Software Development Team Dynamics: Control, Coordination, and Secure Software

Emergent Research Forum

Emmanuel W. Ayaburi
University of Texas as San Antonio
emmanuel.ayaburi@utsa.edu

Nasim Talebi
University of Texas as San Antonio
nasim.talebi@utsa.edu

Abstract

In secure software has resulted of millions of dollars of company’s revenue as complex organizational systems are often software intensive. Researchers are still interested in the impact of the people involved, the process used and the project characteristics. This study looks at the security awareness and maturity level of those involved in the development process coupled with the management of the process to identify factors that lead to enhanced secure software development. Following a survey of prior literature, it is hypothesized that software development team members’ security awareness and maturity affect the design of software in a distributed context. Due to the nature of work of distributed team members, the relationship between team security awareness and maturity coordination on secure software design are also posited to be impacted by control types and coordination mechanisms. We contribute to literature by expanding the understanding of the process of creating secure software.

Keywords: Secure-software, coordination, control, awareness and maturity

Introduction

Information systems (IS) security is becoming pervasive problem in organizational environments (Boss et al. 2015). Organizational information systems are often complex and software intensive, with software security often considered as an after-thought (Malhotra et al. 2016). The cost of security breaches resulting from insecure software runs over several hundreds of millions of dollars and is estimated to be about 180 million dollars a year (Rosenberg, 2008), making software security a top management issue (Curtis 2016). As the concern for security of digital assets has become a global issue due to the vulnerabilities they face (Hui et al. 2016), this study seeks to understand the cause of the problem.

Numerous entities have created initiatives aimed at both software developers and security practitioners for identification, mitigation, and prevention of software vulnerabilities (MITRE 2017; NIST 2017; OWASP 2016; SANS 2017; US-CERT 2016). In a recent study of 337,742 software application assessments from large and small companies, commercial software suppliers, open-source projects, and software outsourcing, more than 60% of the software applications failed to meet the criteria of the Open Web Application Security Project (OWASP) top 10 vulnerabilities or the SANS top 25 software errors (OWASP 2016; SANS 2017). As the prevalence of software vulnerabilities continues, one strategy of addressing the issue is incentivizing the development of secure software beyond financial reward for developers (Denning 2015). The nature of the working environment with respect to the exercise of authority and nature of team interaction can be improved to ensure developers are in tune with their firms’ goal of producing secure software.

In practice, the software development process is designed to focus on functional requirements, which consist of on time delivery within budget for a product that satisfies the customer (Olmsted 2016). Software security does not fit the definition of functional requirement and is often ignored until the end of the development cycle, neglected or forgotten, accordingly treated as a non-functional requirement (Olmsted 2016; Zhioua et al. 2016). Identifying software security issues during software development as opposed to after the software has been developed could lead to 1) reduction of defects 2) early discovery of errors and 3) standardization (Malhotra et al. 2016). Therefore, exploration of how security concerns can more effectively be embedded in the development process, particularly in software development team culture, is an important area of research. This is done when security concerns are looked at from the...
interaction of people, organization and technical tools needed in the development of the software. The result is a socio-technical approach as a lens to understand the problem of insecure software design.

This research uses a socio-technical approach by considering people and organizational factor as well as technical factors to investigate the relationship between team dynamics and secure software development (Luna-Reyes et al. 2005). This is predicated on the idea that the development of secure software is impacted by a complex interaction of the people involved, the process used, and the project characteristics. The achievement of secure software depends on the concerted effort of developers in concert with project characteristics constrained by the dynamics of the team interaction. Some process issues include how to have developers follow the values and goals of the project (control dynamics), creating understanding process of managing dependencies between activities (coordination mechanisms) and people’s issues such as security information (awareness). Control and Coordination Theories are appropriate venues to examine the phenomenon (Crowston and Kammerer 1998; Espinosa et al. 2007a; Maruping et al. 2009). Thus, the purpose of this research is to investigate relationships among team coordination mechanisms and control types to factors influencing secure software development, particularly security awareness and security maturity level. Specifically, this research addresses the following research questions:

RQ1: What are effects of security maturity level and awareness on security of software developed by multiple teams?
RQ2: How do coordination mechanisms in the team impact these relationships?
RQ3: Which team control types influence secure software development?

Despite the study of control and coordination mechanisms in the management of knowledge in geographically distributed software development teams (Maruping et al. 2009), no known studies examine the phenomenon in the context of secure software development. Accordingly, this study contributes to extant literature with application of Control and Coordination Theories in the context of secure software development.

**Theoretical Background and Conceptual model**

Software development is a complex set of activities for which developers need to interact with others and share their knowledge to ultimately deliver software that meet users’ requirements. In this study, we consider the level of knowledge of the individual developers and the ability of the development team to have an effect on the potential of designing secure software. In addition we consider the effect of development team’s environmental factors (that is control type and coordination mechanism) on the developed software. Next we discuss the relevant theories and develop our four hypotheses.

**Software Security Awareness**

Software developers can be considered as the first line of defense in information security and their efforts are a tremendous asset in the struggle to reduce bugs in developed software (Langsworth 2014). Accordingly, developers who have awareness of the information security rules and regulations in the organization as well as current initiatives regarding software security (including but not limited to standards, best practices, and procedures) are the key to improving software security. General security awareness can be defined by an individual’s general knowledge about information security including organizational policy (Bulgurcu et al. 2010). This definition is extended in the context of secure software design to include general awareness of software security issues, organizational policy, and related initiatives outside of the organization designed to improve secure software design. In a study of 464 employees, Bulgurcu et al. (2010) show that employee intention to comply with security policy is significantly influenced by security awareness mediated by attitude and outcome beliefs. Based on these findings, it is postulated that software developers’ awareness of secure software design could positively influence their intention to develop secure software. Thus, it is hypothesized:

H1: The level of security awareness in a distributed software development team is positively associated with secure software development intention.

**Software Security Maturity**

Information security maturity can be defined by an organization’s ability to remain secure (Dzazali 2006a). This definition is extended to include software security, whereby software maturity is the measurement of the organization’s capability to develop secure software. Just as general information
security maturity level reflects the extent to which information security programs are implemented and thus impacts the information security level of the organization (Dzazali 2006), software development teams that are matured reflect the extent to which secure software principles are incorporated and subsequently the level of secured software developed. It is expected that members of a matured team will be proactive in incorporating the necessary mechanisms in their developed software processes and procedures in lieu of reliance on reactive procedures such as bug triggers to look for solutions, or none at all. Thus, it is hypothesized:

H2: The level of software security maturity of distributed software development teams is positively associated with secure software development intention.

**Control Theory and Mechanisms of Control**

The existence of mature software development team members who are aware of software security is not necessarily enough to guarantee the development of secure software. The process of aligning the team members to the team’s goal of producing secure software is important. Hence, Control Theory provides a lens for examination of formal and informal control moderating effects in the proposed model. In the distributed development teams’ context, control refers to the software development team leader’s attempt to ensure all team members are following the values and goals of the project (Maruping et al. 2009). Control can be categorized into formal and informal; while formal control is typically marked by performance standards and formal documentation, informal control in manifested through socializing methods (Maruping et al. 2009). This research examines moderating effects of two specific forms of formal control, and two forms of informal control. The two formal methods include outcome control and behavioral control; in outcome control reward is based on software development teams achieving their goals (i.e., if security concerns has embedded in the code), and behavioral control is based on the software development team leader evaluating processes and procedures (i.e., abiding by secure software coding standards during development). The informal control methods include clan control and self-control. Clan control is accomplished through socializing and acknowledging shared values that should be followed in a team are relevant. Self-control is defined as the extent to which members have authority to determine what to do and how to do it. Software development project teams should exhibit control while being flexible for achieving the challenging goal of delivery secure software (Batra et al. 2010). Therefore, the different types of control including outcome, behavioral, clan, and self-control are proposed to impact the strength of the relationship between team security awareness culture and secure software development intention. Therefore,

H3a: The type of control mechanism (outcome, behavioral, clan, and self-control) have an effect on the strength of the relationship between team security awareness and secure software development, such that the relationship will be strengthen when all four types of control are high.

H3b: The type of control mechanism (outcome, behavioral, clan, and self-control) have an effect on the strength of the relationship between team software security maturity and secure software development, such that the relationship will be strengthen when all four types of control are high.

**Coordination Theory and Coordination Mechanisms**

Another strategy for achieving satisfactory complex projects such as secure software development through distributed teams is through coordination mechanisms. Based on Coordination Theory, mechanisms of technical, temporal, and process coordination are evident (Crowston and Kammerer 1998; Espinosa et al. 2007). A highly coordinated project leads to higher return on investment because all developers understand the interdependencies of their roles. Ensuring that technical features of the software along with a pre-determined security process are being met by developers can enhance the possibility of an outcome of secure software. In addition, the level of coordination among distributed teams with respect to time (temporal) could impact the how the team matures in its security knowledge and compliance. Optimization of software team work with good coordination also drives team performance (Bajaj and Russell 2010). Thus, we hypothesize that these different dimensions of coordination will impact the team such that:

H4a: The type coordination (technical, temporal, and process coordination) in a distributed software development team has an effect on the strength of the relationship between team software security awareness and secure software development, such that the relationship will be strengthened when the three types of the coordination are high.
H4b: The type coordination (technical, temporal, and process coordination) in a distributed software development team have an effect on the strength of the relationship between team software security maturity level and secure software development, such that the relationship will be strengthen when three types of the coordination are high.

In summary, we proposed a research model (shown in Figure 1) of factors that influence the design of secure software by synthesizing key enablers from the security knowledge literature moderated by team management dynamics - control and coordination.

Data and Plan for Empirical Analysis
A quantitative approach using survey methodology will be used to answer the research questions of interest and empirically test the proposed hypotheses. We will be surveying members of distributed software development teams who would accept our invitation to participate in the study. A survey instrument will be created with items extracted from extant literature and used to gauge developers’ responses on the key constructs identified in this study as shown in Table 1 below.

<table>
<thead>
<tr>
<th>Construct</th>
<th>Sources</th>
</tr>
</thead>
<tbody>
<tr>
<td>Secure software security awareness</td>
<td>Bulgurcu et al. (2010)</td>
</tr>
<tr>
<td>Security Maturity Level</td>
<td>Dzazali (2006)</td>
</tr>
<tr>
<td>Coordination Mechanism</td>
<td>Gevers et al. (2006); Mohammed and Nadkarni (2011)</td>
</tr>
<tr>
<td>Control type</td>
<td>Kirsch (1996)</td>
</tr>
<tr>
<td>Perceived Secure Software</td>
<td>Jung et al. (2001)</td>
</tr>
</tbody>
</table>

Table 1: Instrument Source

Structural equation modeling (SEM) technique will be used to empirically validate the model by first validating the measurement model using confirmatory factor analysis followed by estimation of the structural model.

Expected Contribution and Future Work
This study contributes to IS research as it examines the routinization of security knowledge and attitude in distributed software development teams. In addition, it reveals how the nature of factors such as control and coordination impact the process of effective design of secure software. The ability of the software development teams to translate its security knowledge into the design of secure software is impacted by its coordination and control types. The more effective members are at coordinating activities among themselves and the existence of effective control types are major drivers of designing secure software. The present study represents an important contribution to the empirical body of literature on team knowledge coordination and geographically distributed collaboration. The effect of different types of team coordination and control in an asynchronous context in a global software organization will be explored.
REFERENCES


