Contents lists available at ScienceDirect

Simulation Modelling Practice and Theory

journal homepage: www.elsevier.com/locate/simpat

Toward effective adoption of secure software development practices

Shams Al-Amin[a,*], Nirav Ajmeri[b], Hongying Du[b], Emily Z. Berglund[a], Munindar P. Singh[b]

[a] Department of Civil, Construction, and Environmental Engineering, North Carolina State University, Raleigh, NC 27695, United States
[b] Department of Computer Science, North Carolina State University, Raleigh, NC 27695, United States

ARTICLE INFO

Article history:
Received 14 June 2017
Revised 16 February 2018
Accepted 17 March 2018
Available online 2 May 2018

Keywords:
Multiagent system
Security practices
Adoption
Decision theory
Sanction

ABSTRACT

Security tools, including static and dynamic analysis tools, can guide software developers to identify and fix potential vulnerabilities in their code. However, the use of security tools is not common among developers. The goal of this research is to develop a framework for modeling the adoption of security practices in software development and to explore sanctioning mechanisms that may promote greater adoption of these practices among developers. We propose a multiagent simulation framework that incorporates developers and manager roles, where developers maximize task completion and compliance with security policies, and the manager enforces sanctions based on functionality and security of the project. The adoption of security practices emerges through the interaction of manager and developer agents in time-critical projects. Using the framework, we evaluate the adoption of security practices for developers with different preferences and strategies under individual and group sanctions. We use a real case study for demonstrating the model and initialize the occurrence of bugs using a 13 year database of bug reports for the Eclipse Java Development Tools. Results indicate that adoption of security practices are significantly dictated by the preferences of the developers. We also observed that repetitive sanctions may cause lower retention of developers and an overall decrease in security practices. The model provides comparison of security adoption in developers with different preferences and provides guidance for managers to identify appropriate sanctioning mechanism for increasing the adoption of security tools in software development.

1. Introduction

Secure software development tools, or security tools, are programs that analyze software to help developers find and fix vulnerabilities [1,2]. Such tools may analyze software code, find vulnerabilities [3,4], warn developers of probable violations of coding standards [5], or find programming errors through static (FindBugs) [6] or dynamic analysis (Valgrind plugin) [7]. To illustrate the functionality and advantages of a static security analysis tool, consider FindBugs as an example. FindBugs can be run as a plug-in for the Eclipse and NetBeans integrated development environments, using Ant or Maven, from the command line or as a separate tool on its own [1,8,9]. FindBugs can group each bug pattern into a category of correctness, bad practice, performance, or internationalization, and prioritize bugs as high, medium, or low. Findbugs also offers few

[*] Corresponding author.
E-mail address: salamin@ncsu.edu (S. Al-Amin).

https://doi.org/10.1016/j.simpat.2018.03.006
1569-190X/© 2018 Elsevier B.V. All rights reserved.

fixes. However, despite the advantages of these tools, the use of security tools is not common among developers [2]. In a survey [10], 60% developers responded that in their organization developers run FindBugs in an ad-hoc way, 80% responded that there is no policy on how soon each FindBugs issue must be human-reviewed, and 83% responded that FindBugs warnings are not inserted into a separate bug-tracking database in their organization.

Adoption of security practices Previous research that looked into the adoption of security practices are based on quantitative data collected through surveys. Witschey et al. [2] collected quantitative data on the relative importance of factors through an online survey of software developers. They identified 39 factors that affect adoption. These are social system factors such as security concern and awareness, policies and standards, structures, education and training, and culture; innovative factors such as relative advantage, observability, complexity, and trialability; communication channel factors such as trust and exposure; and potential adopter factors. They built a combined logistic regression model using the 39 factors and found 6 factors are statistically significant. The six significant factors are observability, advantages, policies, inquisitiveness, education and exposure. Kina et al. [5] analyzed the decision criteria of software developers based on prospect theory and concluded that developers would avoid selecting tools if the probability of the effect of the tools is unknown. Araújo et al. [11] investigated the effectiveness of existing bug prediction approaches with procedural systems and compared their effectiveness using standard metrics, with adaptations when needed. Ayewah et al. [10] pointed out that the users' willingness to review warnings and fix issues also depends on the characteristics and organization of the project, the time investment they are willing to put into each review, and their tolerance for false positives.

Although surveys can help explore the factors influencing adoption, it is logistically challenging to observe how each developer works in real time. Simulation can be used to explore and predict adoption patterns in different scenarios. Simulations based on technology acceptance model [12,13], diffusion of innovation model [14,15], and application of social network theories [16,17] and decision theories [18–20] have been successfully used in modeling adoption and can be useful in understanding adoption of security practices in software development. Dignum and Dignum [21] proposed to use ideas from social practice theory to support reasoning about action and planning of intelligent agents in a social context. From a decision theory perspective, the factors that influence adoption of security practices can be viewed as a developer's individual preferences and the perceived utility of using security practices. While a developer prefers using security tools only if it leads to a rationally rewarding outcome for his or her individual utility, a manager who oversees the adoption and the overall quality of the outcome may offer rewards or punishments, because the security of the end product depends on the security practices. We propose a rational decision making framework to explain the adoption of security practices by developers. The model simulates developers' preferences and decision making based on their perceptions of the advantages of security tools. The model also includes a manager who has full observability of the adoption practices and implements a sanction mechanism that enforces a policy to use security practices to meet a specified standard.

Norms and Sanction We recognize the term, norms, to describe "directed normative relationships between participants in the context of an organization" [22,23]. Norms are powerful means for regulating interactions among autonomous agents [24–26]. Social interactions form norms which are influential in dictating what behaviors are expected in a community and of the system [27,28]. Failure to comply with normative expectations is met with a sanction, which is a consequence for norm violation applied to a principal or group of principals, by a sanctioning agent. A sanction may be positive or negative and is manifested in reprimand or reward, respectively [29]. In the context of the developers' adoption environment, a norm violation would be met with a negative sanction. When an individual is singled out and censured for defecting against a norm, we recognize this as an *individual* sanction. Alternatively, when a sanction is applied to a group of individuals for actions of a subset of that group, we recognize it as a *group* sanction, also known as *collective* sanction [30].

Example 1. Consider Alex, Barb, Charlie, and Dave, who are software developers working as a team to deliver a product. Erin is their manager. Erin divides the project into multiple tasks and assigns those to the developers. Erin wants to make sure the product is delivered on time and meets the functionality and security requirements. Erin and her team use FindBugs to identify security bugs. Alex and Barb are experienced developers who use security tools. Charlie and Dave are new to the team and not aware of security practices.

Consider a scenario, where Charlie and Dave learn to use FindBugs. Everyone follows the standard security practices and delivers the product on time. Once the product is launched, its functionality and security are found as satisfactory. Erin rewards her team and encourages everyone to continue following security practices.

Now consider a second scenario where the project is time-critical, and Charlie skips executing FindBugs to deliver the product just with required functionality. This can lead to multiple alternatives: (1) Once the product is launched, it is flagged for security concerns. Erin identifies that Charlie did not follow the standard security practices on the concerned artifacts and scolds him. Charlie realizes the importance of using security tools. (2) Based on the evaluation, Erin scolds her team to use security tools in future. (3) Alex finds out that Charlie is not following the security practices and prompts him to use security tools. The illustrative example can be used to distinguish among *individual, group* and *peer* sanctions. Erin sanctioning Charlie is an example of *individual sanction* where the manager monitors and sanctions individuals who do not follow a norm. Erin scolding everyone can be considered as a *group sanction* where the sanctions are imposed on a whole group regardless of who in the group violated or satisfied a norm. Alex prompting Charlie is an example of *peer sanction* where a peer sanctions another peer in the group.

Multiagent Systems Norms are used to regulate agent behavior and facilitate collaboration in open MASs [31,32]. Savarimuthu and Cranefield [33] surveyed simulation models of norms in multiagent systems and proposed five phases of