Applied Ergonomics 70 (2018) 182–193

Contents lists available at ScienceDirect

Applied Ergonomics

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

The influence of commenting validity, placement, and style on perceptions of computer code trustworthiness: A heuristic-systematic processing approach

Gene M. Alarcon[a,*], Rose F. Gamble[b], Tyler J. Ryan[c], Charles Walter[b], Sarah A. Jessup[d], David W. Wood[e], August Capiola[e]

[a] Air Force Research Laboratory, United States
[b] University of Tulsa, United States
[c] CSRA, United States
[d] Oak Ridge Institute for Science and Education, United States
[e] Consortium of Universities, United States

ABSTRACT

Computer programs are a ubiquitous part of modern society, yet little is known about the psychological processes that underlie reviewing code. We applied the heuristic-systematic model (HSM) to investigate the influence of computer code comments on perceptions of code trustworthiness. The study explored the influence of validity, placement, and style of comments in code on trustworthiness perceptions and time spent on code. Results indicated valid comments led to higher trust assessments and more time spent on the code. Properly placed comments led to lower trust assessments and had a marginal effect on time spent on code; however, the effect was no longer significant after controlling for effects of the source code. Low style comments led to marginally higher trustworthiness assessments, but high style comments led to longer time spent on the code. Several interactions were also found. Our findings suggest the relationship between code comments and perceptions of code trustworthiness is not as straightforward as previously thought. Additionally, the current paper extends the HSM to the programming literature.

1. Introduction

Computer software, or computer code, is central to almost every aspect of our modern lives. However, little is known about the psychological aspects of code and the influences on why programmers choose to reuse specific code they did not create, given there are potential risks to reusing code. For example, the OpenSSL encryption many banks used for retaining secured connectivity was generally thought to be secure. However, Google's security team detected a serious issue in the software, known as Heartbleed (Grubb, 2014; Durumeric et al., 2014; CVE-2014-0160) that resulted in the theft of financial and personal data. The prevalence of code reuse and its associated risks suggest perceived code trustworthiness is an important psychological context that is under researched. Comments are an important aspect of computer code, particularly during code inspection (Porter et al., 1998). Comments should help the programmer navigate the code and understand its functional intent (Savitch, 2014). The current study sought to explore the role of code comments on how programmers perceive the trustworthiness of code.

1.1. Trust

Trust has traditionally been thought of as an interpersonal construct. However, the literature has expanded the investigation of trust to organizations (Mayer et al., 1995), automation (Lyons et al., 2016), human-machine teaming (de Visser and Parasuraman, 2011), and websites (Flavian and Guinaliu, 2006). We extend the trust literature to focus on computer code. Trust can be separated into three aspects: trust beliefs, trust intentions, and trust actions (Jones and Shah, 2016). Trust beliefs are perceptions of a referent (i.e., perceived trustworthiness of the code). It is important to note, trust beliefs are not necessarily factually based, and thus may be inaccurate. Trust intentions are the willingness to be vulnerable, to someone or something. Trust actions are the behavioral reliance on the referent, such as reusing a piece of code. The current study focused on trust beliefs and trust actions.

* Corresponding author.
E-mail address: gene.alarcon.1@us.af.mil (G.M. Alarcon).

https://doi.org/10.1016/j.apergo.2018.02.027
Received 25 June 2017; Received in revised form 19 December 2017; Accepted 27 February 2018

1.2. Code reuse

Code is a compilation of commands executed by a computer, written in a programming language, such as Java, C++, and C#. Although trained computer scientists and information technology professionals develop the software, code can get very complex and confusing as multiple systems are integrated. To expedite development, programmers often reuse code in new software. However, by integrating code previously written for other projects, programmers accept a certain level of risk and vulnerability. Programmers rely on both perfunctory perceptions and in-depth inspections of the code when deciding whether to reuse it. This cognitive evaluation is a perception of trustworthiness (Alarcon et al., 2017b). Psychological theories may help to illuminate the relationship between programmers and code. Future development of software could then encompass psychological engineering principles, improving end-product quality and efficiency in software production.

Frakes and Kang (2005) defined code reuse as, "the use of existing software or software knowledge to construct new software" (p. 529). A recent article in CNET projected that roughly 80–90% of code is reused (Hautala, 2015). Code reuse indicates programmers perceived the code as trustworthy because they have integrated it into the program. Reusing code can increase productivity (Banker and Kauffman, 1991) and possibly produce better code, as the code has been previously tested and/or repeatedly updated (Lim, 1994). Much of the literature on code reuse arose from organizational and economic perspectives, highlighting the benefits of implementing reuse on a systemic level. By reusing code assets, programmers can forego the time and effort required to rewrite software (Banker and Kauffman, 1991). Reuse also allows for increases in the flexibility and complexity of code (Babar et al., 2004). However, reusing code may carry over software bugs that were not resolved in previous implementations (Hautala, 2015). Although research has demonstrated the usefulness of reusing code, little research to date has explored how programmers perceive code trustworthiness to pursue reuse.

1.3. Heuristic-systematic processing model

Trust in code is a relatively new field of research in psychology and computer science. Indeed, no theories or process models have been developed specifically for this area of research. Instead, we leverage research from the psychological literature. The heuristic-systematic model (HSM) describes how people process persuasive messages from an information processing perspective (Chaiken, 1980; Chen et al., 1999). Although the model was created to investigate the use of persuasion in contexts such as politics and consumer purchases, it can also be extended to investigate trust in code. The HSM postulates two types of processing that influence persuasion: heuristic processing and systematic processing. A heuristic is a "strategy that ignores part of the information, with the goal of making decisions quickly, frugally" (p. 454) than more intricate strategies (Gigerenzer and Gaissmaier, 2011). Heuristic processing involves retrieving task relevant rules or "heuristics" stored in memory. Heuristics are learned knowledge structures (e.g., norms, established procedures, rules of thumb; Chaiken et al., 1989). In comparison, systematic processing involves critical examination of information pertinent to decision making (Chen et al., 1999). Systematic processing is an in-depth examination of the referent or persuasive message. The two processes can occur separately or simultaneously, depending on the situation.

One of the key tenets of the HSM is that when perceivers are processing information, they are interested in efficiency, looking to exert minimal effort in processing (Chen and Chaiken, 1999). Heuristic processing is efficient in cognitive effort and time, but typically at the expense of accuracy (Chaiken, 1980). In contrast, systematic processing requires more cognitive effort and time, but the decision rendered is more accurate (Chen et al., 1999). However, processing efficiency is not the only requirement for reaching a decision. The sufficiency principle states a perceiver has an actual level of confidence and a desired level of confidence, and actual confidence must be higher than desired confidence for a decision to be reached (Chen et al., 1999). The sufficiency threshold is when the level of actual confidence will satisfy the motives the perceiver is trying to meet. If the sufficiency threshold is met, the perceiver discontinues processing. If the threshold is not met, the perceiver will continue processing. The sufficiency threshold is not static, but can change given the motivation of the perceiver. Of the three motivations described by the HSM (Chen et al., 1999), accuracy motivation is the most relevant to programmers. In the context of programming, accuracy is key as ignoring minor issues may lead even a compiled program to fail execution tests or introduce security risks into a system. Accuracy motivation is the "open-minded and evenhanded treatment" of task-relevant information (Chen et al., 1999, p. 45). When accuracy motivation is low, the perceiver may utilize heuristic processing towards reaching a lower sufficiency threshold. In contrast, if accuracy motivation or cognitive resources are high, the perceiver may engage in systematic processing towards reaching a higher sufficiency threshold.

Although the HSM was developed to assess constructs in social psychology, it can also be used to describe trust in code. Programmers are exposed to various principles and accepted practices for writing code such as readability, organized flow, and appropriate comments (Gaddis, 2016). These practices serve as cues for programmers to use when evaluating a piece of code they did not write. Specifically, these practices facilitate heuristic processing, and such processing can be leveraged when programmers read code. Code commonly referred to as "spaghetti code" does not follow these suggested practices, either from poor initial writing or from being changed by several programmers. We propose programmers use these cognitive heuristics initially to determine if the code is worth further evaluation. A recent cognitive task analysis (Alarcon et al., 2017b) identified three factors that influence trust in computer code: reputation, performance, and transparency. Reputation was defined as trustworthiness cues based on information provided outside the code, such as source, number of reviews, and number of users of the code. Performance was defined as the capability of the code to meet the necessities of the project. Transparency was defined as the perceived comprehensiveness of the code from viewing it. The focus of the current study is on transparency, specifically code commenting. Transparency, through organized and readable code in a well thought out architecture, should evoke heuristic processing leading to trust and reuse. Comments help to facilitate this heuristic processing.

1.4. Commenting

Introductory texts on programming often mention comments as an important aspect of the code (Gaddis, 2016; Wang, 2006). Commenting is a description of the code that is not part of the functionality of the code, but is placed in the code file. Thus, comments do not directly engage with the functionality of a program. From here on, we use "comments" to refer to the descriptions of the code and "instructions" to refer to the functional code, reserving "code" to refer to the combination of instructions and comments as the whole program. Comments can act as headers to describe the overall functional expectations of code (Johnfx, 2011). They can also decompose the instructions into logical segments ("Comment," n.d.), generally highlighting where complex features may be present for easier inspection, integration, and maintenance, which may include alterations to the code given a change in domain, environment, or platform. When performing code reviews, between 20 and 35% of reviewers request additional documentation, including adequate commenting (Mäntylä and Lassenius, 2009; Beller et al., 2014). Code that requires additional documentation often appears to reviewers to have more defects (Porter et al., 1998; Thongtanunam et al., 2015), suggesting that well-commented code