forward any misunderstanding of the discussed issue and allowed the group to clarify them. After a couple of iterations, one of the SWEs kept having varying scores and tried to move on to another ticket by agreeing with the others' scores but the lead SWE remarked: *"You cannot just do that. Either you have to defend the score or tell us why you changed your mind."* The whole team agreed that the meeting was very fruitful in clarifying their understanding of the issues and/or the proposed solutions with the SWEs remarking:

- *"That was more productive than I expected."*
- *"I really liked this session and the discussions cleared things up. I am excited to see where this effort leads."*

These discussions led to contextual analysis of the discovered issues (what is the risk in the system?). They helped uncover root causes of existing issues and bring forward discussion on potential solutions, trade-offs for alternatives, and potential road-blocks in implementing them. Importantly, **these discussions were useful to SecSWE as well.**

The discussions between SecSWE and the lead SWE led to the understanding of how and why the private key ended up in the public-facing server in the first place – it turned out that previously P1 was distributed to the customers using Preboot Execution Environment (PXE) boot over the network. Although this method had not been used for several years, it was still used internally to quickly deploy test environments. As setting up internal test environments did not require the PXE boot kickstarter script to be on a public facing server, it was subsequently moved to an internal server during the fieldwork. This task required collaboration between SecSWE, lead SWEs, as well as the networking engineers to implement, test, and deploy. For the cases of reused keys, short-term solutions of limiting users to only required commands while restricting shell access altogether were proposed. A longer-term goal to set up a per-deployment key management and distribution mechanism was also discussed. During the fieldwork only the task to research the approach was created.

After the first security scrum poker, SecSWE asked others to also report any security issues they found. During the course of the fieldwork 15 security tickets were created that were not related to ASVS or CSF. The following are the categories of vulnerabilities discovered during the fieldwork.

- Mismanagement of cryptographic keys and certificates.
- Lack of access control to remote assets
- Improper handling of passwords
- Unencrypted application update channel
- Remote code execution
- Cross-site scripting (XSS)
- Privilege escalation
- SQL and command injection
- Misconfigured SAML (Security Assertion Markup Language) authentication

These issues were discussed in at least one security scrum poker meeting. SecSWE and the researcher also developed proof-of-concept (PoC) attacks for application-level vulnerabilities such as remote code execution, XSS, Privilege escalation, and SQL and command injection which helped drive further discussions. Out of the 15 security tickets identified, 8 were approved for development after going through both the security scrum poker and the prioritization stages. Six of the approved tickets were included in a sprint plan. The researcher asked for SecSWE's opinion on the increased focus on security. The response was:

> *"I am surprised by the increased focus on security as well. They were not at all interested in these stuff before... I had already reported some of these issues before, although I didn't have time to make PoCs for it. But it's good that we have some attention now."*

4.3 Challenges in Security Ticket Prioritization

Although work was done to identify security issues, getting them prioritized for development still presented challenges.

Security tickets were not considered "real." Purely addressing existing security issues or improving security in existing code/infrastructure was not considered as "real." In one sprint planning meeting after a few security tickets were discussed and included in the sprint, the lead SWE remarked: *"Okay now let's include some real tickets in here as well."* The basis for this point of view seemed to be that security improvements made to existing features or to the infrastructure were not visible to the customers.

Security tickets had higher story points. Many security tickets were voted to have high story points and hence would not leave room to include other feature-driven tickets. The reasons for higher scores include:

- *Technical challenges:* Security tickets required more research and experimentation to figure out the most suitable solution for the product.
- *Dependencies:* Fixing existing security issues required identifying all use cases of the vulnerable feature and the impacts of the changes on the product. Finding dependencies itself was time consuming as documentations may be outdated, and additional developer and QAE testing would be needed.
- *Implied changes in processes:* SWE/support/QAE may be relying on the vulnerable features and may not want to change. SWE may need to provide viable alternatives. *"Before we move on with the fix, we need to first find out if there are undocumented use cases of these things. This is not uncommon with the support team to have some automated scripts which might rely on some access or some feature and we do not want to break them."* This could lead to additional work.

Legacy systems. Older systems already deployed at customer sites may still need to be supported. In such cases, alternative solutions needed to be provided or both new and old systems needed to be supported. Some security holes may be impossible to resolve because of initial bad design. One ticket was blocked due to this very reason as around 20 customer sites were yet to be migrated to the updated system.

Meeting Customer Requirements. Customers were unwilling to allow change of existing features. During a discussion for changing the rule specification UI, which introduced command injection vulnerability, one SWE mentioned that they had already tried to remove that feature before as the product already had an updated alternative built in. But the customers were unwilling to migrate to the new feature as it meant that they had to transition all the existing rules to the new format and they were unwilling to do so. SWE said that he already knew what this customer would say:

> *"If there are security issues then that is your problem and you need to fix it without taking away my features."*

New customer requests. During the course of a sprint, new high-priority customer tickets may be received. In such cases the security tickets would be de-prioritized, as happened to two security tickets included in the sprint plan.

4.4 Security-aware SWEs

After the introduction of security scrum pokers, there was an increase in security-related discussions outside the meetings as well. These ranged from humorous comments – *"SecSWE is not going to be happy if you do that."* or *"He is the security police now!? <laughs>"* to positive reactions for including security tickets during prioritization meetings: *"SecSWE and <the researcher> are pretty good with security."*

Security considerations in other tickets In addition to the security tickets, security considerations were made in three other feature tickets.

1. *User-side error reporting for failed certification validation.* The researcher was assigned this ticket which led to a major refactoring of the code and use of an updated single library for performing uniform certificate validation throughout P1.
2. *Enabling use of new certificate for SAML authentication without requiring application restart.* A certificate reuse misconfiguration was discvoered while working on this ticket. Code was refactored to allow proper configuration changes.
3. *Sending real-time alerts to customers.* As part of the ticket access control on cloud server was tightened to disable shell access.

Potential security issues identified. Security issues were also brought up and discussed by other SWEs.

1. An SWE discussed potential XSS vulnerabilities in another team's application while working with them, and advocated for the other team to consider upgrading a programming framework to the latest stable version.
2. Input validation was added in multiple modules proactively by SWEs working on a ticket with UI changes. Often they asked (in person or over slack) if validation code was already implemented in the module or where to look for reference validation code. In cases where validation was complicated lead SWEs proposed how the validation could be done.

> *"Do we have any input validation code that is used both by <microservice1> and <microservice2>? If so, do you remember where it is located?"*

> *"... I know that it's not a priority for management to validate input that is supposed to be entered only by support, but it doesn't cost much."*

Security considerations in design. A feature requested by a high-priority customer required the ability to access internal configuration options otherwise hidden behind the application for an unorthodox use case of product P1. The initial design for the feature had not considered security risks with the assumption that this feature would only be accessible by the administrative account, which belongs to the support team. SecSWE pointed out that such design could potentially expose command injection and privilege escalation vulnerabilities and started a discussion on the feature, which led to the finding that the original design had overestimated the access requirements to implement the desired functionality. The initial design was then shelved with a follow up design discussion scheduled to allow time to gather information for a more secure approach.

4.4.1 What was Driving the Change

On analysis of the fieldnote data, we find that the positive shift in the development team's security awareness can be attributed to the software engineers being able to identify the applicability of the security knowledge within the context of the everyday work they performed. We observe that by working along with others in the team to apply security knowledge under the concrete context of the software products, the software engineers became attentive to security risks when similar situations were encountered later. When a considerable number of discussions had taken place on a security-related topic, the group ended up with an agreed-upon set of knowledge and the associated set of practices became the "preferred practice" for dealing with this security concern. At this stage, considering this specific aspect of code security became the group's "habit." Later if some SWE in the team needed to work on a relevant part of the code but lacked this specific piece of security knowledge, they would seek guidance from others in the team, in the same manner as they would with other types of development tasks. They then learned and executed the preferred practice of the group.

Our analysis of data shows that what was driving the positive change was the learning dynamics existent in the development team. The initial lack of visible impact from management pushing for adopting S-SDLC was because the CSF and ASVS tickets were detached from the SWEs' regular work, and thus the relevant security knowledge did not have much opportunity to be directly applied in their work. It turned out that application of knowledge was the key driver for learning in an environment like a development team. Later on when SecSWE started to use security scrum poker in the threat modeling work, and involved all SWEs in the discussions, the security knowledge became concretized and contextualized. This drove a learning cycle within the team that allowed the SWEs to start obtaining relevant security knowledge and become more security aware.

We find that understanding the learning dynamics in the development team is crucial to effectively push for secure development practices. In fact, making developers more security aware is no different than cultivating their knowledge in any other aspect of software development. **Our data indicate that, to establish a security culture in a development team, it might be helpful to follow the same learning dynamics that drive how culture forms for that community.**

5 Learning in a Development Team

The analysis of our fieldnote data yielded a model that explains the establishment and evolution of preferred practices in a development team and hence the progression of its culture. In summary, the development team is a situated learning [20] environment where the process of learning drives the creation and evolution of preferred practices. When SWEs needed assistance, they acquired the necessary knowledge from the team. As they performed their task and applied the knowledge in practice, it provided the necessary platform to further drive the process of learning and started to make contributions to the group. This process iterated over many cycles, until the group reached a point of saturation where the knowledge developed within the team was sufficient to facilitate progression in the task at hand. When this process was applied in practice, it not only led to professional growth of the SWE but also served as validation for the knowledge which then became a part of the current culture of practice.

5.1 Subject Matter Experts (SMEs)

As is common in software industry these days, the products the company built were vast entities and no single SWE knew the details of *all* aspects of a product. Multiple dimensions of knowledge were required within the development team in order to build the software, and the in-depth knowledge of each dimension was scattered between different SWEs in the team. An SWE can be the subject matter expert (SME) for some dimensions while at the same time being a novice in others.

When an SWE had the most in-depth knowledge on a topic within a development team, they were often called a subject matter expert (SME) of that particular dimension of knowledge. Although everyone in the team may have a good understanding on the topic, the SME was the one who understood the underlying details of the implementation. When an SWE started to work on a task new to them, they first went (or were directed to go) to the SME on the team. This created an implicit hierarchy within the team based on the dimension of knowledge under consideration, which facilitated the flow of knowledge within the team. This hierarchy transcended job titles. For example, despite holding a junior position in the company, a new hire who had worked on a task could immediately become the SME on certain pieces of knowledge associated with the task, and any future queries related to these pieces would first be directed towards them.

We observe the existence of SMEs throughout our data. When trying to set up a test environment for a new router device, an SWE asked the group: *"I have read through the documentation but I still cannot get it to work in our test environment. Can anyone help me out?"* He was directed to one of the network engineers: *"Normally, I just go and ask <network engineer>. I do that even before going through the documentation. 99% of the time, he knows what to do and I trust him."*

When trying to get access to a development infrastructure, the researcher asked the lead SWE: *"I need to access the CA server to test this feature. How do I get access?"* Lead SWE: *"You should go ask SWE1. He just cleaned up the access list for the CSF thing."*

When the lead SWE was asked the details of an existing script: *"Full disclosure, I have no idea how that script works. <Former employee> implemented it and no one has had to make changes till now. But <support engineer> should provide you more information. They are the ones who use it."* In this case, although the SME is no longer within the company, the workflow established through the use of the automation script still remained and the next most knowledgeable person took responsibility of it.

Our data shows that the roles of SMEs, the knowledge on each dimension, and the preferred set of practices were not static but were developed and evolved within the development environment. When there were multiple potential SMEs on a topic, the responsibility could be passed on to the others as well. In some cases this also led to more official transfers of duties within the team. For example, when dealing with customer issues, the lead SWE was pulled into multiple meetings between the customer support team and the clients. Overwhelmed by the work, the team internally discussed the possibility of having another SWE who was working on the problem module for the past months to take over some of the client discussions, with some guidance from the lead SWE. After reaching an agreement, this was then communicated to the management for future meetings.

5.2 Establishment of Preferred Practices

The development team tended to have established preferences for activities that were carried out repeatedly. We observed team preferences for coding styles, debugging techniques, code reviews, ways of dealing with the IT department, use of scripts/tools for tasks, etc. We also observe that these preferred practices were usually tried and tested approaches of doing things within the team and were communicated to other SWEs in the team as needed. For example, preferred coding styles were communicated through the code itself while any unwarranted deviations were communicated through code reviews and reverted back to the preferred way. Any changes made to improve the existing style were also communicated through code and code review. Other preferences could be communicated mainly through discussions between the SWEs whether in a one-on-one or group setting. Usually an SWE sought help from the group using language like *"Got a second for a rubber ducky?"*, *"Can I borrow some of your time <SWE>?"* SWEs were encouraged to hold these discussions in the group chat as there could be more *"eyes"* on the problem and the solutions could be reached more quickly. These discussions also allowed for the preferred practices to evolve and improve as issues or better options were identified.

These preferred practices became a part of the group knowledge and tended to stick through generations of employees. In such cases some of the in-depth knowledge might be lost with the employee leaving but the preferred practices continued.

- *"That is a script that <former employee> developed and we still use it."*
- *"That playbook was written by a <former employee>. I know what it does but I am not sure if it uses this script internally. I would have to go read through the code but it gets the job done."*

5.3 A Situated Learning Environment

Through analysis of the field notes we find that the roles of "SME" and "learner", assumed by different SWEs for different dimensions of knowledge, drove a learning cycle within the team. This interactive activity of learning was the core process through which preferred practices were established within the team.

The pattern of learning observed here is not new. The concept of learning, not through a teacher/learner dyad, but as a situated activity where a learner not only acquires knowledge from the experts ("old-timers") and their peers but does so while participating and contributing in a community of practice is referred to as situated learning [20, 26, 32]. Learning, in this view, is not simply a process of transfer or assimilation of knowledge from the expert (SME) to learners (SWEs), but rather a generative process where each "reproduction cycle" from "learner" to "old-timer" leaves a trace in the community of practice, in both its social structure and physical, linguistic and symbolic artifacts.

The development team is a dynamic situated learning environment with a wide range of knowledge to be acquired and mastered. Based on the dimensions of knowledge under consideration, SWEs simultaneously perform multiple roles of learning practitioner, aspiring expert, status subordinate, or sole responsible agent [20]. The everyday activity of software development provided situated opportunities to learn, defining the "learning curriculum" for the task that SWEs were performing. As an SWE sought to learn from the team, different SWEs enacted different roles to drive a learning cycle to reproduce the existing culture of practice.

- *"Are you guys available for a zoom to discuss the DNS cache changes for the data viz stuff?"*
- *"Alright type gurus. I'm trying to make an interface that is a Map between two sets of constants. I'm not allowed to do what I posted above. Suggestions? . . . "*

Contradictions also arise as a part of this interactive social process as learners start to contribute. Working on resolutions to these contradictions leads to a renewed practice in the community, i.e., preferred practices are established and evolved as SWEs go through the learning cycle.

In this vein, **creating a secure development culture is the process of making secure coding practices into the preferred practices of the development team.** Thus, facilitating situated learning regarding security within the development team, is key.

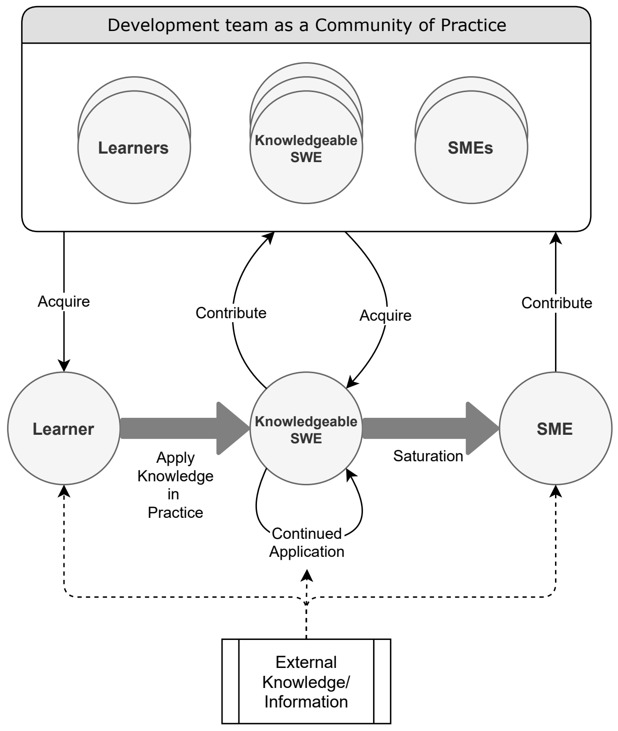

Figure 1: The Learning Cycle

5.4 The Learning Cycle

Figure 1 shows the interactions of an SWE with the development team as he/she progressed from the role of a learner to an SME. At any given time, an SWE could assume different roles for different dimensions of knowledge. For example, the SecSWE could be an SME on certain secure coding practices, but at the same time a learner on some technical details about a particular aspect of the product. External resources were accessible at anytime throughout the process of learning. We first describe the different roles an SWE could assume in this learning cycle.

- *Learner*: An SWE started out as a learner acquiring knowledge, the preferred set of practices, from the team. Learners also looked to external resources on their own, especially for a completely new aspect on which there was no existing knowledge in the team yet. Such acquisition of knowledge only made a difference in the team's practice when the SWE *applied the knowledge* in the practice, whereby he/she started to contribute to the team's knowledge and progressed in the path of professional growth.
- *Knowledgeable SWE*: The application of acquired knowledge in the context of daily practice by the SWE served an important purpose – it provided the basis to have contextual discussions whereby the SWE was able to make contributions to the group. The resulting iterations of the interactive learning process led to a convergence in understanding of the knowledge within the group, and thereby the establishment of preferred practices.
- *SME*: When the learning cycle reached saturation and no new contributions were made to the community of practice, the SWE was able to assume the role of SME. In terms of legitimate peripheral participation [20], the SWE had reached full participation for that particular dimension of knowledge space. The learning cycle then continued for that dimension with other SWEs filling in the role of learner and knowledgeable SWE.

We find that an effective learning cycle went through the following stages to create, maintain, and grow the preferred practices through multiple generations of employees.

Acquisition The most accessible and credible source for a learner was the SME on the topic of interest. They provided access to the current culture of practice to the new learner. The level of knowledge available in the team varied depending on factors such as education, prior experience, applicability in the daily work, and so on. When the knowledge within the team was sufficient, the learning cycle simply reinforced the current preferred practices, as new learners continued buying into it. In case of insufficient expertise within the team, a new knowledge requirement was created which led to individual/group research on the topic through external resources. This could also be facilitated by a new member joining the team who possessed the lacking knowledge.
Security Implication: the expertise levels of the SMEs on security *within the team* determine the team's preferred practice in secure coding.

Application Acquired knowledge needed to be applied in daily practice to drive the learning process. This was a critical step in the learning cycle; without application in daily work, the knowledge was limited to the individual SWE and never became a part of the preferred practice. On the other hand, applicability led to both individual and team growth as the applied knowledge was immediately shared to the peers through development activities like scrum meetings, design/implementation discussions, code review, testing, documentation, and so on. This provided two important driving forces that helped propagate the learning cycle: 1) a shared motivation to solve problems, and 2) the shared context of the work practices which everyone was aware of. These facilitated bi-directional discussions as opposed to a teacher-student scenario as is often perceived as how transfer of knowledge happens.
Security Implication: SWEs' security knowledge, like all other knowledge, needs to be grown with application. This works well for security, since the best time to apply security knowledge is when the code is being written (as opposed to applying security knowledge to fix vulnerabilities later on).

Contributions As SWEs put knowledge into practice, they were able to contribute to the group based on their experience and findings from daily practice. This knowledge exchange through application led to the growth and evolution of the whole team with the increase in existing knowledge on the topic. When there was an established/preferred/agreed upon knowledge base on that given topic, the knowledge in the group reached a level of saturation, and it became a part of the preferred practices.

Security Implication: When security becomes part of the preferred practice, all SWEs in the team will be security aware while writing new code. We observe that the first successful step towards implementing an effective S-SDLC and creating a security culture in the development team was the rise of security-aware SWEs. After all, if the SWEs are capable of writing secure code, it will make a real change in the final products' security. As was pointed out in prior literatures, fixing security bugs retroactively is costly and often encounters resistance from the development team [18, 25]. Companies would be better off to prevent, as much as possible, security vulnerabilities from being introduced in the first place.

6 Revisiting the Shift towards Security

We now identify the key enablers of the positive shift in secure development we observed during the fieldwork.

6.1 Setting Security as a Goal

Past experience suggested that management support was an important factor in the successful implementation of S-SDLC [19]. Our observations supported this. Due to cost in terms of time and efforts required, security was easily perceived as an "obstruction" to the daily practice of SWEs and hence the learning cycle for this dimension of knowledge did not evolve at first. We found that management played an important role to set security as a goal, making it a part of the deliverable. Doing so ensures that security knowledge is not something that overwhelms SWEs but simply applicable to daily practice, eliminating a critical barrier to drive the learning cycle.

6.2 Applying Security Knowledge in Context

Having the management directive and support for secure coding was necessary but not sufficient to eliminate the barrier to adopting secure development practices. Secure coding requires a wide range of security knowledge, and providing adequate education and awareness was pointed out as one major challenge in successfully implementing S-SDLC [19]. While the company provided SWEs virtual training for secure coding and there were also various guidelines and wiki pages the SWEs could access, applying the acquired knowledge in everyday work required expertise in both security and the contextual knowledge of the existing code base. Finding this connection was challenging for SWEs. Without application, the knowledge gained from training was at best internalized by individual SWEs, but remained detached from their daily practice. The SWEs effectively considered security-related tasks as secondary tasks, separate from their primary practice. To overcome this, a bottom up support was also needed to make real progress. In our fieldwork we found that such bottom up support happened through the learning cycle identified in the previous section. The threat modeling and associated security scrum poker meetings, which involved all SWEs, provided the opportunity for the SecSWE to put the relevant security knowledge into the concrete context of the software being built. This started the learning dynamics that enabled all SWEs to progress on the "security dimension."

6.3 The Role of Security Advocates

The work of SecSWE played an important role in facilitating the learning cycle and making security into part of the development team's preferred practices. SecSWE was a "security advocate" [15] even before the management pushed to implement S-SDLC. He worked in the development team, and was also assigned to be a part of the virtual security team, providing additional security resources. Analyzing our data, we find that this structure added more value to security advocacy, making other SWEs more receptive to his advice as they started to consider it "part of his job." Working on the same team provided an important factor in demonstrating the applicability of the security knowledge in the context of the daily practice. This facilitated SecSWE to contribute knowledge as applicable to daily practice, helping to drive a productive learning cycle, which was beneficial to both the rest of the team and SecSWE himself. Through this interactive learning process, SecSWE was able to better understand the necessary details of the product which allowed him to apply his security knowledge in a more context-aware manner. Further iterations of this learning cycle led to more security-aware SWEs in the team.

7 Limitations

Our work is limited by a few factors. First, our findings are based on the fieldwork data collected by a single researcher. Although the researcher had prior training and experience in conducting participant observation research, the collected data are shaped by the researcher's positionality (his age, gender, position in the company, and so forth). For example, the researcher did not have as many interactions with customer service and upper management because of his position in the company. However, the researcher did build an overall understanding of the company during the research, and the results were extensively discussed with the broader research group during analysis to better account for any inherent biases in the data. Second, our findings are based on the observations of a single company with a particular size and structure. Although we believe the development team is representative of one in a mid-sized software development company, the specific challenges of adopting secure development practices and how they were/were not overcome may not be directly applicable and generalizable to every company. As such, the model of how a culture is developed within a software development team might not be comprehensive. Nevertheless, during data analysis, the team paid particular attention to how results related to common problems faced in security and software development to ensure that the findings could be relevant to other companies.

8 Recommendations for Companies

Our findings suggest a potentially useful strategy for a small to medium sized company. Having a security expert as a part of the development team, participating and advocating for security at every stage of the development process, is beneficial in starting a security culture. This not only helps cultivate security-aware developers, but also helps the security expert identify security issues and collectively converge to secure practices that are best suited for the project at hand. Development of the relevant security knowledge in conjunction with the regular software development skills promotes secure coding practices which, overtime, become a part of the team culture. Our research also observed the effect management had in facilitating the positive shift. Even though the initial efforts focused on the compliance tickets were not effective, the fact that management made security an explicit goal provided the opportunity for the security advocates to experiment different strategies that eventually led to positive results.

9 Related Work

Our fieldwork was conducted in the backdrop of the company starting to implement a secure development lifecycle, a concept first articulated by Howard and Lipner [19]. This seminal work highlighted the importance of education and training in creating S-SDLC. Our findings further indicate that understanding the learning dynamics, in particular how preferred practices are established within a software development team through the situated learning framework, can be instrumental in creating positive changes in secure development.

There is a long line of study on developers' role in software security. Some used psychological techniques [24]. Others used surveys and interviews [3, 5, 13, 22, 28, 35] as well as study of code artifacts [3, 22]. More recently, researchers have used secure coding competitions [27, 31] and controlled experiments [2–4, 11, 23] to study the problem. Our work is unique in that we use long-term participant observation conducted in a real company. The longitudinal study based on real-world observations allows us to obtain deep insights that are otherwise hard to come out through snapshots-in-time study or self-reported data.

Palombo et al. [25] used ethnographic methods to study a software company's secure development processes. The authors indicated that a co-creation model where security experts working inside the development team could produce positive changes in secure development processes. Our work revealed the role of learning dynamics in pushing for positive shift in adopting secure development processes. The role situated learning plays in starting a secure development culture is consistent with the co-creation model.

The SecSWE in our study can be viewed as a "security advocate," which has been extensively discussed in recent studies [14–16]. Our findings on the role of team culture in security awareness of SWEs echoes that from prior studies. Assal and Chiasson [5] explored how security best practices are integrated into the software development lifecycles and found that company culture is an influential factor in adoption of security practices. Haney et al. [17] carried out in-depth interviews to understand cryptographic development and testing practices in organizations and found that rigorous secure development and testing practices are guided by a strong security culture within organizations. They also identify that security experts within the team are critical influences in the security culture of an organization and in supporting less-experienced personnel. Our findings confirm the important role security advocates play in starting a security culture, and further provide guidance on how to make security advocates' work effective, through understanding the underlying learning dynamics that drive the formation of a development team's culture.

There are also past work that examined the effect of learning from experience in software development [9], and work that analyzed open-source software development using the situated learning framework [8]. Our work focuses on secure development, and our research findings are consistent with these earlier works which focused on learning's role in software development in general.

10 Conclusion

We present an ethnographic study of secure development processes in a software company. Our research was able to observe the unfolding of implementing a secure development life cycle in the company. Data analysis shows that a positive shift in developers' security awareness resulted from underlying situated learning dynamics, where security knowledge is constantly applied in the concrete work of the development team. This process drives the establishment of secure coding practices as the preferred practices of the team, essentially establishing a secure development culture. We find that a security expert working within the development team could be instrumental in driving this positive shift.

Acknowledgments

We thank Raj Rajagopalan, John McHugh, and the anonymous reviewers for numerous valuable comments on an earlier version of this paper. We owe gratitude to the company, and its employees who participated in the study. This research is supported by the U.S. National Science Foundation under Grant No. 1801633. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

[1] Y. Acar, S. Fahl, and M. L. Mazurek. You are not your developer, either: A research agenda for usable security and privacy research beyond end users. In *2016 IEEE Cybersecurity Development (SecDev)*, pages 3–8, 2016.

[2] Yasemin Acar, Michael Backes, Sascha Fahl, Simson Garfinkel, Doowon Kim, Michelle L Mazurek, and Christian Stransky. Comparing the usability of cryptographic APIs. In *IEEE Symposium on Security and Privacy*, San Jose, CA, USA, 2017.

[3] Yasemin Acar, Michael Backes, Sascha Fahl, Doowon Kim, Michelle L Mazurek, and Christian Stransky. You get where you're looking for: The impact of information sources on code security. In *IEEE Symposium on Security and Privacy*, San Jose, CA, USA, 2016.

[4] Yasemin Acar, Christian Stransky, Dominik Wermke, Michelle L Mazurek, and Sascha Fahl. Security developer studies with github users: Exploring a convenience sample. In *Thirteenth Symposium on Usable Privacy and Security ({SOUPS} 2017)*, pages 81–95, 2017.

[5] Hala Assal and Sonia Chiasson. Security in the software development lifecycle. In *14th Symposium on Usable Privacy and Security*, Baltimore, MD, USA, 2018.

[6] H Russell Bernard, Amber Wutich, and Gery W Ryan. *Analyzing qualitative data: Systematic approaches*. SAGE publications, 2016.

[7] Kathleen M. DeWalt and Billie R. DeWalt. *Participant Observation: A Guide for Fieldworkers*. Lanham: AltaMira Press, second edition, 2011.

[8] Kasper Edwards. Epistemic communities, situated learning and open source software development, 2001.

[9] Wai Fong Boh, Sandra A Slaughter, and J Alberto Espinosa. Learning from experience in software development: A multilevel analysis. *Management science*, 53(8):1315–1331, 2007.

[10] David Geer. Are companies actually using secure development life cycles? *Computer*, 43(6):12–16, 2010.

[11] Peter Leo Gorski, Luigi Lo Iacono, Dominik Wermke, Christian Stransky, Sebastian Möller, Yasemin Acar, and Sascha Fahl. Developers deserve security warnings, too: On the effect of integrated security advice on cryptographic API misuse. In *14th Symposium on Usable Privacy and Security*, Baltimore, MD, USA, 2018.

[12] M. Green and M. Smith. Developers are not the enemy!: The need for usable security apis. *IEEE Security Privacy*, 14(5):40–46, 2016.

[13] Matthew Green and Matthew Smith. Developers are not the enemy!: The need for usable security APIs. *IEEE Security & Privacy*, 14(5):40–46, 2016.

[14] Julie M. Haney and Wayne Lutters. Security awareness in action: A case study. In *Workshop on Security Information Workers, USENIX Symposium on Usable Privacy and Security*, Santa Clara, CA, USA, 2019.

[15] Julie M. Haney and Wayne G. Lutters. "It's scary...it's confusing...it's dull": How cybersecurity advocates overcome negative perceptions of security. In *14th Symposium on Usable Privacy and Security*, Baltimore, MD, USA, 2018.

[16] Julie M. Haney and Wayne G. Lutters. Motivating cybersecurity advocates: Implications for recruitment and retention. In *Computers and People Research Conference*, Nashville, TN, USA, 2019. Association for Computing Machinery.

[17] Julie M. Haney, Mary Theofanos, Yasemin Acar, and Sandra Spickard Prettyman. "We make it a big deal in the company": Security mindsets in organizations that develop cryptographic products. In *14th Symposium on Usable Privacy and Security*, Baltimore, MD, USA, 2018.

[18] Bill Haskins, Jonette Stecklein, Brandon Dick, Gregory Moroney, Randy Lovell, and James Dabney. 8.4.2 error cost escalation through the project life cycle. *INCOSE International Symposium*, 14:1723–1737, 06 2004.

[19] Michael Howard and Steve Lipner. *The security development Lifecycle*, volume 8. Microsoft Press Redmond, 2006.

[20] Jean Lave, Etienne Wenger, et al. *Situated learning: Legitimate peripheral participation*. Cambridge university press, 1991.

[21] JD Meier. *Improving web application security: threats and countermeasures*. Microsoft press, 2003.

[22] Sarah Nadi, Stefan Krüger, Mira Mezini, and Eric Bodden. Jumping through hoops: Why do java developers struggle with cryptography apis? In *Proceedings of the 38th International Conference on Software Engineering*, ICSE '16, page 935–946, New York, NY, USA, 2016. Association for Computing Machinery.

[23] Alena Naiakshina, Anastasia Danilova, Christian Tiefenau, Marco Herzog, Sergej Dechand, and Matthew Smith. Why do developers get password storage wrong? a qualitative usability study. In *ACM SIGSAC Conference on Computer and Communications Security*, Dallas, Tex, USA, 2017.

[24] Daniela Oliveira, Marissa Rosenthal, Nicole Morin, Kuo-Chuan Yeh, Justin Cappos, and Yanyan Zhuang. It's the psychology stupid: how heuristics explain software vulnerabilities and how priming can illuminate developer's blind spots. In *30th Annual Computer Security Applications Conference*, New Orleans, LA, USA, 2014.

[25] Hernan Palombo, Armin Ziaie Tabari, Daniel Lende, Jay Ligatti, and Xinming Ou. An ethnographic understanding of software (in)security and a co-creation model to improve secure software development. In *Sixteenth Symposium on Usable Privacy and Security (SOUPS 2020)*, pages 205–220. USENIX Association, August 2020.

[26] Barbara Rogoff. Developing understanding of the idea of communities of learners. *Mind, culture, and activity*, 1(4):209–229, 1994.

[27] Andrew Ruef, Michael Hicks, James Parker, Dave Levin, Michelle L Mazurek, and Piotr Mardziel. Build it, Break it, Fix it: Contesting secure development. In *2016 ACM SIGSAC Conference on Computer and Communications Security*, Vienna, Austria, 2016.

[28] Adam Shostack, Matthew Smith, Sam Weber, and Mary Ellen Zurko. Empirical Evaluation of Secure Development Processes (Dagstuhl Seminar 19231). *Dagstuhl Reports*, 9(6):1–25, 2019.

[29] James P. Spradley. *Participant Observation*. Holt, Rinehart, and Winston, 1980.

[30] David R Thomas. A general inductive approach for analyzing qualitative evaluation data. *American journal of evaluation*, 27(2):237–246, 2006.

[31] Daniel Votipka, Kelsey Fulton, James Parker, Matthew Hou, Michelle L. Mazurek, and Michael Hicks. Understanding security mistakes developers make: Qualitative analysis from Build It, Break It, Fix It. In *29th USENIX Security Symposium*, Boston, MA, USA, 2020.

[32] Etienne Wenger. *Communities of practice: Learning, meaning, and identity*. Cambridge university press, 1999.

[33] Glenn Wurster and P. C. van Oorschot. The developer is the enemy. In *Proceedings of the 2008 New Security Paradigms Workshop*, NSPW '08, page 89–97, New York, NY, USA, 2008. Association for Computing Machinery.

[34] Shundan Xiao, Jim Witschey, and Emerson Murphy-Hill. Social influences on secure development tool adoption: Why security tools spread. In *Proceedings of the 17th ACM Conference on Computer Supported Cooperative Work & Social Computing*, CSCW '14, page 1095–1106, New York, NY, USA, 2014. Association for Computing Machinery.

[35] J. Xie, H. R. Lipford, and B. Chu. Why do programmers make security errors? In *2011 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 161–164, 2011.

[36] Build security in maturity model | BISIMM. `https://www.bsimm.com/`.

[37] Code quality and security | SonarQube. `https://www.sonarqube.org/`.

[38] Framework for improving critical infrastructure cybersecurity. `https://nvlpubs.nist.gov/nistpubs/CSWP/NIST.CSWP.04162018.pdf`.

[39] The legion of the bouncy castle. `https://www.bouncycastle.org/`.

[40] OWASP ASVS. `https://owasp.org/www-project-application-security-verification-standard/`.

[41] Planning poker - wikipedia. `https://en.wikipedia.org/wiki/Planning_poker`.

[42] The security mindset. `https://www.schneier.com/blog/archives/2008/03/the_security_mi_1.html`.

[43] Software assurance maturity model (SAMM). `https://www.opensamm.org/`.

[44] Software composition analysis | Black Duck Software. `https://www.blackducksoftware.com/`.

A Appendix: Codebook

Coding of the fieldnote data followed the general guidelines of inductive approach [30] and grounded theory [6]. It was an iterative process and started after the first three months of the field work. The research team held weekly meetings to discuss and reflect on the data collected. Themes and patterns emerged from those discussions and various codes were used to tag the content in the raw fieldnote. Coding was done by the embedded researcher only, to protect the privacy of participants. Below is the list of codes used.

- Bug discovery
- Bug discovery:internal
- Communication issue
- Compliance:asvs
- Compliance:csf
- Compliance:csf:thirdparty
- Compliance:encryption
- Compliance:phishing
- Cross product issue
- Customer pressure
- Feature pressure
- Forgotten issue
- Ignored issue
- Infra
- Infra:legacy
- Infra:security
- Learn
- Learn:best practice
- Learn:figure out
- Learn:peer programming
- Learn:review
- Policy change
- Preferred practice:code
- Preferred practice:support
- Preferred practice:workflow
- Remote work issues
- SME
- SME:handover
- SME:new
- Secure development
- Security-aware
- Threat modeling
- Threat modeling:dread
- Training
- Workflow change