Social Influences on Secure Development Tool Adoption: Why Security Tools Spread

Shundan Xiao, Jim Witschey, and Emerson Murphy-Hill
North Carolina State University
890 Oval Drive, Raleigh, NC 27695
{sxiao,jwshephe}@ncsu.edu, emerson@csc.ncsu.edu

ABSTRACT
Security tools can help developers build more secure software systems by helping developers detect or fix security vulnerabilities in source code. However, developers do not always use these tools. In this paper, we investigate a number of social factors that impact developers' adoption decisions, based on a multidisciplinary field of research called diffusion of innovations. We conducted 42 one-on-one interviews with professional software developers, and our results suggest a number of ways in which security tool adoption depends on developers' social environments and on the channels through which information about tools is communicated. For example, some participants trusted developers with strong reputations on the Internet as much as they trust their colleagues for information about security tools.

Author Keywords
adoption; security tools; social factors

ACM Classification Keywords
D.2.0 SOFTWARE ENGINEERING: Tools

General Terms
Human Factors; Security

INTRODUCTION
Software security is a non-functional requirement of software that ensures that software functions correctly even under malicious attack [18]. Many types of software vulnerabilities can compromise security, including memory safety violations, input validation errors, and race conditions. According to the National Institute of Standards and Technology, the number of new software vulnerabilities discovered each year has more than doubled in the last decade; in 2012, more than 5,000 new vulnerabilities were reported[1].

The cost of fixing defects in software, including vulnerabilities, increases over time [4]. Vulnerabilities can be particularly expensive, taking into account the costs of losing customer trust, handling bad publicity, and, potentially, facing litigation. For instance, a single vulnerability in Microsoft's Internet Information Server cost an estimated $2 billion in repair, lost productivity, and support [7]. In comparison, previous work indicates that fixing vulnerabilities can take as little as three minutes of a developer's time if found early enough [6]. However, finding these vulnerabilities in order to fix them can require significant skill and resources.

Because the cost of changing software to fix vulnerabilities increases over time [4], we focus on tools that help developers find and fix vulnerabilities in source code during the implementation phase of the software development lifecycle. We call these tools secure-software development tools, or *security tools* for short. We consider two types of security tools: static analysis tools and dynamic analysis tools. Static analysis tools, such as Fortify SCA[2], Armorize CodeSecure[3], and FindBugs[4], scan application source code for vulnerabilities. Dynamic analysis tools, such as HP WebInspect[5] and IBM AppScan[6], scan running applications for such vulnerabilities as memory leaks and SQL injections. Such tools can augment the security practices, if any, that a development team already uses.

When developers do not use security tools, they may deploy software containing vulnerabilities they could have prevented by using these tools during implementation. We previously explored the problem of software development tool underuse for tools available in integrated development environments (IDEs) [22]. However, the consequences of security tool underuse — such as the release of needlessly vulnerable software — are more severe than those of, for example, refactoring tool underuse. In addition, understanding the existing culture surrounding security tool use is critical if we are to create a culture of secure development. Thus, security tool underuse deserves special attention. We will refer to it as the *security tool adoption problem*.

[1] http://web.nvd.nist.gov/view/vuln/statistics

[2] http://www.hpenterprisesecurity.com/products/hp-fortify-software-security-center/hp-fortify-static-code-analyzer
[3] http://www.armorize.com/codesecure/
[4] http://findbugs.sourceforge.net/
[5] http://www.hpenterprisesecurity.com/products/hp-fortify-software-security-center/hp-webinspect
[6] http://www-01.ibm.com/software/awdtools/appscan/

CSCW'14, February 15–19, 2014, Baltimore, Maryland, USA.

ACM 978-1-4503-2540-0/14/02...$15.00.
http://dx.doi.org/10.1145/2531602.2531722

In this paper, we investigate the security tool adoption problem by studying the factors that influence the adoption of security tools, based on a multidisciplinary field of research called *diffusion of innovations* [34]. This paper's primary contribution is a qualitative account of social factors influencing security tool adoption. We consider software development companies as the social systems in which security tools are spread. Thus, many of these social factors are manifested in and shaped by the organizational structure and policies of these companies. We also address social factors by investigating how developers communicate about security tools and how their attitudes toward different communication channels affect adoption. Our findings, based on an interview study with 42 software developers, can help organizations predict return on investment for new security tools, toolsmiths to understand the needs of their users, and educators decide how to teach and present security tools to their students.

We feel that an account of adoption focusing on such social factors is overdue. Previous work in software engineering has focused on developing better tools, but such work does not account for tool adoption behavior. Previous work in human aspects of software engineering has described social factors on the adoption of software process [37] and programming languages [19], but such work has not, until now, been extended to programming tools in general or security tools in particular. Future security tools would be better positioned to improve the state of the practice if they were designed and presented to developers in ways that make adoption more likely. This foundational work will help us develop an understanding of adoption that we hope will help increase the impact of future research into security tool development.

RELATED WORK

In this section, we discuss existing theories of technology diffusion, as well as related studies of developers' adoption of tools and technologies.

Adoption Theory

To make our investigation of as the security tool adoption problem as comprehensive and principled as possible, we used an existing theory of *diffusion*, or the spread of new ideas and technologies. This theory, Rogers' diffusion of innovations (DOI) theory, is one of the most widely used diffusion theories across disciplines. Rogers defines DOI as "the process by which an innovation is communicated through certain channels over time among the members of a social system" [34]. DOI explains the process by which a technology or idea spreads through a population. It describes how a group of people learn of a new technology, try it for themselves, then decide whether or not to adopt it for long-term use. DOI also names social and technological factors that speed or hinder this process. For instance, highly influential early adopters and organizational mandates can increase both the probability and speed of adoption. Technological factors, like the complexity of a technology and its compatibility with existing technologies, also influence adoption patterns. The present paper focuses on social aspects as they are manifested in software development companies.

Other theories describe what influences individuals' intentions to use information technology. These include the Technology Acceptance Model [8, 40], Perceived Characteristics of Innovating [20], Theory of Planned Behavior [1], and Model of Personal Computer Utilization [39]. Though DOI shares many traits with these theories, DOI accounts for communication in ways other theories do not. Our previous findings indicate that the way developers learn about tools affects their adoption behavior [23], and DOI's focus on communication about new technologies allows us to study this phenomenon further.

Software Technology Transfer

An area of research related to DOI is software technology transfer (STT), in which software engineering innovations are transferred from an idea to common practice [5]. Researchers have studied STT in different contexts. Nahar and colleagues studied transfer of software development tools and methods from a multinational Finnish company to its Indian subsidiary, and described how well-trained internal consultants helped this process succeed [24]. Gorschek and colleagues developed a model of STT based on their experiences collaborating with industry [10]. They found that training adopters in using the technology and long-term commitment from management, among other factors, increase the probability of successful STT. Our paper extends such results to security tools.

Adoption of Software Engineering Technologies

We also expand other research on the adoption of some kinds of software tools, methodologies, and programming languages. One of the most widely studied classes of tools is that of Computer-Aided Software/Systems Engineering (CASE) tools. For example, Premkumar and Potter surveyed developers to evaluate a model of factors that influence CASE adoption found five important factors, such as presence of a product champion and support from management [28]. Iivari conducted a similar study evaluating ten factors; his results suggest the importance of management factors, voluntariness, and the perceived advantages of CASE tools [15]. Kemerer demonstrated how learning curves can affect CASE adoption and the challenges in applying traditional learning curve models to CASE tools [16]. Other studies of CASE adoption identified factors that were part of or extended factors from DOI theory [30, 31, 14].

Researchers have also studied diffusion of software development methodologies. For example, Raghavan and Chand used DOI in an informal case study to investigate why the adoption of structured design methodologies was slow [29]. They recommend that researchers produce more evidence for the benefits of such methodologies and find ways to effectively communicate these benefits. Riemenschneider and colleagues showed that developers' intention to adopt is determined by the presence of an organizational mandate, the compatibility of the methodology, and peer influence [32]. Hardgrave and colleagues investigated the determinants of developers' intention to follow methodologies, showing that an organizational mandate cannot guarantee the sufficient use of

the software methodology [12]. Singer and Schneider studied how to improve the adoption rate of software engineering methodologies using social software [36].

Recent work has also investigated the diffusion of programming languages and language features. We previously empirically investigated the adoption of Java generics by mining the history of 40 popular open-source Java programs, and found that adoption of generics there is typically incomplete and championed by one or two individual developers [26]. In their work, Meyerovich and Rabkin posed several open-ended questions and hypotheses about the adoption of programming languages and features [19].

One particularly relevant set of methodologies are Secure Development Lifecycles (SDLs), which prescribe practices that integrate security into the software development process. McGraw suggests the use of tools to help build more secure software [18], making policies relevant to our work. The Microsoft SDL suggests a list of static analysis tools to use in the implementation phase and a list of dynamic analysis tools to be used in the verification phase [13]. Because using security tools is part of SDLs, the adoption of an SDL entails the adoption of security tools, which is the main concern of the present work. However, according to Geer's survey, few software development companies formally adopt SDLs [9]. The main self-reported reasons for non-adoption are that formal SDLs are too time consuming, developers are not aware of the existence of SDLs, and formal SDLs require too many resources. In "A Few Billion Lines of Code Later" [3], Bessey and colleagues discuss challenges faced when trying to deploy the security tool Coverity in industry. Our work empirically investigates some of the social and technical issues described anecdotally there.

Finally, though it does not specifically concern tool adoption, Xie and colleagues' study on the reasons developers make security errors [41] is similar to ours methodologically and in spirit. We also use an interview study and are concerned with developers' attitudes toward secure development. One of their central findings is that many developers feel that security is not their responsibility. While Xie and colleagues' interview study focused on developers' reasons for not considering security when they write software, ours also focuses on tools and developers' reasons for using or not using security tools.

METHODOLOGY

We conducted one-on-one interviews with 42 software professionals to collect data with which to study the security tool adoption problem. More specifically, our research objective was to qualitatively describe the factors that influence adoption or rejection decisions, as well as to describe how those factors interact. In this section, we describe how we recruited participants, some characteristics of the participants we interviewed, how we conducted the interviews, and the how we analyzed the data we collected.

Recruitment and Incentives

We sought to interview professional software developers from diverse backgrounds. We planned to recruit approximately the same numbers of developers working in large, medium-sized, and small companies. To that end, we recruited study participants in four ways: we emailed invitations to 50 software development companies listed on Guru[7], a recruitment website that connects freelance software developers with those who want to hire them. We also emailed personal and professional contacts working in software development. We posted flyers in break rooms at a large software development company. Finally, we asked developers who had already participated in our study to help recruit friends and colleagues who might be interested in participating. We incentivized participation by offering $50 gift cards for completing the interview.

When developers expressed interest in participating, we asked them to sign a consent form and complete a screening survey online. We used the results of this survey to select developers who were at least 18 years old, held a job related to software development, and actively wrote code as part of their professional activities in the past six months. We emailed developers who reported meeting these criteria to schedule the interview.

Participants

Forty-four developers signed the consent form and completed the screening survey. However, we later learned that one of these people did not meet our selection criteria; another could not complete the interview due to significant communication difficulties. We excluded these interviews from our analysis. Participants worked in various countries and developed software in many application areas, including banking, military, financial, and clinical software, as well as web browsers, and software as a service. They used popular programming languages such as Java, JavaScript, C, C++, C#, Python, Ruby, and Perl. Their work experience ranged from 1 to 28 years, with a mean of 11.0 years and a median of 7.3 years.

We sought to interview developers who program as part of their jobs, but also those who may be involved in the tool adoption decision-making processes: managers, security experts, and software consultants. Table 1 shows the numbers of participants who reported working in each development role. Table 1 also shows how many developers worked in small, medium-sized, or large companies. We classified companies thus: small companies employ 50 or fewer developers, medium-sized companies employ more than fifty but fewer than 500 developers, and large companies employ 500 or more developers. As shown, we interviewed 31 programmers, 5 software development managers, 4 software security experts (some of whom developed security tools themselves), and 2 software development consultants. The majority of our participants are programmers, because we were interested in programmers' experience, or lack thereof, with security tools. We also asked managers, security experts, and consultants questions about their activities outside programming if time allowed after asking the more general tool adoption questions. Specifically, we asked managers what factors they considered in deciding whether to adopt a security tool for their team; we asked security experts what factors contributed

[7] http://www.guru.com/

to the adoption of a security tool, how best to advertise security tools, and how to increase awareness of the importance of secure coding practices among ordinary programmers; and we asked consultants questions about differences between domains with respect to secure coding.

Interview Methodology

We interviewed each participant for 45 minutes to an hour over Skype or phone. The interviewer used a script to guide the interview[8]; this script contained open-ended questions about participants' security tool adoption choices, and more directed questions about the importance of social and technological factors from DOI theory.

As in other exploratory studies [41], we used a semi-structured interview methodology [35]: if the interviewer found a participant's answer particularly interesting, she asked questions not in the script to collect more detailed data. If one of these impromptu questions yielded useful information, we added it to the script for the next interview. If a question stopped yielding new information, we stopped asking participants that question. Thus, we did not ask every participant every question.

Because the interview was retrospective in nature, participants might have forgotten about tools that they no longer use. To mitigate the risk of formerly-used tools being unreported, we tried to prompt responses about such tools by giving examples of security tools they may have used and security problems they may have solved in the past. This gave participants an opportunity to recall security tools that they had forgotten about or not have previously thought of as security tools.

Before we started each interview, we confirmed with each participant that we could record his or her voice. After the participant consented, we explained the purpose of our study and our definition of security tools. At the end of the interview, we thanked the participant and arranged for delivery of the gift card offered as incentive.

Data Analysis

We recorded each interview we conducted, then transcribed each recording. We used a qualitative data analysis program called Atlas.ti[9] to analyze the interviews using open coding [35]. We started with an initial set of codes based on findings in Rogers' discussion of diffusion of innovations [34]. We created new codes when we found interesting patterns that were previously unaccounted for. For example, our "evaluation process" code, which accounts for companies and managers having a formal process in place to evaluate new security tools, was not part of our initial set of possible relevant factors. When evaluating whether a participant actually adopted a tool, we considered an instance of adoption to occur when a developer used a tool on a regular basis for at least one week.

[8]The interview script is available at **http://www4.ncsu.edu/~sxiao/adoption/interview_script.pdf**

[9]http://www.atlasti.com/

In our results sections, we will refer to each participant by an alias of the form Pi, from P1 to P42, to allow the reader to track individual participants. When discussing responses to questions that we only asked a subset of participants, we will use expressions such as "x out of y participants said A" to indicate that of the y developers who we asked a particular question, x answered A. Note that this work is exploratory and qualitative, and considers a relatively small number of participants. While we will present some statistical information about our participants and their behavior, such information serves only to summarize our results — the reader should not infer any generalizations about developers' behavior from our data.

PARTICIPANTS' SECURITY TOOL USE RATES

In this section, we discuss general patterns we discovered in participants' use of security tools. We divided the participants into 3 categories based on their statements in our interviews: regular users, who used security tools regularly in their daily programming work; occasional users, who used security tools only when they considered it necessary to search for vulnerabilities; and non-users, who never used security tools.

When we initially asked participants if they had used any security tools, many participants said that they had not. However, when we gave these participants examples of what we considered security tools, some participants reported using them. We attribute this to our deliberately liberal definition of security tools. Some tools that participants used, like Coverity, are marketed as security tools, with finding vulnerabilities given as a primary use case. However, we also consider a tool a security tool if it can be used to find vulnerabilities, even if that is not its exclusive purpose. For instance, memory profilers like Valgrind[10] help developers find buffer overruns and other memory errors that can, in some cases, allow an attacker to gain control of the instruction pointer and execute arbitrary code on a compromised machine. As another example, FindBugs can analyze code for API misuse that leaves a server open to cross-site scripting and SQL injection attacks. Thus, we considered the participants who regularly used Valgrind or FindBugs to be regular users, even though finding vulnerabilities is not these tools' primary use case. We have chosen this broad definition because, at this exploratory stage, we would rather overgeneralize than miss a factor we had not previously accounted for.

The nineteen occasional users reported many reasons for not using security tools regularly. Nine of the nineteen reported using a tool to fix a security problem, then discontinuing use of the tool after fixing it. For example, P33 once used a memory profiler to find memory leaks in his code, then stopped using it once he had repaired the leaks. Seven of the nineteen said that they stopped using a security tool because they moved to a development team in a different application area in which they felt security tools were not needed. Two of the nineteen reported that they stopped using a security tool out of "laziness", and the other three said they were not sure why they discontinued use of a tool.

[10]http://valgrind.org/

Company Size	*Programmer*	*Manager*	*Security Expert*	*Consultant*	**Total**
Large (500+ Developers)	14	3	3	2	22
Medium (51-499 Developers)	8	1	0	0	9
Small (1-50 Developers)	9	1	1	0	11
Total	31	5	4	2	42

Table 1. Participants' Backgrounds

Among the ten non-users, six participants worked at small companies and reported that these companies did not require that they use secure coding practices. Participants who worked for these small companies appeared to know less about secure development and secure coding than other participants. Some thought of security as a narrow concern mainly focused on operational security, software access control, and user authentication.

In general, most of the regular users came from large companies where security tools have been integrated into the standard development process, while many of the non-users came from small companies that do not require developers to use secure coding practices.

ORGANIZATIONAL FACTORS

One of the strengths of DOI theory is that it accounts for how characteristics of the social systems in which innovations diffuse affect their adoption. Since all our participants were software developers working in industry, we consider the companies in which they work as the main social systems in which they use and learn about security tools. The social norms and standards of these companies manifest themselves as the company's policies, structure, culture, and practices. In this section, we will discuss the impact these social environments have had on participants' adoption decisions concerning security tools. Specifically, developers work in different companies with different cultures, practices, standards, and structures that could influence adoption in many ways. Our results show that these characteristics of the company for which a developer works can greatly influence her decisions in adopting security tools.

Policies and Standards

A company's policies and standards refers specifically to the policies or standards a company holds that relate to secure coding and the use of security tools. Several participants reported that the standards of their company directly influence the perceived necessity of security tools and their actual use of security tools.

Only two participants reported that their companies have formal secure development lifecycles that require them to use security tools. Fewer than half of the participants reported that their companies have and enforce formal secure coding standards at all. Of the thirteen regular users, seven reported that their companies require them to use some specific security tools as part of the development process, while none of the nineteen occasional users and ten non-users reported their companies have these policies or standards. In other words, all participants who were required to use security tools used them; this is notable, as other research shows that people will circumvent security policies if they are inconvenient, even in hospitals, where the consequences of circumvention can be harmful to patients [17]. However, this conforms to Straus and colleagues' finding [38] that mandates effectively encourage adoption of new technologies in organizations.

In most cases, however, security standards are informal, verbal "best practices". Participants reported following these practices and expecting their coworkers to follow them, though no policies or guidelines were written or enforced by their employers. P11 said that the standards in his department were "informal" and not "well-promoted". P21 said that there was no formal requirement that passwords be stored encrypted, but that doing so was an "unwritten standard". These decisions might be enforced by informal communication between colleagues, such as pointing out vulnerabilities on mailing lists.

Two participants reported that in their companies, using a tool that is new to the company requires written authorization; obtaining this authorization sometimes takes weeks. Thus, these participants rarely, if ever, investigated new tools on their own to avoid this long wait. For example, P36 reported that at his company, if he wanted to start using a new security tool, he would "have to get approval from [his] manager" or through a more complex process, "depending on the tool". Tools to be "only [used] internally", that were not involved in any client-facing services, might be approved or rejected by management in "in a week or... a couple of weeks", which P36 considered a "very short time". Tools used in customer-facing services and products would "take a lot longer" to approve. P19 said that at his company, tools were evaluated to ensure that the were cost-effective and fit into the organization's existing workflow.

The managers we spoke to shed some light on this evaluation process. For instance, P5 and P7 indicated they evaluated tools based on their cost. Both considered developer time part of that cost. P5 who, he managed development of a system with components written in many languages, said that the initial cost in time to set up security tools for these varied components was disproportionate to the benefits it could have. P7 said he prefers tools that "can be scripted to run automatically".

Among the thirteen regular users, seven worked in large companies where security tools were integrated into the development process. Three of these seven were required to use a security tool on their local workstation. The other four reported that security tools were integrated into their companies' build server; the security tools automatically check for vulnerabilities as part of a nightly build or before participants check in their changes. These participants were able to circumvent the costs of installing, configuring, learning, and running these

security tools — they needed only to read and understand the reports generated by the tools.

Four participants reported feeling they did not need to use security tools because they felt mandatory manual code reviews already ensured security. Two participants mentioned they had used security tools in conjunction with manual code review; this practice gave them more confidence in their code's security. Overall, seven of the thirteen non-users reported feeling no need to use security tools because they could depend on code reviews or testing by others in the company ensure security. P9 trusted security experts to "catch the problems" in his code. This corroborates Xie and colleagues' finding that developers tend to place trust in process when it comes to software security [41].

Culture

By culture, we mean the beliefs and social norms surrounding security and security tools in a company. Different company cultures can influence developers' security tool use: for example, P28 worked in a company where installing new tools on his workstation is free and encouraged. He and his coworkers consider such exploration "cool".

However, other companies seem to have a culture that discourages using new tools. For instance, as mentioned previously, P36 worked in a company where the approval process for the use of a new tool might take weeks. He reported that management would "encourage" developers to see if there was a tool that was already approved for use in order to avoid this process. Between this encouragement from management and the length of the approval process, P36 said that he would need a particularly "compelling reason" to use a tool that had to be approved. Thus, policies that necessitated a long evaluation period helped create a culture of avoiding new security tools.

Our interviews with managers confirmed that managerial involvement affected company culture around security. P5, a manager, said it was "very clear" that security is "everybody's responsibility" in his company, and that is was "a culture thing" in his company. He and other managers actively fostered that culture by encouraging collaboration between security experts and other developers and offering security classes to developers.

Domain and Security Concerns

By a company's domain and security concerns, we mean the importance of security in the domain in which the company develops software. Of thirty-four participants, twenty-six considered their companies' domains security-sensitive. Despite this high awareness of security issues, only ten participants were regular users, as mentioned previously. There is a dissonance, then, between developers' knowledge and behaviors: Sixteen participants think the security of their code is important, but do not use tools to help increase that security. Most developers may not have a concrete understanding of the consequences of deploying insecure software: five participants reported that they only started using security tools after vulnerabilities in their companies' code were exploited. Three participants said they do not use security tools because they never had been "hacked". This is predicted by DOI theory, as it is a common problem for other *preventive innovations* [34]; we will discuss this issue further in the Discussion section.

The perceived user base of the applications developed by software developers had an influence on participants' security tool adoption decisions as well. Four participants said they did not use security tools primarily because the applications they developed were only used internally by authenticated users, so they felt they did not need to worry about the security of these applications. Their conception of software security mainly concerned data access control and user authentication, as opposed to other kinds of security, like memory safety and input sanitization. Six participants mentioned that they care about security because their applications have a large number of users. For instance, P28 said that developers "need to make sure that [their software wasn't] opening up security holes" for the software's 400 or 500 million users. P19 said that security was less important than other requirements of his software because he developed "internal applications". Both considered user-facing applications more likely to be attacked and were more concerned about security if many users might suffer from an exploit.

Seven participants said that in their companies, functionality is the first concern because of tight deadlines for delivering products. Like participants in Xie and colleagues' study [41], they reported not having time to worry about security, as security is a non-functional requirement. P19, for instance, said that, partly because of limited development time, his team felt that for the software he developed, "achieving the task [was] more important than having a fully secure solution".

Structure

A company's structure, with respect to security tool adoption, refers to whether the company has dedicated security and testing teams. Of 31 participants, 17 reported that the companies they work in have dedicated security teams. Security teams perform different functions in different companies. P13 reported that the security team at his company held regular meetings with developers to suggest the use of new tools. In P6's company, developers could request security audits from the security teams. These security teams would use security tools to find vulnerabilities.

Security teams design security guidelines, offer security trainings and look for security tools for ordinary programmers to use. However, only six of the security teams interacted often with ordinary developers to consult and to review developers' code. The other security teams, which did not often interact with developers, tended to focus on operational security — they prioritized access control and authentication concerns, penetration testing, and auditing open-source software used in the company but developed outside it. For instance, P25 reported that the security team at his company would scan his software for vulnerabilities just before a release, not through the entire development lifecycle.

P32's statements illustrate the attitude of one developer who rarely interacts with his company's security team. He re-

ported that the security team was "less than helpful" and "never checked any code" that he or his coworkers wrote. Rather, they looked for vulnerabilities in open-source code used by the company's products. P32 considered this a "waste of time" and thought that "they should have [spent] their time analyzing [the company's] code base".

While Xie and colleagues [41] found that developers hold a relaxed attitude towards security when they have security teams or testing teams to back them up, we found that not all of our participants held this attitude. Only two out of twenty-six participants thought ordinary developers were less responsible for security than security and testing teams. The other participants believed they had a, to quote P35, "shared responsibility" for writing secure software. They said that while security experts and testers helped ensure software security, writing secure code is ultimately the developers' responsibility. Especially in cases where ordinary developers interacted often with the security team, participants reported that they felt social pressure from the security team by having them audit the code and review the new features from a security perspective. This ultimately made developers feel personally responsible for their code's security. Nov and Arazy found that personality traits also influence peoples' decisions to help others in computer-supported cooperative work contexts when they know that they could defer responsibility to others [25]. This notion of personal responsibility for security is complex, and warrants further study.

Education and Training

Education and training refers to the training a company provides for general security problems and for specific security tools. Overall, no participants reported adopting a security tool through education and training. Though three of thirty-two participants mentioned that they learned about some security vulnerabilities in a software engineering course, none of the participants had had any security-specific courses in college. Twelve of thirty-one participants said that their companies provided training that touched upon security. However, only three of them have training for specific security tools. Rather, most of the training consisted of education on specific vulnerabilities and security best practices. Most trainings are optional, some are held only once a year, and some are just for newly hired developers. P6 indicated that questions about security were part of his employer's interview process. Thus, he assumed that developers who joined the company already understand security vulnerabilities, so he felt security training was unnecessary in his company.

P36 explained why security tool training may be uncommon in his company. To avoid the time costs of training all of its developers to use security tools, his company "want[ed] to let certain individuals be more specialized about security" and not "[have] the whole team understand security". This shows another way that the presence of security teams can negatively affect developers' security tool adoption behavior: while security teams can make developers feel social pressure to code securely, as discussed previously, they also can make developers feel that security is not their responsibility.

COMMUNICATION CHANNEL FACTORS

Rogers [34] makes a number of claims about the characteristics of communication channels that make people more or less likely to adopt the use of a tool that they hear about through those channels. Through our interviews, we have found two high-level characteristics of these channels that influenced participants' adoption behavior: the amount of *exposure* that the channel gives the innovation to individuals, and the extent to which the individuals *trust* that communication channel.

DOI theory accounts for three types of communication channels: mass media, interpersonal channels, and interactive communication on the Internet. Examples of mass media channels include webpages, television, radio, newspaper and magazines. Interpersonal channels involve face-to-face communication between two or more people, such as in peer recommendation. Interactive communication on the Internet happens on online forums and discussion websites, where people can discuss common interests, and social networks, such as Twitter and Facebook.

Exposure

To learn what communication channels expose developers to security tools, we asked participants what security tools that they had heard about and how they heard about them. Specifically, we asked them to list the security tools they had used, then asked how they first learned about each. We also asked participants if they had ever learned of any security tools through each of the channels listed in Table 2. We did so to prompt them to discuss tools that they had heard of but never used.

In Table 2, we list the channels that participants mentioned and the number of participants who mentioned each. In the following subsections, we indicate what participants said about each type of channel with respect to how and how much they exposed participants to security tools.

Interpersonal Communication

Our previous work [23] indicates that developers sometimes learn about tools through interpersonal channels — specifically, from their peers. The results of our interviews corroborate this finding: participants mentioned a total of 34 instances of learning about a tool through an interpersonal channel, 18 of which were through coworkers' suggestions.

P37 recalled a time when he discovered a security tool when a coworker who sat next to him shared a new tool with him in person. However, P3 mentioned that such peer recommendation happens rarely because developers do not often drop by other developers' offices; this corroborates our previous findings [23].

Twenty-two of thirty participants reported that they had recommended security tools to other developers. All eight of the other participants indicated that they did not recommend security tools to their friends because of the tools' cost. P5, for instance, reported that one tool his company evaluated cost 50,000 dollars per license. They felt that these tools are prohibitively expensive for individual users, and that adopting a

Communication Channel Type	Channel	Participants Mentioned	Total
Interpersonal Channels	Coworker Recommendation	18	34
	Required by Management	6	
	Conference	6	
	Security Team Recommendation	4	
	Security Tool Vendor	4	
	Required by Customer	2	
Mass media	Technical Blogs and Websites	4	7
	Security Tool's Official Website	3	
	Web Advertisements	–	
	Television	–	
	Radio	–	
	Newspapers and Magazines	–	
Interactive Communication on the Internet	Online Forums and Discussion Boards	4	4
	Social Networks	–	

Table 2. Security Tool Discovery Channels

tool is a managerial, not individual, decision. Eighteen of another thirty participants reported actually using a security tool or tools recommended to them by peers. We were surprised that participants reported six instances of first hearing of tools at security conferences and trade shows; we did not expect so many participants to learn about tools through interpersonal channels outside their companies.

Mass Media Communication

Participants reported seven instances of discovering a tool through mass media channels. However, no participants discovered tools through traditional mass media channels like television, magazines, or radio. They only mentioned mass media channels that presented information to them on the Internet.

Participants reported four instances of discovering a security tool by reading a technical blog or website. Participants also said that they sometimes searched for security tools online when they felt the need to use them; for example, when participants suspected their code had memory leaks, they might search for the terms "security tools for memory leaks". These searches sometimes brought them to security tools' official websites. Participants reported three instances of learning about a security tool through a tool's official website discovered through web search.

Though two participants recalled seeing security tools advertised online, neither recalled what tool they saw advertised. Moreover, participants reported that they had never seen security tool advertisements through traditional mass media channels, such as television and radio. Three participants reported they had seen security tool advertisements in software magazines, but none remembered the names of the security tools advertised. Overall, the Internet played a more important role than traditional mass media channels in participants' awareness of security tools.

We find it interesting that only five out of thirty-four participants recalled noticing security tool advertisements in Google search, forums, and software magazines. Participants reported that they generally ignored security tool advertisements because they did not trust such advertisements; we discuss the issue of trust further in the following section.

Interactive Communication on the Internet

Participants reported four times that they saw a security tool in online forums and discussion threads in their research about security tools. However, participants reported that they had never learned of security tools through social networks, which is consistent with our previous finding that developers rarely discover tools on Twitter [23].

Trust

Trust refers to the reliance of a developer on a communication channel to deliver relevant and useful information about security tools. We asked participants to tell us how much they trust different communication channels, both individually and relative to other channels.

Interpersonal Communication

Participants generally trusted "word of mouth" more than other sources — for instance, P3 said that he would be likely to try a tool recommended by someone he knew, more so than one he read about. Three participants reported trusting tool recommendations most from people with similar professional duties. P33 indicated that he trusted tool recommendations from technical managers, but not those from managers without a technical background. This also conforms with DOI research [34] and our previous computer supported cooperative work research [23], both of which suggest that people are more likely to trust information from individuals who share similar socioeconomic and educational status.

Mass Media Communication

Participants reported that they do not trust advertisements in mass media, such as those on webpages and magazines, because they considered mass media advertisements "biased". P37 indicated that he "did not trust [advertisements] to be unbiased" and would rather seek information from an expert he trusted. Only five of twenty-seven participants recalled noticing security tool advertisements on webpages or in magazines; most said they did not trust advertisements and ignored them.

Two participants also mentioned that they trust security tools and advertisements for them more if the tool is made by a well-known company. For example, participants mentioned they trust security tool advertisements from Microsoft and Hewlett-Packard more than from other companies that they had never heard of.

Interactive Communication on the Internet

We found it particularly interesting that some participants showed nearly equal trust toward recommendations received in person from peers and recommendations on the Internet given by people with good reputations or demonstrated credibility. P28 claimed that he trusted people he interacted with on the Internet, such as Stack Overflow users, more than he trusted some people he interacted with physically, such as his coworkers. Another participant reported that he would trust a well-moderated, high-scoring answer to a question on Stack Overflow over a recommendation from a colleague, particularly if the answer was from a highly-ranked Stack Overflow user. However, he also reported that he would trust a colleague's recommendation over a less thorough answer from a Stack Overflow user without a high rank.

DISCUSSION

We find it remarkable that so many participants believed that security is important and that they were personally responsible for the security of the software they develop, and yet did not use security tools. We believe that this is partly because security tools are what Rogers calls a *preventive innovation*. Preventive innovations are technologies that lower the probability of some unwanted future event. Condoms and vaccines, for instance, are preventive innovations. Preventive innovations diffuse slowly because of the temporal distance between the use of the technology and its effects, and because it's easier to see the negative effects of not using them than the positive effects of using them [34]. We believe that the challenges to creating cultures around thorough testing, as detailed by Pham and colleagues [27], may be due to the preventive nature of writing good tests.

We consider security tools a preventive innovation because security tools are used to lower the probability of security problems before deploying the software, but security problems generally occur after software is deployed. We found that a developer's code being "hacked" works as a *cue-to-action* for that developer to adopt a tool. A cue-to-action, in DOI theory, is an event that crystallizes a favorable attitude towards an innovation into a behavioral change [34]. In security tool adoption, being "hacked" can push developers who are considering using security tools to start using them. This indicates that one way to prevent losses that occur when vulnerabilities are exploited and must be repaired is to find more effective ways to warn developers about such losses.

The results of our study suggest a number of strategies for educators, software development companies, toolsmiths, and developers to increase adoption of security tools. These are preliminary recommendations that we hope to expand in more detail and to further justify in our future work.

Educators

We found some developers believe most security problems concern user authentication and access control. We think educators are in a unique position to present the broad spectrum of security problems to computer science students. Educators can also introduce students to tools that help solve those problems and encourage them to learn and explore new tools throughout their careers. Thus, educators are in a unique position to instill their students with inquisitive and enthusiastic attitudes toward security and security tools. Classroom studies such as Singer's [37] demonstrate that educators can effectively teach testing in the classroom, and we hold that they can also effectively promote other good software engineering practices, like the use of security tools.

Software Development Organizations

Companies can design policies regarding security tools to increase adoption among developers. We found, perhaps surprisingly, that policies requiring developers to use security tools seem quite effective. While secure development processes help developers code more securely, trust in process can make developers complacent and less likely to code securely, as reflected in our findings and those of Xie and colleagues [41]. Our findings also suggest that if companies structure their security processes so that security teams and other developers often interact, developers are more likely to feel personally responsible for security.

Companies might also encourage and support security tool use training developers to use tools that support it; this does not seem to be a common practice. Companies seeking to use the best and newest security tools to find vulnerabilities in their code might encourage developers to attend security conferences and make it easy to start using new tools in their organizations.

Finally, developers might be more security-conscious if their employers emphasize the potential costs to users of exploits on their code; developers who are conscious of these costs may be more likely to use security tools.

Toolsmiths

We found a number of ways that toolsmiths might focus their efforts to increase awareness of their tools. Toolsmiths might increase adoption of their tools by focusing their efforts on trusted channels like word-of-mouth, blogs, and Stack Overflow, and not using their resources promoting their tools with advertisements.

Toolsmiths designing tools to support code review and security auditing might also keep our results in mind. We found that developers who interact with security teams as part of the auditing process are more likely to adopt security tools, so future code review tools might be designed to support social as well as technical aspects of the code review process. Previous work has found that code review already serves social as well as code-improving purposes [2], and that code review tools already provide social benefits, like traceability, that other means of review do not [33]. Future code review tools might be designed to support the social interaction of developers and security teams.

Developers

We find it unlikely that this paper's audience includes developers unconcerned with software security. However, we remind those who are concerned that their peers are likely to take their security tool recommendations seriously. Our findings, here and elsewhere [23], indicate that developers are more likely to adopt tools they learn about from their peers than ones they learn about elsewhere. These findings show that individuals who tell others about the tools they use can be forces for change in their organizations. In addition, developers with good reputations on the Internet can act as forces for change within communities of developers in the software development community at large.

LIMITATIONS

One limitation for our retrospective interview study is the *recall problem*. Some participants noted that it was difficult to remember the tools they have tried out. Thus, our data on instances of adoption may be incomplete. For example, participants might remember the functionality of a security tool, but forget how complete its documentation was. We asked pointed questions to try to obtain this data. If a participant did not remember, we assumed the missing data was not important in his decision-making. Also, participants tended to forget the names of security tools they rejected more than those they adopted. In order to remind participants of the tools they might have tried, we created a list of well-known security tools, grouped by the programming languages they work with. After participants mentioned all the security tools they remember, we asked them to read this list based on the language they used most to find if they missed any tools.

This retrospective interview study also could not adequately capture the temporal aspects of DOI theory. An observational study might be better suited for evaluating how well the DOI adoption process describes security tool adoption.

Another limitation of our interview study is that, given the nature of semi-structured interviews, we did not ask all participants the same questions. Thus, we cannot guarantee that our data is complete since we cannot know how participants would respond to questions we did not ask them. However, we think that the breadth of our results justifies this possible incompleteness.

Because we designed our interview questions based on DOI theory and our initial ideas of how it would apply to security tool adoption, we may have biased our results with our preconceptions. To reduce this risk, we asked participants many open-ended questions to describe their adoption and rejection experiences before we started asking any pointed questions based on our interpretation of DOI. We tried to record all factors that we had not initially predicted and integrate these factors into our questions for subsequent interviewees.

The participants we recruited may not be representative of all software developers for a number of reasons. Though we did not specifically choose participants based on location, more than half of our participants reported working in the United States. All participants we recruited speak English, while many potential security tool adopters do not speak English. Also, developers who were willing to participate our study might hold more positive attitudes than average toward the software engineering research community, and perhaps are more likely to be aware of and adopt new security tools. We did not speak with any developers outside of industry, such as those who contribute to open-source software. This limits the kinds of social systems to which our work generalizes to those in industry. Most of our subjects' primary responsibility was writing code. While we did interview security experts and managers, our implications apply more reliably to those whose primary activity is writing code. Finally, we did not collect detailed demographic information from participants. Such qualities of individuals, such as age [21] and gender [11], can affect various aspects of software development, and may also affect social processes such as adoption.

FUTURE WORK

As mentioned in the Limitations section, we did not interview developers about open-source development. An interview study of developers who write open-source software would help determine if our findings generalize to developers outside of industry.

We believe that a qualitative and comprehensive model of influences on developer adoption of security tools would be more actionable than the current descriptions of participants' responses. Such a model would incorporate developers' perception of technological qualities of security tools as well as the factors concerning interpersonal communication presented here. For instance, several participants mentioned that automatabilty and interoperability with technologies they already used were important factors when deciding whether or not to adopt a tool. Our analysis currently does not account for these more technical factors, but could with further investigation.

A survey study would also be helpful in evaluating and quantifying our findings. We hope that this will result in validated recommendations we can make to managers seeking to increase security tool adoption in their organizations and a quantified predictive model that can predict whether or not developers in a given organization will adopt a given tool.

Our results have also shown that some developers attribute their tool use to having seen exploits on their companies' code; others attribute their disuse to having never been "hacked". This indicates that one way to persuade developers to use security tools may be to frame the results of such tools in terms of specific attacks that would successfully exploit vulnerabilities in developers' code. Future study could explore this.

CONCLUSION

There are many security tools that can help software developers write more secure code, but not all developers use these tools. In this paper, we have presented a number of social and organizational factors that we found influenced developers' security tool adoption decisions. We found these factors by conducting interviews, based on an established framework for understanding adoption called diffusion of innovations, with 42 professional software developers about their past experience about security tools. Our results suggest several ways

for people in different roles in the software development community to improve the adoption rate of security tools, and thus to foster a culture that produces more robust and secure software.

Acknowledgment

Thanks to all participants for their time, to Michael Bazik for his comments and feedback, and to the National Security Agency for funding this research.

REFERENCES

1. Ajzen, I. The theory of planned behavior. *Organizational Behavior and Human Decision Processes 50*, 2 (Dec. 1991), 179–211.
2. Bacchelli, A., and Bird, C. Expectations, outcomes, and challenges of modern code review. In *Proceedings of the 35th International Conference on Software Engineering* (2013).
3. Bessey, A., Block, K., Chelf, B., Chou, A., Fulton, B., Hallem, S., Henri-Gros, C., Kamsky, A., McPeak, S., and Engler, D. A few billion lines of code later: using static analysis to find bugs in the real world. *Communications of the ACM 53*, 2 (Feb. 2010), 66–75.
4. Boehm, B. W. *Software Engineering Economics*, 1st ed. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1981.
5. Buxton, J., and Malcolm, R. Software technology transfer. *Software Engineering Journal 6*, 1 (Jan. 1991), 17 –23.
6. Cornell, D. Remediation statistics: What does fixing application vulnerabilities cost? RSA Conference, 2012.
7. Cowley, S. Code red costs could top $2 billion. PCWorld, August 2001. `http://www.pcworld.com/article/57744/article.html`.
8. Davis, F. D. Perceived usefulness, perceived ease of use, and user acceptance of information technology. *MIS Quarterly 13*, 3 (Sept. 1989), 319–340.
9. Geer, D. Are companies actually using secure development life cycles? *IEEE Computer 43*, 6 (June 2010), 12–16.
10. Gorschek, T., Wohlin, C., Carre, P., and Larsson, S. A model for technology transfer in practice. *IEEE Software 23*, 6 (Nov.-Dec. 2006), 88–95.
11. Grigoreanu, V., Burnett, M., Wiedenbeck, S., Cao, J., Rector, K., and Kwan, I. End-user debugging strategies: A sensemaking perspective. *ACM Transactions on Computer-Human Interaction 19* (2012), 5:1–5:28.
12. Hardgrave, B. C., Davis, F. D., and Riemenschneider, C. K. Investigating determinants of software developers' intentions to follow methodologies. *Journal of Management Information Systems 20*, 1 (July 2003), 123–151.
13. Howard, M., and Lipner, S. *The Security Development Lifecycle*. Microsoft Press, Redmond, WA, USA, 2006.
14. Iivari, J. From a macro innovation theory of IS diffusion to a micro innovation theory of IS adoption: An application to CASE adoption. In *Proc. of the IFIP WG8.2 Working Group on Information Systems Development* (1993), 295–320.
15. Iivari, J. Why are CASE tools not used? *Communications of the ACM 39*, 10 (Oct. 1996), 94–103.
16. Kemerer, C. F. How the learning curve affects CASE tool adoption. *IEEE Software 9*, 3 (May 1992), 23–28.
17. Koppel, R., Wetterneck, T., Telles, J. L., and Karsh, B.-T. Workarounds to barcode medication administration systems: Their occurrences, causes, and threats to patient safety. *Journal of the American Medical Informatics Association 15*, 4 (2008), 408 – 423.
18. McGraw, G. Software security. *IEEE Security and Privacy 2*, 2 (Mar.-Apr. 2004), 80–83.
19. Meyerovich, L. A., and Rabkin, A. S. Socio-plt: principles for programming language adoption. In *Proc. of Onward!*, ACM (2012), 39–54.
20. Moore, G. C., and Benbasat, I. Development of an Instrument to Measure the Perceptions of Adopting an Information Technology Innovation. *Information Systems Research 2*, 3 (Sept. 1991), 192–222.
21. Morrison, P., and Murphy-Hill, E. Is programming knowledge related to age? *Mining Software Repositories* (2013), 3–6.
22. Murphy-Hill, E., Jiresal, R., and Murphy, G. C. Improving software developers' fluency by recommending development environment commands. In *Proc. of FSE*, ACM (2012), 42:1–42:11.
23. Murphy-Hill, E., and Murphy, G. C. Peer interaction effectively, yet infrequently, enables programmers to discover new tools. In *Proc. of CSCW* (2011), 405–414.
24. Nahar, N., Kakola, T., and Huda, N. Diffusion of software technology innovations in the global context. In *Proc. of the Hawaii International Conference on System Sciences* (Jan. 2002), 2749–2757.
25. Nov, O., and Arazy, O. Personality-targeted design: theory, experimental procedure, and preliminary results. In *Proc. of CSCW*, ACM (New York, NY, USA, 2013), 977–984.
26. Parnin, C., Bird, C., and Murphy-Hill, E. Java generics adoption: how new features are introduced, championed, or ignored. In *Proc. of MSR*, ACM (2011), 3–12.
27. Pham, R., Singer, L., Liskin, O., Figueira Filho, F., and Schneider, K. Creating a shared understanding of testing culture on a social coding site. In *Proc. of ICSE*, IEEE Press (Piscataway, NJ, USA, 2013), 112–121.
28. Premkumar, G., and Potter, M. Adoption of computer aided software engineering (CASE) technology: an innovation adoption perspective. *SIGMIS Database 26*, 2–3 (May 1995), 105–124.

29. Raghavan, S., and Chand, D. Diffusing software-engineering methods. *IEEE Software 6*, 4 (July 1989), 81–90.

30. Rai, A., and Howard, G. Propagating case usage for software development: An empirical investigation of key organizational correlates. *Omega 22*, 2 (Mar. 1994), 133–147.

31. Rai, A., and Patnayakuni, R. A structural model for CASE adoption behavior. *Journal of Management Information Systems 13*, 2 (Sept. 1996), 205–234.

32. Riemenschneider, C., Hardgrave, B., and Davis, F. Explaining software developer acceptance of methodologies: a comparison of five theoretical models. *IEEE TSE 28*, 12 (Dec. 2002), 1135–1145.

33. Rigby, P. C., and Bird, C. Convergent Software Peer Review Practices. In *Proceedings of the the joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering (ESEC/FSE)*, ACM (2013).

34. Rogers, E. M. *Diffusion of Innovations*. Free Press, 1995.

35. Seaman, C. Qualitative methods in empirical studies of software engineering. *IEEE TSE 25*, 4 (1999), 557–572.

36. Singer, L., and Schneider, K. Influencing the adoption of software engineering methods using social software. In *Proc. of ICSE* (2012), 1325–1328.

37. Singer, L.-G. *Improving the Adoption of Software Engineering Practices Through Persuasive Interventions*. PhD thesis, Gottfried Wilhelm Leibniz Universitt Hannover, 2013.

38. Straus, S. G., Bikson, T. K., Balkovich, E., and Pane, J. F. Mobile technology and action teams: Assessing blackberry use in law enforcement units. *Proc. of CSCW 19*, 1 (2010), 45–71.

39. Thompson, R. L., Higgins, C. A., and Howell, J. M. Personal Computing: Toward a Conceptual Model of Utilization. *MIS Quarterly 15*, 1 (Mar. 1991), 125.

40. Venkatesh, V., and Davis, F. D. A theoretical extension of the technology acceptance model: Four longitudinal field studies. *Management Science 46*, 2 (Feb. 2000), 186–204.

41. Xie, J., Lipford, H., and Chu, B. Why do programmers make security errors? In *Proc. of Visual Languages and Human-Centric Computing* (Sept. 2011), 161–164.