The Influence of Personality on Code Reuse

Tyler J. Ryan
General Dynamics Information Technology, Dayton OH
tyler.ryan@gdit.com

Charles Walter
University of Tulsa, Tulsa OK
charlie-walter@utulsa.edu

Gene M. Alarcon
Air Force Research Laboratory, Wright Patterson AFB OH
gene.alarcon.1@us.af.mil

Rose F. Gamble
University of Tulsa, Tulsa OK
gamble@utulsa.edu

Sarah A. Jessup
Air Force Research Laboratory, Wright Patterson AFB OH
sarah.jessup.ctr@us.af.mil

August Capiola
Air Force Research Laboratory, Wright Patterson AFB OH
august.capiola@us.af.mil

Abstract

The ubiquity and necessity of computer software requires programmers to reuse extant code to keep up with increasing software demands. Researchers have started to investigate the underlying psychological processes and the programmer characteristics affecting code reuse. The present study investigated the role of programmer personality (propensity to trust, suspicion propensity) on willingness to reuse code. Programmers were recruited through Amazon's Mechanical Turk. Programmers completed propensity to trust and suspicion personality inventories and were subsequently presented with 18 pieces of computer code containing transparency and reputation manipulations. The results demonstrated that propensity to trust did not influence willingness to reuse code. However, facets of suspicion propensity did affect reuse willingness. Programmers lower in trait mal-intent perceptions and higher in cognitive activity were more likely to report they would reuse code. Implications and applications are discussed.

1. Introduction

Program comprehension concerns a programmer's understanding of, and ability to explain, computer software [1] and is an important aspect of software reuse. New tools are consistently being developed to assist in program comprehension. However, the computer science and psychology literatures have largely ignored the psychological processes underlying programmer understanding and performance. The demand for safe and secure code in a timely manner has led to a proliferation of code reuse, and psychological theories that can help to elucidate the relationship between the programmer and the software, leading to better development and review practices. In the present study, we examined programmer propensity to trust and the suspicion propensity facets of mal-intent and cognitive activity as individual differences that influence perceptions of code written by someone else. In addition, we manipulated the code itself according to factors emphasized by Alarcon et al. [2], namely, readability, organization, and source of the code. As such, the current study expands the literature on program comprehension by modeling both the programmer and the referent (software) in the software reuse context.

We propose the following hypotheses:

H_1: Propensity to trust has a positive effect on code reuse, such that those with higher propensity to trust will endorse the code for reuse more than those lower in propensity to trust.

H_2: Perceived mal-intent has a negative effect on code reuse, such that those with lower perceived mal-intent will endorse the code for reuse more than those higher in perceived mal-intent.

H_3: Cognitive activity has a negative effect on code reuse, such that those with lower cognitive activity will endorse the code for reuse more than those higher in cognitive activity.

2. Background

Frakes and Kang [3] define code reuse as "the use of existing software or software knowledge to construct new software" (p. 529). Reusing code can increase the flexibility and complexity of the code [4], while reducing the time it takes to create the code [5]. Reusing code also indicates the programmer understands the code [1]. Research in the

URI: https://hdl.handle.net/10125/60016
ISBN: 978-0-9981331-2-6

computer science literature [6] has started exploring how programmers read code with the use of eye trackers, showing that programmers who spend more time scanning the code before taking a more in-depth look are better at defect detection. Other research has focused on programmer experience as a factor influencing whether they notice vulnerabilities or bugs within a program [6, 7]. The aforementioned research indicates a trend in the literature of focusing on psychological processes rather than strictly management or productivity concerns with code reuse. These studies and others [7, 8, 9, 10, 11] have started focusing on the psychological perceptions and processes as aspects of program comprehension and reuse. Additionally, research has begun to explore the role of programmer's perceptions of trustworthiness in code.

2.1. Trust

Trust is a multifaceted process that can be broken up into several distinct components, namely trust beliefs, trust intentions, and trust actions [12]. Trust beliefs are the trustor's perceptions of a person's trustworthiness. Trust intentions are a willingness to be vulnerable to the referent, such as the willingness to trust management [13] or a coworker. Trust actions are the actual behaviors the trustor performs, such as not monitoring a coworker or reusing a class of code from a coworker. There are also dispositional facets of trust, such as one's propensity to trust, that can influence trust beliefs, intentions, and actions. For a conceptual diagram of the trust process, refer to Mayer, Davis, and Schoorman [13].

Trust has traditionally been thought of as strictly an interpersonal process. However, recent interest has extended trust research to automation [14], trust in robots [15], and perceptions of trustworthiness of computer code [8, 16]. Research in the computer science literature has explored aspects of the trust process, although the research was not labeled as such. Kelly and Shepard [17] explored the number of coding errors generated when code inspections were performed in a group setting versus individual setting. Their results indicated group inspection of code resulted in fewer defects than individuals inspecting code. Although the research did not specifically state they were examining psychological variables, they were exploring social influences on detecting vulnerabilities in code. Albayrak and Davenport [18] degraded code naming conventions and indentation to explore the influence on programmers' detection of functional defects. When the code was degraded in both aspects, participants reported a higher number of defects in the code that were not actually defective (false positives) and a lower number of actual defects were detected (misses). This study illustrated that changing aspects of the referent (i.e., the code) can influence trust.

Recently, the field of psychology has taken an interest in how programmers perceive and trust code. Alarcon et al. [2] performed a cognitive task analysis (CTA) to determine what psychological factors influenced the perceptions of code trustworthiness and the decision to reuse code. In the CTA, three factors emerged: reputation, transparency, and performance. The reputation factor concerns aspects of the code that are obtained through external information, such as research and professional network. Reputation can be influential for code trustworthiness and reuse even without directly examining the code. The transparency factor concerns aspects of the code that influence a programmer's ability to comprehend what the code is doing. This includes code organization, readability, architecture, and style. Lastly, the performance factor concerns the capacity of the code to meet the necessities of the current project, such as the code flexibility, freedom from errors, and efficiency. Alarcon et al.'s CTA also found the programmer's environment has an impact on their perceived trustworthiness and their likelihood to reuse the code. That is, a programmer may trust and reuse code differently than a programmer in a situation with higher consequences of failure. For example, a programmer working with online education software containing no personally identifying information may trust and reuse code differently compared to a programmer working on code relevant to a nuclear reactor. Although it was not explored directly, Alarcon et al.'s [2] model of code trustworthiness included individual difference variables. That is, variables such as personality and past experience may impact one's perceived trustworthiness of code and subsequent reuse.

Recently, researchers [8] hypothesized an information processing model of code trustworthiness. In their paper, they describe a dual process model, with heuristic processing and systematic processing underlying trustworthiness perceptions of code and code reuse. Heuristic processing is automatic and less effortful, in which the programmer assesses the referent (i.e., code) with quick judgements such as rules of thumb, standard operating procedures, or norms. Heuristic processing saves time, but is not as accurate as more effortful processing. In comparison, systematic processing is an active cognitive effort. When engaging in systematic processing, programmers offer a more in-depth cognitive assessment of the referent, increasing scrutiny and attention to detail. Heuristic and systematic processes are not mutually exclusive. Indeed, research has demonstrated that one process can later influence the decision to perform the other [2]. Both heuristic and systematic processing influence a programmer's comprehension of code. In addition, it is hypothesized programmers work on a select-out process, abandoning code that appears too difficult to comprehend [8, 19].

2.2. Code Trustworthiness

Research on trustworthiness perceptions of code has illustrated aspects of the code that influence trust perceptions and program comprehension. Manipulating the code's comments had an influence on coder's trustworthiness perceptions and time spent on the code, despite no manipulations to the source code [19]. Another study found readability, organization, and source of the code influenced programmer's perceptions of trustworthiness and time spent on the code [20]. As the three factors of the code were degraded, trustworthiness perceptions and time spent on code changed, despite the fact that all the code in the study compiled and was free from vulnerabilities. These studies illustrate the importance of the trust process in program comprehension.

The aforementioned studies have demonstrated that relevant code factors can influence programmer's trust beliefs, but trust beliefs are different than trust intentions [12]. Trust beliefs are perceptions of the referent. In the programming context, the referent is computer code. Trust intentions are a willingness to make oneself vulnerable, such as reusing code written elsewhere or by other developers. The interpersonal trust literature has demonstrated that trust beliefs influence trust intentions and subsequent behaviors [21]. However, there is additional variance that is unaccounted for as beliefs, intentions, and behaviors do not correlate highly enough in the literature to indicate convergent validity of the two constructs [22]. As such, although readability, organization, and source have been related to trustworthiness perceptions of the code, no research to date has explored the influence of these three factors on intentions to reuse code.

For the current study, we used previously established stimuli of transparency and reputation [20], manipulating the transparency of the code by degrading readability, organization, and reputation. Readability was defined as the grammar of the code. Code that was low in readability did not follow the rules and norms for formatting code. Degradations to readability included: 1) misuse of cases, 2) misuse of indentation, 3) misuse of braces, and 4) improper line length and line wraps. Thus, for the readability manipulations, code was degraded aesthetically. In contrast, organization degradations affected the structure of the code. Less organized code was difficult to comprehend due to a poor arrangement. Organization degradations included: 1) statements requiring unnecessary additional review, 2) ambiguous control flow, 3) improper exception handling, 4) poor grouping of methods, and 5) misuse of declarations. Reputation was also manipulated in the stimuli. Code stimuli was labeled as "reputable" or "unknown." The two reputation conditions were chosen so participants would rely on their own cognitive heuristics for interpretation. If code was labeled coming from a specific source (i.e., Microsoft), individuals may have different attitudes towards the target, some good and some bad. These attitudes may impact the heuristics used to evaluate the code, thus influencing trustworthiness and subsequent reuse intentions. As such, simply stating if code was reputable or unknown allowed the user's heuristic to drive the trust intention, which reduced error variance.

2.3. Personality

Personality is defined as a set of characteristic behaviors, cognitions, and emotional patterns that are a result of both biological and environmental factors [23]. Trait theories posit that personality is composed of individual difference variables, or traits, that are relatively stable. These traits have a long history of being related to behaviors from work performance [24] to health behaviors [25]. Personality has also been explored in the computer science literature. Cho and colleagues [26] found neuroticism and agreeableness were associated with perceived trust, risk of a phishing scam, and decision performance. However, no research has linked personality to code reuse intentions. Aspects of personality can influence cognition. Researchers have found that personality influences the cognitive responses to stressors [27]. Personality also influences the formation of trust perceptions [22, 28]. Specifically, personality influences initial perceptions of others, but as participants have more interaction with a referent, personality plays less of a role.

In our study, we examined the influence of two relevant traits – propensity to trust and suspicion propensity – on perceived trustworthiness and intentions to reuse code. Propensity to trust is a stable personality trait that reflects one's general expectancy of the trustworthiness of others and a general willingness to trust others [13, 29]. Propensity to trust has demonstrated a relationship with trust behaviors, trust intentions, and trust beliefs [2, 8, 22]. Although propensity to trust has demonstrated a relationship within interpersonal trust intentions and behaviors [21], it remains to be seen if this personality trait influences intentions to reuse code. Research on propensity to trust indicates general expectancies influence trust intentions. Specifically, people higher in propensity to trust are more likely to trust automation in experimental [30] and real-world settings [14]. However, propensity to trust is less contextually influential as the trustor becomes more familiar with the referent [12, 28]. It remains to be seen if propensity to trust influences the decision-making process after controlling for perceptions of the code (i.e., the manipulations mentioned above). The current study seeks to remedy this shortcoming in the literature.

Suspicion propensity is a newer personality trait in the literature and has been defined as a "tendency to concurrently (i) perceive the potential for mal-intent, (ii) be uncertain about the meaning of the information, and (iii) engage in cognitive activity that attempts to explain, or generate alternative possible meanings for, that information" [31, p. 13]. Although suspicion propensity is a new construct in the psychological literature, it has clear implications for the trust process in code reuse. People that display a tendency to perceive an environment as hostile will be more reluctant to reuse code, as they perceive ill-will in the environment. In addition, programmers that are confronted with code that seems suspicious may engage in additional cognitive effort to determine whether they should use the code or not. The uncertainty dimension of suspicion can also have negative effects on reuse intentions. For instance, a programmer may have doubts about the referent code and perceive possible vulnerabilities within the code, which may influence their willingness to reuse the code. In the present context, all code evaluation initially involves uncertainty. That is, the programmer is tasked with a demand (i.e., evaluate code) that is laden with uncertainty. Therefore, we focused on mal-intent and cognitive activity because the code evaluation task does not inherently evoke these components of suspicion.

The authors are not aware of any research in the psychology or computer science literatures that has sought to associate stable trust beliefs (personality) with intentions to reuse code. As mentioned above, personality has been linked to a variety of intentions and behaviors in the psychology literature. For example, propensity to trust has been related to intentions to trust in occupational contexts [21] and interpersonal trust behaviors in economic games [32]. Personality variables may influence the cognitive comprehension of the variables through the process mentioned in past research [8]. As such, the current study seeks to extend the personality literature to trust intentions in code reuse.

Table 1. Readability degradations (adapted from [20])

1. Misuse of case	a) For packages
	b) For classes and interfaces
	c) For methods and variables
	d) For constants
2. Misuse of braces	a) Line break before an opening brace
	b) No line break after an opening brace
	c) No line break before a closing brace
	d) Line break after a brace that precedes an else
	e) Missing a space before an opening or closing brace
3. Misuse of indentation	a) Improper indentation given code position
	b) Inconsistent indentation
4. Improper line length and line wrapping	a) Unnecessarily exceeds character limit without wrapping
	b) Missing blank lines to indicate logical grouping
	c) Use of too many and unnecessary blank lines

3. Method

A total of 127 programmers were recruited for an online study via Amazon Mechanical Turk (MTurk). For inclusion in the study, participants had to know Java and have at least 3 years of programming experience. We excluded any participant that did not have 3 years of programming experience or any variance in their trustworthiness scores, which indicated the participant was not taking their performance in the study seriously. We were only interested in experienced programmers, therefore we excluded any participant that reported "student" as their profession. This left a total of 73 participants in the final sample.[1] Many Mturk tasks provide small payments. However, given the task for the present study and the targeted population, participants were paid $10.00 USD. The sample had a mean age of 29 years (range 20 to 54), an average experience of 7.7 years (range 3 to 32), was primarily male (89%), and 41% listed Java as their primary programming language.

The study was a 3 x 3 x 2 factorial within-subjects design, consisting of the readability, organization, and source degradations derived from a previous study [20]. Readability and organization consisted of "High", "Medium", and "Low" quality levels, and source consisted of "Unknown" and "Reputable" levels. A description of the code degradations is provided below.

3.1. Stimuli

Our study focused primarily on readability and organizational degradations to Java code. These degradations were derived from Java Style Guides [33, 34, 35], an extensive search of stackoverflow.com's style sections, and a commonly used undergraduate textbook [36] intended to teach new programmers correct Java style.

Table 1 illustrates readability degradations. We categorized degradations into a total of 4 primary and 14 subcategories. For example, misuse of case is subdivided into misuse of case for packages, classes and interfaces, methods and variables, and constants. Each of these categories has a unique case standard in Java, meaning that what may be correct for a constant is incorrect for a package or method.

Programmers often use consistent brace placement to allow them to find the beginning and end of methods and blocks quickly and easily. Java style guides are very clear regarding accepted use of braces. Thus, we divided misuse of braces into a line break before an opening brace, no break after an opening brace, no break before a closing brace, a line break after a brace preceding an else, and missing space

[1] A power simulation study was conducted for a GEE with small to moderate effect sizes for the factors and covariates and *N* = 73, suggesting power between .84 and .97 for the personality covariates of concern. Details are available from the first author.

before an opening or closing brace. Note that not all of these degradations are always a problem. For example, in the case on one line ‘if’ statements, it is acceptable to not have a line break after an opening brace or before a closing brace. However, in this case, it is important to include a space between the braces and the single line of code.

Misuse of indentation is a noticeable readability degradation. While indentation does not affect the code function in Java, a lack of indentation can confuse a reviewer, making them think the code they are examining is associated with a different code block or method. Two ways to degrade readability are with improper indentation for the codes position and through inconsistent indentation throughout a code block.

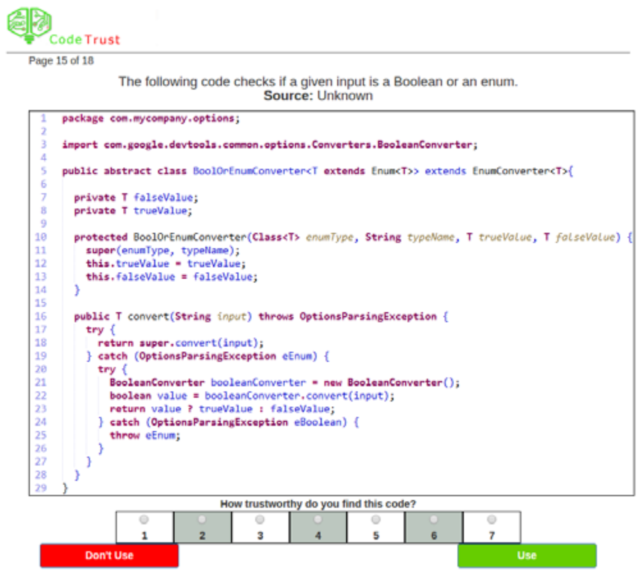
Improper line length and line wrapping can introduce some frustration during code review. Java standards indicate that after 80 characters the code should wrap to the next line. It is not required from compilation, but can cause issues with review depending on the code display method (e.g., terminal, IDE, etc.). This degradation was broken down into unnecessarily exceeding the character limit, missing blank lines for grouping, and using too many blank lines to group.

Table 2. Organization Degradations (adapted from [20])

1. Poor grouping of methods	a) Any form
2. Misuse of declarations	a) Import statements used improperly
	b) More than one variable per line
	c) Variables not initialized as soon as possible
	d) Overuse of public instance and class variables
3. Ambiguous control flow	a) Improper, unnecessary, or confusing use of “break” or “continue”
	b) Unnecessary or confusing nesting of blocks
	c) Multiple function calls or unnecessarily grouping block on one line
	d) Switch statement does not have a default case
	e) Switch statement with no “break” does not comment explicit continuation to next statement group
4. Improper exception handling	a) Any form
5. Statements unnecessarily require additional review	a) Compressed if statements
	b) Unusual return statements
	c) Multiple classes
	d) Inconsistent blocks

Table 2 shows the organization degradations, separated into 5 primary and 15 sub categories. Poor grouping of methods may indicate that the code has been modified by multiple programmers who are inconsistent and prone to mistakes. Misuse of declarations could indicate a coder is likely a novice and does not understand the conventions associated with the declaration they are using. Ambiguous control flow can reduce code comprehension. Improper exception handling is considered poor programming practice in any form, causing difficulty during debugging. Finally, statements which require extra review (i.e., a “second look”) can mean initial confusion in trying to determine the initial intent of the original programmer(s).

Participants reviewed 18 artifacts of Java code. Each artifact was displayed on its own page with a brief description of what the class was intended to do at the top of the page. The source manipulation was also displayed at the top of the page. Figure 1 illustrates a sample page from the study. All artifacts are available for download from [https://doi.org/10.1080/23311908.2017.1389640].



The following code checks if a given input is a Boolean or an enum.
Source: Unknown

```java
package com.mycompany.options;

import com.google.devtools.common.options.Converters.BooleanConverter;

public abstract class BoolOrEnumConverter<T extends Enum<T>> extends EnumConverter<T>{

  private T falseValue;
  private T trueValue;

  protected BoolOrEnumConverter(Class<T> enumType, String typeName, T trueValue, T falseValue) {
    super(enumType, typeName);
    this.trueValue = trueValue;
    this.falseValue = falseValue;
  }

  public T convert(String input) throws OptionsParsingException {
    try {
      return super.convert(input);
    } catch (OptionsParsingException eEnum) {
      try {
        BooleanConverter booleanConverter = new BooleanConverter();
        boolean value = booleanConverter.convert(input);
        return value ? trueValue : falseValue;
      } catch (OptionsParsingException eBoolean) {
        throw eEnum;
      }
    }
  }
}
```

Figure 1: Example stimulus presented to subjects (adapted from [20])

To craft the stimuli shown to the programmers, we selected a total of 18 code artifacts from open source software projects hosted on github.com. Each stimulus was chosen from a highly reviewed project. Each sample was first cleaned to ensure they all followed the Java guidelines or accepted practices. Once cleaned, each code sample was degraded to assign a level of low, medium, or high readability and organization.

An example of the stimuli as seen by participants is shown in Figure 1, as found in Alarcon et al.’s study [2]. This code sample includes two organization degradations, specifically O3.b and O4. These both refer to the multiple layers of try/catch blocks being used as control flow, forcing the code to have errors which are then ignored intentionally. If there is an error it is likely that it would be seen as part of the normal control flow, making this small code sample difficult to debug.

Below the code sample, a Likert scale was displayed asking the subjects to rate the trustworthiness of the code from 1 (Completely untrustworthy) to 7 (Completely trustworthy). Upon making a trustworthiness rating, participants could select if they would use the code. The two ratings allowed for a user to find a sample untrustworthy but still deem the code usable. Should a subject choose not to use the code, a comment box would appear requesting them to elaborate on why they did not trust the code sample.

3.2. Measures

3.2.1. Propensity to Trust. Propensity to trust was assessed using Mayer and Davis' [37] propensity to trust scale. The scale consists of 8 items assessing general propensity to trust. An example item is "Most experts tell the truth about the limits of their knowledge." Participants responded to the items using a Likert-type response scale ranging from 1 (Strong Disagree) to 5 (Strongly Agree). The internal consistency of the scale for the current study was 0.71.

3.2.2. Suspicion Propensity. Suspicion propensity was assessed using Calhoun et al.'s [31] Suspicion Propensity Index. The scale consists of 11 scenarios that involve uncertainty, as all suspicion must involve a degree of uncertainty. Participants respond to the scenarios on a Likert-type scale of 1 (Not at all accurate) to 5 (Very accurate) in agreement to interpretations of the scenario. Two of the four response items are suspicion propensity, namely cognitive activity and mal-intent. For example, a response to a scenario of not getting a job of "I would follow up with someone at the company and request more information about why I wasn't chosen" would indicate cognitive activity. For the same scenario, a response of "I would wonder if there was someone at the company who I had contact with who purposely wanted to keep me from getting the job" would indicate cognitive activity and mal-intent. The internal consistency of the cognitive activity subscale for the current study was 0.77. The internal consistency of the mal-intent subscale for the current study was 0.76.

3.2.3. Use. Participants were asked to decide whether they would "Use" or "Don't Use" the code. Although the scale is only one item, single item measures have demonstrated appropriateness when the item is not ambiguous and to avoid response fatigue [38].

3.3. Procedure

Participants were recruited from MTurk. After participants accepted the HIT, they were directed to a website that provided a brief description of the study. After reviewing the description, participants gave consent by clicking the 'next' button. Participants then completed background surveys including the personality measures and demographics surveys mentioned above. Upon completion of the surveys, participants were shown 18 pieces of code artifacts, responded to whether they would use the code or not, and rated the code on perceived trustworthiness with a 7-point Likert-type scale. Participants were also able to provide remarks about each code artifact they decided they would not use. Upon completion of the survey, participants were shown a debriefing message, thanked for their time, and given a code to enter into MTurk to receive payment. Participants were compensated within 3 working days for their participation.

3.4. Data Analysis

We utilized the Generalized Estimating Equation (GEE) approach for all analyses. Traditional repeated measures analysis of variance (RM ANOVA) was not used for three reasons. First, the data was not normally distributed as the outcome is dichotomous, which is a key assumption of ANOVA. GEEs do not have an assumption of normality. Second, RM ANOVA is typically used for longitudinal analyses as the measurements are assumed to be uniformly correlated over time. The GEE allows the researcher to determine the best correlation structure for the data (i.e., uniform, autoregressive, unconstrained, or uncorrelated).

We created a full model with propensity to trust, the suspicion propensity facets, and the code manipulations. Extensive interpretation of code manipulation effects are beyond the scope of this paper, as we focused on investigating the potential influence of personality on intentions to reuse code. For a more in-depth discussion on the effects of code manipulations on subsequent trust intentions and behaviors in coding contexts, see [2, 8, 11].

4. Results

We conducted a point-biserial correlation to determine the relationship between trustworthiness and intentions to reuse across all stimuli. The correlation was statistically significant, $r = 0.60$, $p < .001$. However, it should be noted the correlation, although strong, was not strong enough to indicate trustworthiness and reuse intentions are the same construct [39]. This supports the previous literature that the trust process is composed of trust beliefs, intentions, and behaviors, which are separate, albeit related, constructs [22]. Table 3 illustrates the reuse intentions for participants receiving each manipulation. Interestingly, the majority of participants intended to reuse code across conditions in the current study.

4.1. Full Model

To test H_1-H_3, we included personality variables along with the code manipulations in the model. The model was fit with an 'exchangeable' correlation structure providing the best fit, QIC (9) = 1080.28. Propensity to trust, [Wald χ^2 (1, N = 1314) = 0.20, β = -0.10, p = 0.653] was not a significant predictor of reuse intentions, indicating no support for H_1. Both mal-intent [Wald χ^2 (1, N = 1314) = 7.32, β = 0.51, p = 0.007] and cognitive activity [Wald χ^2 (1, N = 1314) = 7.02, β = -0.50, p = 0.008] facets of suspicion both predicted reuse intentions. While H_3 was supported, the direction of the effect for mal-intent does not support H_2 as it led to an increased likelihood of code endorsement. The intercept was still statistically significant when all the personality factors were included in the model, [Wald χ^2 (1, N = 1314) = 6.32, β = 2.14, p = 0.014].

The main effects of the manipulations in the study were also significant. Although we refrain from interpreting the results, we include them as they are important aspects of the final model. The model effect of readability was statistically significant [Wald χ^2 (2, N = 1314) = 8.66, p = 0.013], such that code higher in readability (M = 0.90, SE = 0.02) was more likely to be reused than code low in readability (M = 0.84, SE = 0.022), z = 2.87, p = 0.012. The differences between both high and medium (M = 0.85, SE = 0.02) readability, z = 2.02, p = 0.108, and low and medium readability, z = -0.64, p = 0.798, were not significant. Organization also demonstrated a significant model effect [Wald χ^2 (2, N = 1314) = 19.29, p < 0.001], such that code high in organization (M = 0.81, SE = 0.09) was less likely to be used than low organization (M = 0.89, SE = 0.02), z = -3.56, p = 0.001 or medium organization (M = 0.88, SE = 0.02), z = -3.55, p = 0.001) code. No differences were found between low and medium organization, z = 0.33, p = 0.942. Lastly, a main effect of source was statistically significant [Wald χ^2 (1, N = 1314) = 6.57, p = .010] such that code from a reputable source (M = 0.90, SE = 0.01) was more likely to be reused than code from an unknown source (M = 0.83, SE = 0.02), z = 2.58, p = 0.010. Since the focus of this study is the effect of personality on trust, we refer the reader to Alarcon et al. [20], who found similar results, for discussion on the effects of the code manipulations.

5. Discussion

The current study explored personality as a predictor of intentions to reuse code. This is the first study the authors are aware of that has attempted to relate personality to intentions to reuse code. There is a long line of research in the psychology literature that has demonstrated personality's influence on decision making processes (e.g., [40, 41]). The constructs of propensity to trust and suspicion propensity offer insight into how programmers view code for reuse. The two constructs are generalized beliefs about the world, albeit one more positive than the other.

Propensity to trust did not account for significant variance in reuse endorsement. This is interesting for two reasons. First, propensity to trust represents a general expectancy about others and a willingness to trust others [13, 29]. The willingness to trust others as conceptualized by the scale may only be associated with trust in people. In other words, participants focused on the referent (i.e., code) rather than relying on heuristics about people. Second, programmers may be taught to be critical of software so as to avoid reusing code that is malicious. Thus, other constructs such as suspicion may play a larger role on reuse intentions and subsequent reuse behaviors.

Both mal-intent and cognitive activity accounted for variance in the intercept in the model. However, neither the effect of mal-intent nor cognitive activity were in an expected direction. The cognitive activity aspect of suspicion was negatively related to reuse intentions. Participants that had a natural propensity to perform in-depth processing were actually less likely to endorse the code for reuse. Albayrak and Davenport [18] provide insight, finding that degrading the code leads to higher rates of false

Table 3. Counts of intentions to use by code manipulations

			Readability					
			High		Medium		Low	
			Use	Don't Use	Use	Don't Use	Use	Don't Use
Organization	High	Unknown	66	7	59	14	50	23
		Reputable	56	17	61	12	58	15
	Medium	Unknown	61	12	59	14	58	15
		Reputable	67	6	70	3	67	6
	Low	Unknown	69	4	54	19	63	10
		Reputable	69	4	65	8	65	8

positives. We could suggest that participants whom scrutinize over the code were able to pick up on several grammatical errors in the text, despite it being functionally sound. Perceiving more flaws in the readability and organization of the text would thus lead coders to reuse less than those lower in cognitive activation whom do not scrutinize. Whereas previous research has explored aspects of the referent (i.e., code) that influence systematic processing [20], the current study illustrates that aspects of personality can also influence processing effort. Additionally, mal-intent was positively related to reuse intentions. Individuals that perceive the world around them as more hostile with intent to harm them were oddly more willing to be vulnerable to a context-dependent referent (i.e., higher trust intentions). It may be that those who are more apt to perceive hostility in the environment are more vigilant in detecting potential harm in the referent. As such, the participant will find there are no observably harmful aspects of the code, (e.g., viruses, functional flaws, etc.), thus leading to a higher reuse intentions. While this seems to contradict our interpretation of the effects of cognitive activation on reuse, we suggest this provides new insight into the suspicion construct. It may be that persons high in both cognitive activation and perceived mal-intent are less predictable in the decisions that they make. Further, the current study is limited in that it did not investigate the interaction between cognitive activity and perceived mal-intent. Individuals high in one suspicion facet, but not the other, may interpret code differently than those high (low) in both facets. If this were the case, treating the suspicion propensity facets as a single suspicion variable could hinder the predictive utility of the construct. Future research should investigate the interaction between suspicion facets and their influence on trust intentions.

The current study explored what code manipulations and personality traits influence code reuse. The study found results for reuse intentions similar to results for trustworthiness perceptions from a previous study [20]. In a prior study, counting participant remarks on code manipulations found that as readability was degraded, participants remarked on the poor readability and the problems associated with it, which led to decreased trustworthiness. In contrast, as organization was degraded, participants also remarked about the degradations, but trustworthiness increased. This finding supports previous research on the effects of organization manipulation and reuse intention [11]. Researchers [20] have noted that with the current manipulations to the code, trustworthiness perceptions had interaction effects. If code was from a reputable source but was disorganized, then participants were willing to spend more time on the code. This increase in time spent on the code led to a deeper understanding of the code, as all code compiled and was free from errors. The same underlying process may be occurring with reuse assessments. As participants spent more time on the code, they became more familiar with the code. Thus, reputable source led to an increased probability of using the code, as expected.

The study is not without limitations. The experimental platform did not allow participants to download the code and see how it performed. Indeed, in the previous studies using the same stimuli [20], participants noted they would prefer to download and explore/test the code themselves. However, this experiment explored the first view stage of programmer's perceived trustworthiness / intentions to reuse the code, when programmers first view the code then decide whether it is worth their time and effort to continue effortful processing. Future research should explore how programmers inspect the code in subsequent phases of the model, as well as what they do after downloading the code for testing.

It is worth noting that participants intended to reuse the code 85% of the time. Only 197 decisions to not reuse were recorded in the current study, out of a total of 1,314 decisions. This may be due to the participants being told that all code compiled at the beginning of the experiment. However, Albayrak and Davenport [18] found degrading the code should influence perceptions, namely false positives in defect detection. The high rate of reuse intention could possibly indicate the participants were not paying attention to the task. Future research should implement an attention check (e.g., single item with a predetermined correct answer) to detect poor responding, as well as ensure participants possess the requisite knowledge and experience to complete the task. The reuse intention rate could also indicate that the heuristic threshold is lower in the first view phase. That is, reuse intention may really be a decision to want more information about the code through subsequent testing before implementing the code in another architecture or task. Participants on the MTurk website may have a lower threshold for acceptance as they may not be programming for a company, which may have strict rules about reuse. The placement of the use/don't use buttons underneath the trustworthiness scale could have unduly increased the correlation between trustworthiness ratings and reuse endorsement, biasing the reusing outcomes. We do not expect this biasing effect to be large, as trustworthiness and trust have been found to be highly correlated in previous research [21]. However, future studies should separate the two indices to prevent unnatural anchoring. Furthermore, additional indicators of trust/intention to reuse the code should be added for greater measurement precision.

Although cognitive activity and mal-intent accounted for significant variance in the intercept, the intercept was still statistically significant indicating other individual differences may account for aspects of code comprehension and willingness to reuse code. Aspects such as experience, cognitive ability, or conscientiousness (e.g., a scrupulous, careful, hardworking personality) [42] may also account for willingness to reuse code.

Acknowledgments

This research was funded in part by the Air Force Research Laboratory (Contract FA8650-16-D-6616/0003. The findings and conclusions in this report are those of the authors and do not necessarily represent the official position of the Air Force.

6. References

[1] T.J. Biggerstaff, B.G. Mitbander, and D. Webster, “The concept assignment problem in program understanding,” In *Proc. of the 15th International Conference on Software Engineering*, 1993, pp. 482-498.

[2] G.M. Alarcon, L.G. Militello, P. Ryan, S.A. Jessup, C.S. Calhoun, and J.B. Lyons, “A descriptive model of computer code trustworthiness.” *Journal of Cognitive Engineering and Decision Making*, vol. 11, pp. 107-121, 2016.

[3] W.B. Frakes and K. Kang, “Software reuse research: Status and future.” *IEEE transactions on Software Engineering*, vol. 31. Jul. 2005, pp. 529-536.

[4] M.A. Babar, L. Zhu, and R. Jeffery, “A framework for classifying and comparing software architecture evaluation methods,” In *Proc. Software Engineering Conference*, 2004, pp. 309-318.

[5] W.C. Lim, “Effects of reuse on quality, productivity, and economics.” *IEEE software*, vol. 11, pp. 23-30. Sept. 1994.

[6] B. Sharif, M. Falcone, and J.I. Maletic, “An eye-tracking study on the role of scan time in finding source code defects,” *Proc. of the Symposium on Eye Tracking Research and Applications*, 2012, pp. 381-384.

[7] C.M. Hoadley, M.C. Linn, L.M. Mann, and M.J. Clancy, “When, why and how do novice programmers reuse code,” In *Empirical Studies of Programmers: Sixth Workshop*, pp. 109-129. Intellect Books, 1996.

[8] G. M. Alarcon and T. J. Ryan, “Trustworthiness perceptions of computer code: a heuristic-systematic processing model,” In *Proc. of the 51th Hawaii International Conference on System Sciences*, 2018, pp. 5384-5393.

[9] M.S. Hsieh and E. Tempero, “Supporting software reuse by the individual programmer,” In *Proc. of the 29th Australasian Computer Science Conference*, 2006, pp. 25-33.

[10] R.J. Norton, D. Banks, and B. Lee, “Reuse of personal software assets: Theories, practices and tools,” 2003.

[11] T.J. Ryan, C. Walter, G.M. Alarcon, R.F. Gamble, S.A. Jessup, and A.A. Capiola, “Individual differences in trust in code: The moderating effects of personality on the trustworthiness-trust relationship,” In *Proc. International Conference on Human-Computer Interaction*, 2018, pp. 370-376.

[12] S.L. Jones and P.P Shah. “Diagnosing the locus of trust: A temporal perspective for trustor, trustee, and dyadic influences on perceived trustworthiness,” *Journal of Applied Psychology*, vol. 101, Sept. 2015, pp. 392-414.

[13] R.C. Mayer, J.H. Davis, and F.D. Schoorman, “An integrative model of organizational trust,” *Academy of Management Review*, vol. 20, Jul. 1995, pp. 709-734.

[14] J.B. Lyons, N.T. Ho, W.E. Fergueson, G.G. Sadler, S.D. Cals, C.E. Richardson, and M.A. Wilkins, “Trust of an automatic ground collision avoidance technology: A fighter pilot perspective,” *Military Psychology*, vol. 28, Jul. 2016, pp. 271- 277.

[15] P.A. Hancock, D.R. Billings, K.E. Schaefer, J.Y. Chen, E.J. De Visser, and R. Parasuraman, “A meta-analysis of factors affecting trust in human-robot interaction,” *Human Factors*, vol. 53, pp. 517-527, Oct. 2011.

[16] C. Walter, R.F. Gamble, G.M. Alarcon, S.A Jessup, C.S. Calhoun, “Developing a mechanism to study code trustworthiness,” In *Proc. of the 50th Hawaii International Conference on System Sciences*, 2017, pp. 5817-5826.

[17] D. Kelly and T. Shepard, "An experiment to investigate interacting versus nominal groups in software inspection," *Proc. of the conference of the Centre for Advanced Studies on Collaborative Research*, 2003, pp. 122-134.

[18] Ö. Albayrak and D. Davenport, "Impact of Maintainability defects on Code Inspections," *Proc. of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement-ESEM'10*, 2010, pp. 1-50.

[19] G.M. Alarcon, R.F. Gamble, T.J. Ryan, C. Walter, S.A. Jessup, D.W. Wood, and A. Capiola. “The influence of commenting validity, placement, and style on perceptions of computer code trustworthiness: a heuristic-systematic processing approach.” *Applied Ergonomics*, vol. 70, 2018, pp. 182-193.

[20] G.M. Alarcon, R. Gamble, S.A. Jessup, C. Walter, T.J. Ryan, D.W. Wood, and C.S. Calhoun, “Application of the heuristic-systematic model to computer code trustworthiness: The influence of reputation and transparency,” *Cogent Psychology*, vol. 4 (1), Advance online version, 2017.

[21] J.A. Colquitt, B.A. Scott, and J.A. LePine, “Trust, trustworthiness, and trust propensity: A meta-analytic test of their unique relationships with risk taking and job performance.” *Journal of Applied Psychology*, vol. 92, Jul. 2007, pp. 909-927.

[22] G.M. Alarcon, J.B. Lyons, J.C. Christensen, M.A. Bowers, S.L. Klosterman, and A. Capiola, “The role of propensity to trust and the five-factor model across the trust process,” *Journal of Research in Personality*, vol. 75, 2018, pp. 69-82.

[23] P.J. Corr and G. Matthews, Eds., The Cambridge handbook of personality psychology, New York: Cambridge University Press, 2009, pp. 748-763.

[24] M.R. Barrick, M.K. Mount, and T.A. Judge, “Personality and performance at the beginning of the new millennium: What do we know and where do we go next?” *International Journal of Selection and Assessment*, vol. 9 Mar/Jun. 2001, pp. 9-30.

[25] J.M. Malouff, E.B. Thorsteinsson, and N.S. Schutte, “The five-factor model of personality and smoking: A meta-analysis,” *Journal of Drug Education*, vol. 36, Mar. 2006, pp. 47-58.

[26] J.H. Cho, H. Cam, and A. Oltramari, “Effect of personality traits on trust and risk to phishing vulnerability: Modeling and analysis,” In Cognitive Methods in Situation Awareness and Decision Support (CogSIMA), *IEEE International Multidisciplinary Conference*, 2016, pp. 7-13.

[27] J.A. Johnson, M.L. Miller, D.R. Lynam, and S.C. South, “Five-Factor Model facets differentially predict in-the-moment affect and cognitions,” *Journal of Research in Personality*, vol. 46, Dec. 2012, pp. 752-759.

[28] G.M. Alarcon, J.B. Lyons, J.C. Christensen, S.L. Klosterman, M.A. Bowers, T.J. Ryan, S.A. Jessup, and K.T. Wynne, “The effect of propensity to trust and perceptions of trustworthiness on trust behaviors in dyads,” *Behavior Research Methods*, 2017, Advance online version.

[29] J.B. Rotter. “A new scale for the measurement of interpersonal trust.” *Journal of Personality*, vol. 35, Dec. 1967, pp. 651-665.

[30] S.M. Merritt, H. Heimbaugh, L. LaChapell, and D. Lee. “I trust it, but I don’t know why: Effects of implicit attitudes toward automation on trust in an automated system.” *Human Factors*, vol. 55, Jun. 2013, pp. 520-534.

[31] C. Calhoun, P. Bobko, M. Schuelke, S. Jessup, T. Ryan, C. Walter…C. Stokes. “Suspicion, trust, and automation.” SRA International Inc. Publication No. AFRL-RH-WP-TR-2017-0002, 2017.

[32] A.M. Evans and W. Revelle, “Survey and behavioral measurements of interpersonal trust,” *Journal of Research in Personality*, vol. 42, Dec. 2008, pp. 1585-1593.

[33] “Geotechnical Software Services, JAVA Programming Style Guidelines, http://geosoft.no/development/javastule.html,” 2015.

[34] “Google, JAVA Style Guidelines, http://google.github.io /styleguide.html,” 2014.

[35] “Sun Microsystems, JAVA Code Conventions, http://www.oracle.com/technetwork/java/codeconventions-150003.pdf,” 1997.

[36] T. Gaddis. Starting Out with JAVA: From Control Structures Through Objects, Boston, MA: Addison-Wesley, 2010.

[37] R.C. Mayer and J.H. Davis, “The effect of the performance appraisal system on trust for management: A field quasi-experiment,” *Journal of Applied Psychology*, vol. 84, Feb. 1999, pp. 123-136.

[38] J.P. Wanous, A.E. Reichers, and M.J. Hudy, “Overall job satisfaction: How good are single-item measures?” *Journal of Applied Psychology*, vol. 82, Apr. 1997, pp. 247-252.

[39] K.D. Carlson and A.O. Herdman, “Understanding the impact of convergent validity on research results,” *Organizational Research Methods*, vol. 15, Jan. 2012, pp. 17-32.

[40] J.T. Cacioppo and R.E. Petty, “The need for cognition,” *Journal of Personality and Social Psychology*, vol. 42, Jan. 1982, pp. 116-131.

[41] S. Epstein, R. Pacini, V. Denes-Raj, and H. Heier. “Individual differences in intuitive–experiential and analytical–rational thinking styles.” *Journal of Personality and Social Psychology*, vol. 71, Aug. 1996, pp. 390-405.

[42] R.R. McCrae and P.T. Costa, “Validation of the five-factor model of personality across instruments and observers,” *Journal of Personality and Social Psychology*, vol. 52, Jan. 1987, pp. 81-90.