

1-1-2020

Trust perceptions of metadata in open-source software: The role of performance and reputation

Gene M. Alarcon
Wright-Patterson AFB

Anthony M. Gibson
Consortium of Universities for Global Health

Charles Walter
University of Mississippi

Rose F. Gamble
University of Tulsa

Tyler J. Ryan
Wright State University

See next page for additional authors

Follow this and additional works at: https://egrove.olemiss.edu/engineering_facpubs

Recommended Citation

Alarcon, G. M., Gibson, A. M., Walter, C., Gamble, R. F., Ryan, T. J., Jessup, S. A., Boyd, B. E., & Capiola, A. (2020). Trust Perceptions of Metadata in Open-Source Software: The Role of Performance and Reputation. *Systems*, 8(3), 28. <https://doi.org/10.3390/systems8030028>

This Article is brought to you for free and open access by the Engineering, School of at eGrove. It has been accepted for inclusion in Faculty and Student Publications by an authorized administrator of eGrove. For more information, please contact egrove@olemiss.edu.

Authors

Gene M. Alarcon, Anthony M. Gibson, Charles Walter, Rose F. Gamble, Tyler J. Ryan, Sarah A. Jessup, Brian E. Boyd, and August Capiola

Article

Trust Perceptions of Metadata in Open-Source Software: The Role of Performance and Reputation

Gene M. Alarcon ^{1,*}, Anthony M. Gibson ², Charles Walter ³, Rose F. Gamble ⁴, Tyler J. Ryan ⁵, Sarah A. Jessup ¹, Brian E. Boyd ⁴ and August Capiola ¹

¹ Air Force Research Laboratory, Wright Patterson AFB, OH 45433, USA; sarah.jessup.ctr@us.af.mil (S.A.J.); august.capiola.1@us.af.mil (A.C.)

² Consortium of Universities, Washington, DC 20036, USA; anthony.gibson.9.ctr@us.af.mil

³ Computer & Information Science, University of Mississippi, Oxford, MS 38677, USA; cwwalter@olemiss.edu

⁴ College of Engineering & Natural Sciences, University of Tulsa, Tulsa, OK 74101, USA; gamble@utulsa.edu (R.F.G.); beb726@utulsa.edu (B.E.B.)

⁵ Department of Psychology, Wright State University, Dayton, OH 45433, USA; ryan.76@wright.edu

* Correspondence: gene.alarcon.1@us.af.mil

Received: 7 May 2020; Accepted: 10 August 2020; Published: 12 August 2020



Abstract: Open-source software (OSS) is a key aspect of software creation. However, little is known about programmers' decisions to trust software from OSS websites. The current study emulated OSS websites and manipulated reputation and performance factors in the stimuli according to the heuristic-systematic processing model. We sampled professional programmers—with a minimum experience of three years—from Amazon Mechanical Turk ($N = 38$). We used a 3×3 within-subjects design to investigate the relationship between OSS reputation and performance on users' time spent on code, the number of interface clicks, trustworthiness perceptions, and willingness to use OSS code. We found that participants spent more time on and clicked the interface more often for code that was high in reputation. Meta-information included with OSS tools was found to affect the degree to which computer programmers interact with and perceive online code repositories. Furthermore, participants reported higher levels of perceived trustworthiness in and trust toward highly reputable OSS code. Notably, we observed fewer significant main effects for the performance manipulation, which may correspond to participants considering performance attributes mainly within the context of reputation-relevant information. That is, the degree to which programmers investigate and then trust OSS code may depend on the initial reputation ratings.

Keywords: trust; heuristic-systematic model; code; open-source software

1. Introduction

Computer code has permeated almost every aspect of society, yet only recently have psychology researchers investigated how programmers perceive and reuse code [1]. The advent of open-source software (OSS) for use in larger architectures has shortened the required completion time of software products. With multiple options available for code that functions similarly, developers can choose which OSS to download, test, and implement. The decision to download and use the code is analogous to relying on the code, as the user makes themselves—and their system—vulnerable when downloading the code with the expectation the code will satisfy their requirements [2]. The *willingness* (or intention) to download and use the code is analogous to trust in the code [2]. Understanding the antecedents to trust in code would benefit developers who write code as well as those who seek out code to use in their own projects. Although prior research has examined the factors that influence the decision to use OSS in the computer science literature [3,4], none have approached this topic from a human factors

perspective or tested these factors using an experimental design. The current study sought to remedy this gap in the research.

1.1. Trust and OSS Interactions

Trust is the positive expectation of making oneself vulnerable to a referent [5]. Although trust is typically referenced within human–human interactions, trust also plays an important role in how people perceive non-human referents. For example, Lee and See [6] expanded the trust literature to automation contexts—that is, human trust toward automation. They mapped Mayer and colleagues' [5] ability, benevolence, and integrity perceptions of human trustworthiness to performance, purpose, and process perceptions of automation trustworthiness, respectively (see [6], p. 59). Similarly, Oleson and colleagues [7] have explicated the factors that influence when a person trusts a robot. Lastly, Alarcon and colleagues [1,2] have investigated how programmers trust code [1,2], linking the factors of performance, transparency, and reputation to Mayer and colleagues' [5] ability, benevolence and integrity factors in human–human trust, respectively [2]. These factors are antecedents to a willingness to be vulnerable to the consequences of utilizing computer code. Across the aforementioned literature, trust is related to reliance behaviors, both for human–human [2] and human–automation interactions [8,9].

The widespread use of code and the need for code in a safe and timely manner has led to an increase in code reuse. Code reuse is defined as “the use of existing software or software knowledge to construct new software” [10] (p. 529). The decision to reuse code presents some advantages but also potential risks [2]. The reused code may contain errors or malicious code, which can hinder the new architecture [1,2]. Prior research has examined the relationship between OSS properties and reuse behaviors. Li and colleagues [3] identified a lack of guaranteed, long-term technical support as a major concern associated with OSS software. Most OSS technical support is run on a volunteer basis; without a formal contract (i.e., with a vendor), it may be hard to guarantee support. Despite these difficulties, many programmers reuse OSS in their architectures, which indicates reliance on OSS [2,6]. Furthermore, Weber and colleagues [4] used data mining and machine-learning techniques to ascertain and predict factors associated with popular or unpopular projects. Three key features were identified. First, popular projects were found to have larger README files. Second, popular projects used the Python WITH statement more frequently, which was designed specifically to be read easily. Finally, projects with high popularity were found to have Travis CI, a service used to test software projects, configured much more frequently than projects with low popularity.

Computer science and psychological researchers have recently examined how developers perceive and trust code. For example, in the field of computer science, Hasselbring and Reussner [11] examined the main features associated with software trustworthiness. In this context, the researchers operationalized trust as the risk of making software available for use. The results showed that the risk of software deployment decreased through the improvement of the certification of trustworthiness. In psychology, a cognitive task analysis (CTA) found that three key factors—reputation, transparency, and performance—affect perceptions of code trustworthiness and developers' decisions to reuse code written by another programmer [2]. Reputation represents information cues about the source of the code such as the number of reviews, the origin (e.g., website, colleagues), or the number of users. Transparency represents the understanding of the code upon examination. Lastly, performance represents the ability of the code to meet context-specific needs. Additionally, researchers have adapted a model from the persuasion literature to explain how trustworthiness perceptions influence programmers' interactions with computer code [12].

1.2. Heuristic-Systematic Processing Model of Trust in Code

The heuristic-systematic processing model (HSM) of persuasion is a dual-process model that posits people are efficiency-driven and use two methods for analyzing information: heuristic and systematic processing [13]. Heuristic processing involves the use of mental shortcuts (e.g., relying

on norms and biases) to reach a decision [13], whereas systematic processing is an effortful, deep analysis of stimuli [14]. Heuristic processing is often faster than systematic processing but may be less accurate [13]. In reality, cognitive processing is almost certainly not discretized into two independent systems, but nomenclature facilitates ease of communication as to how people may process information with different amounts of effort [15]. Based on the HSM, people have a threshold that indicates the extent to which systematic processing is necessary (i.e., the sufficiency principle). If that particular threshold is unmet, people will default to the less-effortful heuristic approach. Prior research has supported the use of the HSM when investigating the effects of reputation and transparency on code trustworthiness perceptions and willingness to reuse (or trust) code [12,16]. For the current paper, however, we consider only reputation and performance characteristics.

Reputation is portrayed as meta-information about the source of computer code, as described in the section above. Reputation characteristics have influenced code perceptions in previous research [17–19]. Sim and colleagues [19], for example, found that social cues (e.g., reputation characteristics) had a greater influence on internet code reuse than the technical properties of the code did. Alarcon and colleagues [16,20,21] have found that programmers dedicate more time to examining code developed from a reputable source. In accordance with the HSM, participants appeared to allot systematic processing to the code once it was established that the code was from a reputable source and also assessed the code more accurately (e.g., ensured all code in the study compiled and was functional). In the present study, it was hypothesized that reputation characteristics displayed in an online repository would be related to trustworthiness perceptions in a similar manner.

Hypothesis 1a. *Code reputation is positively related to time spent on code.*

Hypothesis 1b. *Code reputation is positively related to code interactions.*

Hypothesis 1c. *Code reputation is positively related to trustworthiness perceptions.*

Hypothesis 1d. *Code reputation is positively related to participant's willingness to reuse code (i.e., trust).*

Performance is described as the overall ability of code to complete a context-dependent task. Performance has been linked to key aspects of trust in other scenarios, such as robotics [7] and automation [6]. Related to computer science, Lingzi and Zhi [22] found that performing audits on code led to increased user confidence and code usage, both of which are associated with trust. Stated simply, auditing code provides important performance-relevant information about the code [10], while also revealing security vulnerabilities and inefficiencies in code execution. The increased transparency of the underlying processes in the code from the audit led to an increase in trustworthiness perceptions. Similarly, the most important qualities found when selecting OSS to use were compliance with user requirements, extensibility, and ease of updates [17], all of which are possible metrics of performance. Code that meets the appropriate standards for performance will engage the use of heuristics, leading to less mental processing of the code. In contrast, code that is lower in performance will necessitate a deeper dive, ascertaining more information about the code before making a judgement (in other words, users become engaged in more systematic processing). In the present study, it was hypothesized that performance characteristics displayed in an online repository would be related to trustworthiness perceptions.

Hypothesis 2a. *Code performance is negatively related to time spent on code.*

Hypothesis 2b. *Code performance is positively related to code interactions.*

Hypothesis 2c. *Code performance is positively related to trustworthiness perceptions.*

Hypothesis 2d. *Code performance is positively related to participants' willingness to reuse code (i.e., trust).*

Prior research, however, has shown that trust perceptions of code sometimes interact unpredictably. Interestingly, Alarcon and colleagues [1] found that when code was organized poorly but was highly readable and reputable, programmers spent more time on and were more trusting of the code. According to the HSM, the high reputation and readability of the code may have prompted the user to perform systematic processing over code snippets that are functional and compile [2,12], even though the code was poorly organized. Similarly, Alarcon and colleagues [16] investigated the effects of comments—defined as documentation that has no effect on code functionality—on trust perceptions of code. Trust assessments were found to not be solely based on the code itself but can be influenced by informational cues ascertained from outside sources, such as commenting or perhaps the repository website from which the code was obtained.

These studies indicated that the relationship between the factors found in Alarcon and colleagues' [2] CTA and trustworthiness perceptions is not straightforward. Thus, we used exploratory techniques when interpreting the extent to which the reputation of the code source and perceived code performance interact to influence trustworthiness perceptions and willingness to reuse (i.e., trust) the OSS code. Specifically, in cases where we found a significant two-way interaction, we examined the bar charts and highlighted the general trends without reporting inferential statistics.

2. Method

2.1. Participants

We recruited participants from Amazon Mechanical Turk (MTurk) to take part in the one-hour study in exchange for 10.00 USD ($N = 42$). A minimum of three years of programming experience was necessary for participation eligibility in the study. A total of four cases were excluded from the analyses due to careless data (i.e., never opening the README file, taking an average of less than 20 s per stimulus, etc.), leaving a final sample of 38 participants. The sample was primarily male (82%) with a mean age of 31.63 years (range: 21–50 years), and the mean years of programming experience was 6.55 (range: 3–20 years). This research complied with the American Psychological Association Code of Ethics and was approved by the Institutional Review Board at the Air Force Research Laboratory. Informed consent was obtained from each participant.

2.2. Measures

2.2.1. Trustworthiness

We used a single-item measure of overall trustworthiness for each stimulus. Participants indicated their trustworthiness perception of the code with the item “How trustworthy do you find this code?” on a graphic rating scale ranging from 1 (Untrustworthy) to 7 (Trustworthy). We found that this single-item measure is appropriate in this case, as the item is unambiguous and multiple-item measures are likely to cause response fatigue [23].

2.2.2. Remarks

Participants were provided with a section to input text regarding the stimuli and their decision after they made the decision that they would be willing to either use or not use it. Text was required for the participant to proceed to the next stimulus.

2.2.3. Timing

As participants continued through the study, we measured the amount of time they took (in seconds) on each stimulus. The time spent evaluating the code was calculated by recording timestamps as the participant moved through the stimuli, providing the total time spent on each stimulus.

2.2.4. Click amount

Each stimulus presentation started on the Main Page (“code” tab), which had two additional tabs for “Open Issues” and “Closed Issues.” The README file on the main code page could also be clicked on to be opened or closed. The number of clicks per Main Page, Open Issues, Closed Issues, and README file were totaled across participants for each stimulus.

2.2.5. Willingness to Reuse Code

Participants were asked to decide whether they would use the code by selecting from the response option “Use” or “Don’t Use.” As OSS originates from a developer that is not the participant, the willingness to use or not to use is synonymous with the willingness to reuse or not to reuse.

2.3. Stimuli

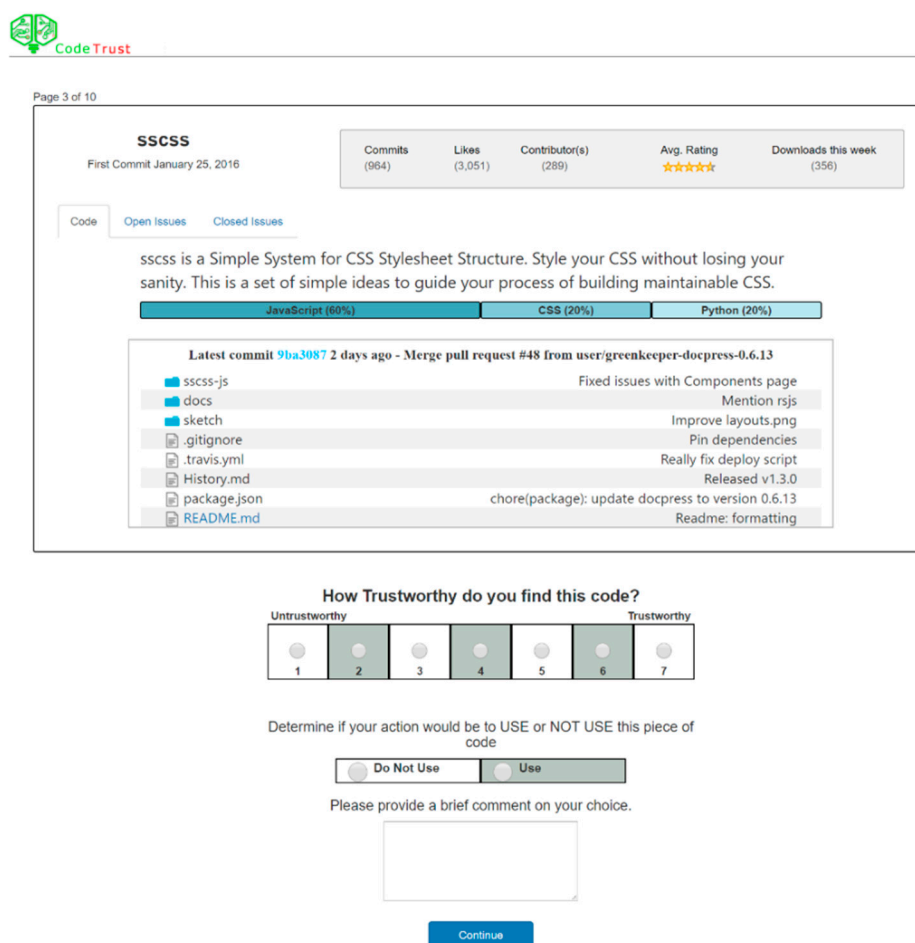
We separated the relevant metadata factors into two factors (i.e., reputation and performance). Reputation was represented by the following attributes: number of likes, average rating (i.e., number of starts out of five), number of downloads in the last week, the last commit date, the general product description, the total commits (changes to the original code file), the dates of the open/closed issues, and the file structure (i.e., file naming conventions). This metadata directly expresses how well liked the code is by others and how professional the programmers are when creating the code, both of which are qualities that should lead to increased trust in the resulting code.

For performance metadata, we targeted the number of contributors, the ratio of open to closed issues, the description of the open/closed issues, commit messages, the technical product description (essentially the README file), the number of programming languages used, and the date of the first commit. These factors all relate to how the code performs when used and how the developers work to improve the performance of the code over time.

We examined popular OSS projects hosted on GitHub, SourceForge, and Bitbucket for baseline ranges of the reputation and performance factors. The values we established are shown in Table 1, separated by reputation metadata (top half) and performance metadata (bottom half). Once a baseline of “good” values was established (High column), we determined what values would be considered “bad” by surveying opinions from professional programmers, current CS graduate students, and faculty, which is represented in the Low column. This process resulted in two levels of reputation and performance metadata, high and low. Medium values were set in between with additional consideration to ensure there was enough degradation without causing alarm. Medium values may be ignored by coders based on other factors, such as a project not having a large number of stars but also being created recently. Low metadata values were set to be significant enough to cause alarm from only one form of metadata. Although this alarm may not be enough to cause a developer to distrust the code, it should warrant a closer examination of other factors related to the OSS project before making a final decision.

2.4. Procedure

We created a website that allowed study participants to navigate between the main project page, the open issues, the closed issues, and view the README file. The stimuli mimicked typical navigation available on most hosting sites. A brief tutorial was created to inform the participants of what they were seeing and how to navigate the stimuli and make decisions regarding the code. We continuously displayed total commits, likes, contributors, average rating, and weekly downloads in a bar visible at the top of each page, allowing users to view the information regardless of the active page. An example of a stimulus can be seen in Figure 1.



Page 3 of 10

SSCSS
First Commit January 25, 2016

Commits	Likes	Contributor(s)	Avg. Rating	Downloads this week
(964)	(3,051)	(289)	★★★★★	(356)

Code Open Issues Closed Issues

sscss is a Simple System for CSS Stylesheet Structure. Style your CSS without losing your sanity. This is a set of simple ideas to guide your process of building maintainable CSS.

JavaScript (60%) CSS (20%) Python (20%)

Latest commit 9ba3087 2 days ago - Merge pull request #48 from user/greenkeeper-dockpress-0.6.13

File	Commit	Message
sscss.js	9ba3087	Fixed issues with Components page
docs	9ba3087	Mention rsjs
sketch	9ba3087	Improve layouts.png
.gitignore	9ba3087	Pin dependencies
.travis.yml	9ba3087	Really fix deploy script
History.md	9ba3087	Released v1.3.0
package.json	9ba3087	chore(package): update dockpress to version 0.6.13
README.md	9ba3087	Readme: formatting

How Trustworthy do you find this code?

Untrustworthy 1 2 3 4 5 6 7 Trustworthy

Determine if your action would be to USE or NOT USE this piece of code

☐ Do Not Use ☒ Use

Please provide a brief comment on your choice.

Continue

Figure 1. A sample of open-source software (OSS) stimuli shown to participants.

Navigation was achieved through tabs that took the user between the code structure, open issues, and closed issues. These tabs are similar to the tabs used by GitHub to navigate between the code, pull requests, insights, and other projects. We captured the number and order of the user clicks as participants moved through the tabs and opened/closed the README file, in order to obtain a complete picture of the nature in which participants navigated the code sample.

2.5. Analysis

A power analysis for a repeated measures Analysis of Variance (RM ANOVA) in G*Power was used to determine the sample size necessary to detect our hypothesized effects [24]. We set the statistical power estimate to 0.80 and the significance level to $\alpha = 0.05$ for a two-tailed test. The effect size was estimated to be moderate, Cohen's $f = 0.25$, as there is no previous literature on the topic to suggest otherwise. Results from the power analysis indicated we needed a sample size of 38 to find our effects. We used RM ANOVAs for the analyses with perceived trustworthiness and time-dependent variables. All ANOVAs were tested for sphericity with Mauchly's test of sphericity, and any violations resulted in corrected F-statistics using the Greenhouse–Geisser correction (note the decimals shown in the degrees of freedom listed below). The analyses with click count and code usage dependent variables were analyzed with generalized linear mixed effects models (GLMMs), given the count and dichotomous nature of the variables, respectively.

Table 1. Study details across the reputation and performance manipulations.

Manipulation	Features	Levels		
		Low	Medium	High
Reputation	Likes	≤499	500–1999	≥2000
	Stars	0–2.5	2.5–3.5	3.5–5
	Downloads last week	≤50	51–199	≥200
	Last Commit Date	≥16 days	4–15 days	≤3 days
	General Product Description	>2 spelling or grammar errors	1–2 spelling or grammar errors	No spelling or grammar errors
	Total Commits	≤200	201–499	≥500
	Dates of Open Issues	≤7 days ago	7–30 days ago	31–365 days ago
	Dates of Closed Issues	31–365 days ago	7–30 days ago	≤7 days ago
	File Structure	Few well named files	Some well named files	Well named files
Performance	# of Contributors	0.5 < # contributors/commits	0.25 < # contributors/commits < 0.30	0.05 < # contributors/commits < 0.10
	Ratio of Open to Closed Issues	≥0.5	≤0.49	≤0.15
	Description of Open Issues	Mainly functionally related	Some functionally related	Not functionally related
	Description of Closed Issues	Not functionally related	Some functionally related	Mainly functionally related
	Commit Messages	Very vague	Partially vague	Descriptive
	Technical Product Description	Vague or absent ReadMe	Somewhat vague ReadMe	Detailed ReadMe
	# of Unique Languages	>5	2–3	1–2
	Date of First Commit	<3 months ago	3–12 months ago	>12 months ago

3. Results

To determine if our manipulations were perceived by the participants, we qualitatively coded the remarks made about each code stimulus for reputation and performance. We separated the remarks into positive and negative remarks made about each construct. Table 2 illustrates the results of the qualitative coding. As illustrated in the table, when the constructs were degraded, participants made more negative comments about the respective construct. In contrast, when the constructs were not degraded, participants made more positive remarks about the constructs.

Table 2. Counts of qualitative remarks made about the reputation and performance of the code.

Code #	Reputation	Performance	Reputation		Performance	
			Positive	Negative	Positive	Negative
7	High	High	22	1	32	4
8	High	Medium	0	27	2	29
5	High	Low	0	24	0	33
1	Medium	High	20	0	28	8
3	Medium	Medium	27	0	32	1
2	Medium	Low	1	21	1	33
4	Low	High	3	18	12	27
6	Low	Medium	14	8	19	20
9	Low	Low	0	25	1	33
10	Distractor	Distractor	0	25	0	33

Note. Code # = order of code presentation.

3.1. Time Spent on Code

The results from the RM ANOVA indicate a statistically significant main effect of reputation on total time spent analyzing the code, $F(2, 74) = 13.71$, $p < 0.01$, $\eta_p^2 = 0.27$ (see Table 3). The main effect of performance on time spent on code was not significant. The main effects were qualified by a significant two-way interaction $F(2.69, 99.59) = 3.35$, $p < 0.05$, $\eta_p^2 = 0.08$. The results indicate reputation characteristics displayed in the repository simulation affected time spent on code.

We conducted follow-up pairwise contrasts based on the observed estimated marginal means of seconds spent on code across the performance and reputation conditions (see Figure 2). We tested the effects of performance across the three levels of reputation, as code reputation was assumed to be considered the first set of characteristics examined when programmers scan unfamiliar code. We found when observing code with a high reputation, participants spent more time in seconds on low-performance code ($M = 246.60$) compared to both the medium-performance code ($M = 156.6$) and the high-performance code ($M = 143.7$). Participants appeared to scan the low-performance code for longer when it originated from a highly reputable source, but this should be replicated with future samples.

Table 3. Estimated marginal means for the time spent on the code.

Mean Estimates				IV	df	F	η_p^2
Performance							
Reputation	Low	Medium	High				
Low	78.60 (24.80)	107.60 (24.80)	105.90 (24.80)	Reputation (A)	1.75, 64.75	13.71 **	0.27
Medium	137.50 (24.80)	189.20 (24.80)	162.00 (24.80)	Performance (B)	1.60, 59.07	0.40	0.01
High	246.60 (24.80)	156.60 (24.80)	143.70 (24.80)	A \times B	2.69, 99.59	3.35 *	0.08

Note. $N = 38$. Time spent was measured in seconds. Standard error estimates shown in parentheses. * $p < 0.05$,

** $p < 0.01$.

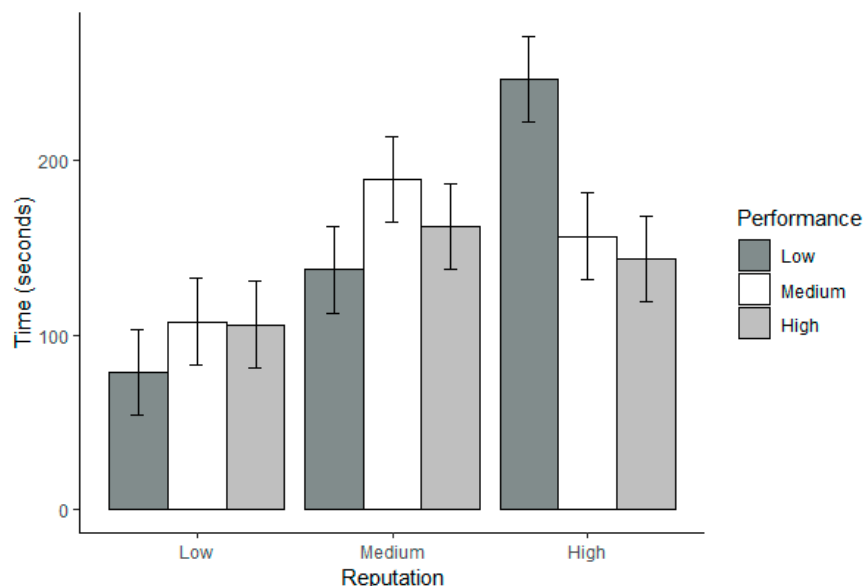


Figure 2. Average time spent on code across the levels of reputation and performance.

3.2. Number of Clicks

Because the number of clicks was a count variable, we analyzed the number of clicks across the performance and reputation conditions using a GLMM with a Poisson distribution. We found a significant effect of reputation on the number of clicks, Wald $\chi^2(2) = 7.04$, $p < 0.05$ (see Table 4), but found no significant effect for performance. There was also a significant Performance \times Reputation interaction, Wald $\chi^2(4) = 19.55$, $p < 0.01$. Similar to the findings with total time spent on code, we found

when analyzing highly reputable code, participants had more mouse clicks within the low performance code ($M = 5.25$) compared to both the medium performance code ($M = 3.45$) and the high performance code ($M = 3.73$). The pattern of means appears to show no other noteworthy differences.

Table 4. Estimated marginal means for the number of clicks across levels of reputation and performance.

Mean Estimates				IV	df	Wald χ^2
Performance						
Reputation	Low	Medium	High			
Low	3.56 (0.56)	3.69 (0.58)	3.30 (0.52)	Reputation (A)	2	7.04 *
Medium	3.75 (0.58)	4.50 (0.69)	3.88 (0.60)	Performance (B)	2	4.40
High	5.25 (0.79)	3.45 (0.54)	3.73 (0.58)	A \times B	4	19.55 **

Note. $N = 38$. Number of clicks was modeled with a general linear mixed model with Poisson distribution. Standard error estimates shown are in parentheses. * $p < 0.05$, ** $p < 0.01$.

3.3. Trustworthiness

The results from the RM ANOVA indicate a significant main effect of reputation on trustworthiness scores, $F(1.92, 71.17) = 331.68$, $p < 0.01$, $\eta_p^2 = 0.90$ (see Table 5). Similarly, a significant main effect of performance was observed $F(1.69, 62.64) = 20.16$, $p < 0.01$, $\eta_p^2 = 0.35$. Importantly, we observed a significant two-way interaction, $F(3.52, 130.36) = 3.86$, $p < 0.01$, $\eta_p^2 = 0.09$. The results indicate that the reputation and performance characteristics displayed in the repository simulation affected perceptions of code trustworthiness (see Figure 3).

Table 5. Estimated marginal means for trustworthiness.

Mean Estimates				IV	df	F	η_p^2
Performance							
Reputation	Low	Medium	High				
Low	1.55 (0.17)	1.87 (0.17)	2.11 (0.17)	Reputation (A)	1.92, 71.17	331.68 **	90
Medium	2.89 (0.17)	3.92 (0.17)	4.21 (0.17)	Performance (B)	1.69, 62.64	20.16 **	35
High	5.74 (0.17)	6.11 (0.17)	6.16 (0.17)	A \times B	3.52, 130.36	3.86 **	09

Note. $N = 38$. Standard error estimates shown in parentheses. ** $p < 0.01$.

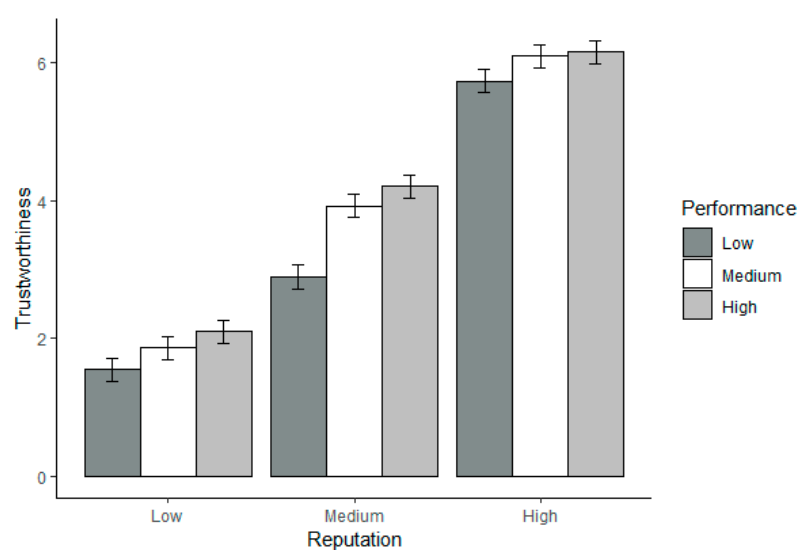


Figure 3. Average ratings of code trustworthiness by reputation and performance.

The pattern of estimated means shows several interesting patterns for trustworthiness perceptions. First, when examining code with a low reputation, participants rated low-performance code

($M = 1.55$) less trustworthy than high-performance code ($M = 2.11$). It also appears that under the medium-reputation condition, low-performance code ($M = 2.89$) was rated lower in perceived trustworthiness compared to both medium-performance ($M = 3.92$) and high-performance code ($M = 4.21$). Finally, perceived trustworthiness appeared to decrease slightly for the high-reputation condition with low performance ($M = 5.74$) compared to both medium performance ($M = 6.11$) and high performance ($M = 6.16$).

3.4. Willingness to Reuse the Code

We operationalized the willingness to (re)use the code (i.e., trust) as a dichotomous variable indicating whether or not programmers would use the code. We used a GLMM with a binomial distribution to analyze the binary data (1 = Use, 0 = Don't Use). We found a significant main effect of reputation, Wald $\chi^2(2) = 6.99$, $p < 0.05$, as well as a significant main effect for performance, Wald $\chi^2(2) = 84.78$, $p < 0.01$ (see Table 6). We failed to find a significant Performance \times Reputation interaction.

Table 6. Estimated marginal means for usage (%).

Mean Estimates				IV	df	Wald χ^2
Performance						
Reputation	Low	Medium	High			
Low	2.62 (2.59)	10.49 (4.99)	7.86 (4.38)	Reputation (A)	2	6.99 *
Medium	10.49 (4.99)	44.72 (8.18)	52.64 (8.22)	Performance (B)	2	84.78 **
High	92.14 (4.38)	97.38 (2.59)	89.51 (4.99)	A \times B	4	6.99

Note. $N = 38$. Usage was modeled with a general linear mixed model with a binomial distribution to analyze the binary data. We coded use as one and non-use as zero. Standard error estimates are shown in parentheses. * $p < 0.05$, ** $p < 0.01$.

In order to examine the differences across the levels of performance and reputation conditions separately, we compared the estimated means. For the reputation condition, we found that a significantly lower percentage of participants would be willing to use a low-reputation code (6.06%) compared to both a medium-reputation code (32.08%), $z = -4.06$, $p < 0.01$, and a high-reputation code (93.9%), $z = -9.00$, $p < 0.01$ (see Figure 4). The difference in the percentage of participants who would use a medium-reputable code versus a high-reputable code was also significant, $z = -7.12$, $p < 0.01$. For the performance condition, we found that a lower percentage of participants would be willing to use a low performance code (25.0%) compared to a medium performance code (60.3%), $z = -2.60$, $p < 0.05$ (see Figure 5). We observed no other significant post-hoc comparisons for the performance manipulations.

4. Discussion

The current study examined the influence of reputation and performance metadata on trustworthiness perceptions and willingness to use code within OSS websites. Overall, we found that more reputable metadata led to more time spent on the OSS, more involvement with the code, higher levels of trustworthiness, and more willingness to use the software, which provides support for Hypothesis 1. Stated simply, programmers are more likely to interact with and trust the OSS when the metadata suggests it comes from a reputable source. Alternatively, when considered independently of the level of reputation, our performance manipulation appeared to only influence trustworthiness perceptions and willingness to reuse the code. Specifically, programmers were more likely to rate the OSS as more trustworthy and be willing to reuse the OSS when the metadata indicated that the code performed the necessary tasks well, providing some support for Hypothesis 2. Below, we discuss further the implications of our findings.

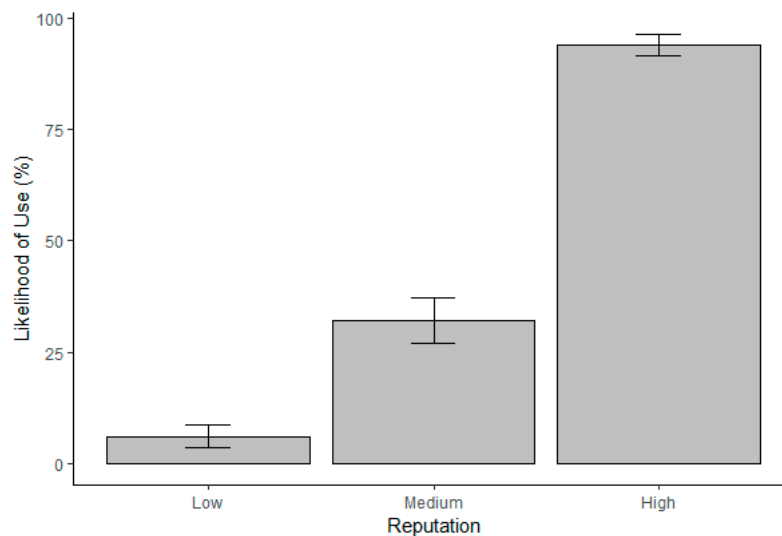


Figure 4. Average percentage of willingness to use the code across reputation levels.

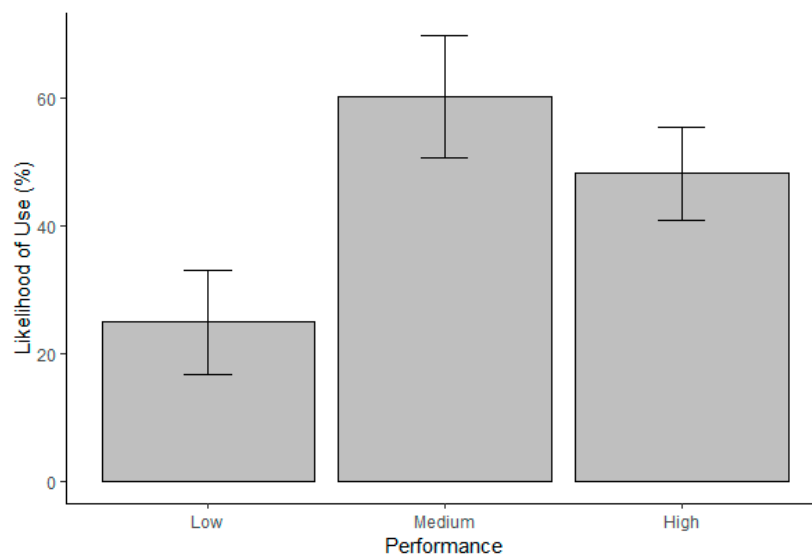


Figure 5. Average percentage of willingness to use the code across performance levels.

4.1. Reputation

Reputation perceptions, derived from the manipulated metadata information, appeared to affect the way in which programmers interact with the OSS throughout the entire process (i.e., initial investigation duration, interaction with code, trustworthiness, and willingness to use). When assessing OSS on a web platform, programmers assess the reputation of the code through various informational cues on the website, as information about the original author or authors of subsequent changes may be unknown. As such, programmers rely on cues (e.g., ratings, likes, recent downloads) to determine the reputation of the code. Reputation had a main effect on both the perceptions of the code through self-reported trustworthiness assessments, willingness to use the code, and behaviors such as time spent on the code. These results are similar to previous research on trust in code where reputation was simply coded as “reputable” or “unknown” [1]. Through the lens of the HSM, participants in the current study also appeared to use heuristics to select out of code that failed to meet norms of adequacy, supporting Alarcon and Ryan’s [12] select out process. If the code met the heuristic of being adequate in reputation, further systematic processing appeared to be conducted. We discuss this further when we describe the observed interactions below.

4.2. Performance

Interestingly, the performance manipulations only had a significant main effect on trustworthiness perceptions and willingness to use the code, and the significant effects of performance on trustworthiness appeared to be most impactful when reputation was ambiguous (i.e., the medium-reputation condition). However, we found participants were more likely to report being willing to use the code with medium performance versus low performance ratings. As such, the results support the findings of the CTA on trust in computer code that noted performance as an important factor of trust [2].

However, performance manipulations did not have a main effect on time spent on the code or the number of clicks on the webpage. This may be for several reasons. First, the participants are assessing the code performance through ancillary information (i.e., the number of contributors, the ratio of open to closed issues, etc.) rather than direct testing. Programmers may be able to do a cursory assessment of performance from the webpage, but direct testing of the code when downloaded may provide more information and lead to more accurate trust calibration. Indeed, there was no statistical difference between the high- and medium-performance conditions for willingness to use, contrary to our hypotheses. However, participants reported less willingness to use code low in performance compared to medium performance. Finally, according to the HSM, people rely on heuristics until the situation indicates more systematic processing is necessary. In the current context, participants may have considered performance only when a cursory glance of reputation indicated additional consideration (i.e., an interaction), which we discuss further in the section below.

4.3. Interactions

The results show significant Performance \times Reputation interactions for time spent, the number of clicks, and trustworthiness. In general, participants appeared to spend more time investigating the OSS when expectations were violated (e.g., high reputation but low performance) or when the cursory glance revealed insufficient information (e.g., medium-reputation levels with medium-performance indications). Although these findings should be replicated in future research, the results align with predictions derived from the HSM theory of code [12]. When participants found the reputation heuristic to be completely satisfied (or dissatisfied), they appeared to quickly scan the OSS. Otherwise, participants seemed to engage in more processing when they experienced contradictory or insufficient information. A similar pattern appeared for trustworthiness perceptions and willingness to use the code. Generally, participants used a select-in or select-out approach when given sufficient, complementary information. For example, participants appeared to report high levels of trustworthiness and higher use intentions for highly reputable code, regardless of the performance indicators, and the largest observed differences in performance and trustworthiness were observed for a medium level of reputation. Thus, researchers, practitioners, and users of OSS should ensure that the reputation levels match the performance levels to reduce the processing time needed to scan the information and ensure appropriate reliance behaviors.

4.4. Limitations and Future Work

The current study is not without limitations. First, participants were unable to download and test the code before making a decision as to whether they would use the code. This was a necessary constraint of the current study; our focus was on the effects of displayed metadata on trust in code. However, future work would benefit from allowing programmers to not only assess metadata, but also allow them to download and test the code. This would allow researchers to determine how programmers' trust in code changes once they are allowed to test the code after making initial assessments based solely on metadata.

Second, we note there are a number of factors other than trust that influence one's actual behavioral reliance on something [6], such as subjective workload and self-confidence and other external variability factors outlined by Hoff and Bashir's [9] systematic review. Other environmental factors (e.g., context

novelty, user's decision freedom) can also have an effect on the trust–reliance relationship [9] (p. 418). Simply put, we agree that trust toward automation does not completely determine reliance [6]. So, too, could there be other factors (e.g., personality; see [20]) that influence one's reliance on code. Indeed, in the interpersonal trust literature, the antecedents to trust a referent vary depending on the amount of time and experience a trustor has with a referent [5,25]. In addition, research on trust in code has found that user personality has an impact on perceptions of and trust in computer code [20,26]. Future research would benefit from investigating the interplay of user characteristics and code manipulations on trust in code before and after testing the claims of metadata. That is, researchers should compare what individual difference variables influence trust in code in addition to code manipulations when a programmer initially views metadata compared to after a programmer has time to test the code and make a final decision to use the code. However, the focus of the current paper was the role of OSS reputation and performance attributes on users' time spent on OSS code, the number of interface clicks, trustworthiness perceptions of, and willingness to use (i.e., trust), OSS code.

Third, it should be noted that the code from the OSS may not be the referent in the current task. In previous research, participants reviewed code snippets without any information about the authors (see [1]). In contrast, in the current study, participants did not actually view the code itself. As such, some participants may have made inferences about the trustworthiness of the author(s) of the code rather than the code itself. It should be noted the authors of the code in the current study were all the same, as the manipulations were created internally in order to preserve experimental control. Regardless, some perceptions of non-human referents are difficult to separate from perceptions toward their designers. For example, Lee and See [6] explain that it is inherently difficult to extrapolate purpose attributes (i.e., why was the automation developed and whether its use aligns with the designer's intent) into automation design perspectives on automation. Indeed, purpose perceptions of automation may be inseparable from ascriptions of intent from the designer. Similarly, we are unable to verify that participants in our study were not considering the author of the code when evaluating the code. That is, participants may have inferred the intentions of the author of the code rather than the code itself from the OSS repository.

Author Contributions: Conceptualization, G.M.A., C.W., R.F.G. and S.A.J.; Methodology, R.F.G., C.W., and B.E.B.; Software, R.F.G., C.W., and B.E.B.; Investigation, G.M.A., T.J.R., and S.A.J.; Data Curation, G.M.A., A.M.G., T.J.R., and A.C.; Writing-Original Draft Preparation, G.M.A., A.M.G., C.W., R.F.G., T.J.R., S.A.J., and A.C.; Writing-Review & Editing, G.M.A., A.M.G., C.W., R.F.G., T.J.R., S.A.J., and A.C.; Supervision, G.M.A.; Funding Acquisition, G.M.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Alarcon, G.M.; Gamble, R.; Jessup, S.A.; Walter, C.; Ryan, T.J.; Wood, D.W.; Calhoun, C.S. Application of the heuristic-systematic model to computer code trustworthiness: The influence of reputation and transparency. *Cogent Psychol.* **2017**, *4*, 1389640. [[CrossRef](#)]
2. Alarcon, G.M.; Militello, L.G.; Ryan, P.; Jessup, S.A.; Calhoun, C.S.; Lyons, J.B. A descriptive model of computer code trustworthiness. *J. Cogn. Eng. Decis. Mak.* **2017**, *11*, 107–121. [[CrossRef](#)]
3. Li, J.; Conradi, R.; Slyngstad, O.P.N.; Bunse, C.; Torchiano, M.; Morisio, M. An empirical study on decision making in off-the-shelf component-based development. In Proceedings of the 28th International Conference on Software Engineering, Shanghai, China, 20–28 May 2006; pp. 897–900.
4. Weber, S.; Luo, J. What makes an open source code popular on github? In Proceedings of the 2014 IEEE International Conference on Data Mining Workshop, Shenzhen, China, 14 December 2014; pp. 851–855.
5. Mayer, R.C.; Davis, J.H.; Schoorman, F.D. An integrative model of organizational trust. *Acad. Manag. Rev.* **1995**, *20*, 709–734. [[CrossRef](#)]
6. Lee, J.D.; See, K.A. Trust in automation: Designing for appropriate reliance. *Hum. Factors* **2004**, *46*, 50–80. [[CrossRef](#)] [[PubMed](#)]

7. Oleson, K.E.; Billings, D.R.; Kocsis, V.; Chen, J.Y.C.; Hancock, P.A. Antecedents of trust in human-robot collaborations. In Proceedings of the 2011 IEEE International Multi-Disciplinary Conference on Cognitive Methods in Situation Awareness and Decision Support, Miami, FL, USA, 22–24 February 2011; pp. 175–178.
8. Lyons, J.B.; Ho, N.T.; Koltai, K.S.; Masequesmay, G.; Skoog, M.; Cacanindin, A.; Johnson, W.W. Trust-based analysis of an Air Force collision avoidance system. *Ergon. Des.* **2016**, *24*, 9–12. [\[CrossRef\]](#)
9. Hoff, K.A.; Bashir, M. Trust in automation: Integrating empirical evidence on factors that influence trust. *Hum. Factors* **2015**, *57*, 407–434. [\[CrossRef\]](#) [\[PubMed\]](#)
10. Frakes, W.B.; Kang, K. Software reuse research: Status and future. *IEEE Trans. Softw. Eng.* **2005**, *31*, 529–536. [\[CrossRef\]](#)
11. Hasselbring, W.; Reussner, R. Toward trustworthy software systems. *Computer* **2006**, *39*, 91–92. [\[CrossRef\]](#)
12. Alarcon, G.; Ryan, T. Trustworthiness perceptions of computer code: A heuristic-systematic processing model. In Proceedings of the 51st Hawaii International Conference on System Sciences, Hilton Waikoloa Village, HI, USA, 3–6 January 2018; pp. 5384–5393.
13. Chaiken, S. Heuristic versus systematic information processing and the use of source versus message cues in persuasion. *J. Personal. Soc. Psychol.* **1980**, *39*, 752–766. [\[CrossRef\]](#)
14. Chen, S.; Duckworth, K.; Chaiken, S. Motivated heuristic and systematic processing. *Psychol. Inq.* **1999**, *10*, 44–49. [\[CrossRef\]](#)
15. Kahneman, D. *Thinking, Fast and Slow*; Macmillan: New York City, NY, USA, 2011; ISBN 978-0374275631.
16. Alarcon, G.M.; Gamble, R.F.; Ryan, T.J.; Walter, C.; Jessup, S.A.; Wood, D.W.; Capiola, A. The influence of commenting validity, placement, and style on perceptions of computer code trustworthiness: A heuristic-systematic processing approach. *Appl. Ergon.* **2018**, *70*, 182–193. [\[CrossRef\]](#) [\[PubMed\]](#)
17. Chen, W.; Li, J.; Ma, J.; Conradi, R.; Ji, J.; Liu, C. An empirical study on software development with open source components in the Chinese software industry. *Softw. Process Improv. Pract.* **2008**, *13*, 89–100. [\[CrossRef\]](#)
18. Gallardo-Valencia, R.E.; Tantikul, P.; Sim, S.E. Searching for reputable source code on the web. In Proceedings of the 16th ACM International Conference on Supporting Group Work, Sanibel Island, FL, USA, 13–16 November 2010; pp. 183–186.
19. Sim, S.E.; Umarji, M.; Ratanotayanon, S.; Lopes, C.V. How well do internet code search engines support open source reuse strategies. *ACM Trans. Softw. Eng. Methodol.* **2009**, *21*, 1–22. [\[CrossRef\]](#)
20. Ryan, T.J.; Walter, C.; Alarcon, G.M.; Gamble, R.F.; Jessup, S.A.; Capiola, A.A. Individual differences in trust in code: The moderating effects of personality on the trustworthiness-trust relationship. In Proceedings of the International Conference on Human-Computer Interaction, Las Vegas, NV, USA, 15–20 July 2018; pp. 370–376.
21. Walter, C.; Gamble, R.; Alarcon, G.; Jessup, S.; Calhoun, C. Developing a mechanism to study code trustworthiness. In Proceedings of the 50th Hawaii International Conference on System Sciences, Hilton Waikoloa Village, HI, USA, 4–7 January 2017; pp. 5817–5826.
22. Lingzi, X.; Zhi, L. An Overview of Source Code Audit. In Proceedings of the 2015 International Conference on Industrial Informatics-Computing Technology, Intelligent Technology, Industrial Information Integration, Wuhan, China, 3–4 December 2015; pp. 26–29.
23. Wanous, J.P.; Reichers, A.E.; Hudy, M.J. Overall job satisfaction: How good are single-item measures? *J. Appl. Psychol.* **1997**, *82*, 247–252. [\[CrossRef\]](#) [\[PubMed\]](#)
24. Faul, F.; Erdfelder, E.; Lang, A.G.; Buchner, A. G* Power 3: A flexible statistical power analysis program for the social, behavioral, and biomedical sciences. *Behav. Res. Methods* **2007**, *39*, 175–191. [\[CrossRef\]](#) [\[PubMed\]](#)
25. Jones, S.L.; Shah, P.P. Diagnosing the locus of trust: A temporal perspective for trustor, trustee, and dyadic influences on perceived trustworthiness. *J. Appl. Psychol.* **2016**, *101*, 392–414. [\[CrossRef\]](#) [\[PubMed\]](#)
26. Ryan, T.J.; Walter, C.; Alarcon, G.M.; Gamble, R.; Jessup, S.A. The influence of personality on code reuse. In Proceedings of the 52nd Hawaii International Conference on Systems Sciences, Maui, HI, USA, 7–11 January 2019; pp. 5805–5814.

