3-1-2020

Would you fix this code for me? Effects of repair source and commenting on trust in code repair

Gene M. Alarcon
Wright-Patterson AFB

Charles Walter
University of Mississippi

Anthony M. Gibson
Consortium of Universities for Global Health

Rose F. Gamble
University of Tulsa

August Capiola
Wright-Patterson AFB

See next page for additional authors

Recommended Citation

Alarcon, G. M., Walter, C., Gibson, A. M., Gamble, R. F., Capiola, A., Jessup, S. A., & Ryan, T. J. (2020). Would You Fix This Code for Me? Effects of Repair Source and Commenting on Trust in Code Repair. Systems, 8(1), 8. https://doi.org/10.3390/systems8010008

systems

MDPI

Article

Would You Fix This Code for Me? Effects of Repair Source and Commenting on Trust in Code Repair

Gene M. Alarcon [1,*], Charles Walter [2], Anthony M. Gibson [3], Rose F. Gamble [4], August Capiola [1], Sarah A. Jessup [1] and Tyler J. Ryan [5]

1 Air Force Research Laboratory, Wright Patterson AFB, OH 45433, USA; august.capiola.1@us.af.mil (A.C.); sarah.jessup.ctr@us.af.mil (S.A.J.)
2 Department of Computer and Information Science, University of Mississippi, University, MS 38677, USA; cwwalter@olemiss.edu
3 Consortium of Universities, Washington, DC 20036, USA; anthony.gibson.9.ctr@us.af.mil
4 Tandy School of Computer Science, University of Tulsa, Tulsa, OK 74101, USA; gamble@utulsa.edu
5 Department of Psychology, Wright State University, Dayton, OH 45435, USA; ryan.76@wright.edu
* Correspondence: gene.alarcon.1@us.af.mil

Received: 20 December 2019; Accepted: 11 March 2020; Published: 18 March 2020

Abstract: Automation and autonomous systems are quickly becoming a more engrained aspect of modern society. The need for effective, secure computer code in a timely manner has led to the creation of automated code repair techniques to resolve issues quickly. However, the research to date has largely ignored the human factors aspects of automated code repair. The current study explored trust perceptions, reuse intentions, and trust intentions in code repair with human generated patches versus automated code repair patches. In addition, comments in the headers were manipulated to determine the effect of the presence or absence of comments in the header of the code. Participants were 51 programmers with at least 3 years' experience and knowledge of the C programming language. Results indicated only repair source (human vs. automated code repair) had a significant influence on trust perceptions and trust intentions. Specifically, participants consistently reported higher levels of perceived trustworthiness, intentions to reuse, and trust intentions for human referents compared to automated code repair. No significant effects were found for comments in the headers.

Keywords: trust; automated program repair; human factors psychology

1. Introduction

The proliferation of software, generically referred to as computer code, in products ranging from watches to drones necessitates rapid code generation and repair as new bugs emerge during deployment. The urgency of code repair has led to the development of automated code repair processes, in which one software repairs another without human intervention. Though not yet mainstream technology, little is known about both how programmers perceive the use of automated program repair and the quality of repairs made to the code. Prior human factors research has identified trust as an important antecedent of reliance behaviors when humans interact with automated systems [1]. Research on how developers trust automated code repair can influence training requirements for how to deploy automated code repair and the development of the automated code repair tools, which can potentially increase reliance behaviors. The current study explored how programmers perceive changes made by a human versus changes made by an automated program repair software, GenProg. In the section below, we describe GenProg and expand on how the trust in automation literature can be leveraged to increase reliance on programs like GenProg.

2. Related Work

2.1. GenProg

The need for securing and correcting code in a timely manner has led to innovation, resulting in research toward automatic code repair. Previous researchers found that debugging code accounted for 50% of development costs in software [2]. There are two major activities related to debugging code: (1) the fault must be identified, and (2) the fault must be removed or repaired. Researchers have begun using automation to detect faults (e.g., static program analysis) [3], but those faults still need to be repaired by programmers. To remedy this situation, computer science researchers have been exploring automated repair programs that repair code, which can then be validated by testers [4,5]. Interest in such techniques has grown increasingly over the last 20 years, with more research articles published on the topic each year [6].

GenProg is an automated code repair process that uses genetic algorithms to create patches from existing code to repair the same code [5]. The input for GenProg is a faulty program and that program's test cases. The process utilizes genetic algorithms by taking other parts of the code base and mutating them to fit into the area of the code where it failed the test case. The process first localizes the faults given the test cases. Next, it changes the code in the localized fragments. Within this context, GenProg iteratively performs one of three operations: (1) insert a program statement into the next line of code, (2) delete a statement in the code, and (3) replace the code by both deleting and inserting. It conducts this in an iterative process. In other words, if the change passes the test cases, GenProg outputs the modified program. If it fails to pass the test cases, it revisits the process, adapting localized fragments until the modified code has passed all test cases. Although research has expanded the efficiency of automated code repair programs in the last two decades [7], little research has examined programmers' trust perceptions of these changes. The prior research on trust in automation can potentially elucidate the ways in which programmers perceive automated code repair programs.

2.2. Trust in Automation

Trust in automation has become an increasingly important area of research in the last few decades. Automation is "technology that actively selects data, transforms information, makes decisions, or controls processes" [1] (p. 50). Trust in automation is the belief the automated system functions efficiently in relation to performance, safety, and use rate and is directly and proportionally related to reliance [1]. Automation has various degrees of decision execution ranging from low (i.e., it alerts the user of an error) to high decision execution (it acts autonomously and fixes the error without input from the user) [8]. Higher levels of decision execution for automation inherently requires the system to also perform the lower levels of automation. In other words, for automation to perform higher decision execution, such as choice, it must also perform the lower level tasks of selection and situation awareness. For example, the static program analysis tool used by programmers is a situational awareness tool (i.e., it selects flaws in the code and alerts the user to the issues). In contrast, GenProg selects the problem areas, assesses the situation by determining which factorized code block has the issue or issues, determines a choice to repair (i.e., delete, insert, or both), and takes action to fix the issue. Thus, not all automation is created equally. Although little research exists on trust in automated code repair, researchers have extended the trust in automation research to trust in code [9], which we describe further below.

2.3. Trust in Code

Researchers in computer science have examined how programmers trust and reuse code for decades. They have explored the effects of code reuse [10,11], code formatting [12], and code documentation [13] on programmers' use of code. Although not specifically named as trust, the studies focused on reliance behaviors (i.e., reusing the code), which are related to trust. Psychologists have also become interested in the cognitive processes underlying a decision to trust code [9]. The automation

factors and dimensions mentioned above give insight into how people view automation, but they fail to provide a comprehensive theoretical model on how programmers process the information from the interaction. Alarcon and Ryan [14] adapted Chaiken's [15] heuristic–systematic processing model (HSM) to apply to human interactions with computer code.

The HSM is a dual-process model, which includes both heuristic and systematic processing [15]. Heuristic processing is characterized by lower cognitive effort, in which people make decisions based on cognitive biases, rules of thumbs, and norms. Heuristic processing often leads to quicker, but less accurate, assessments. In contrast, systematic processing entails in-depth and effortful assessments of stimuli, which often results in slower, but more accurate, assessments. A key distinction of the HSM from other dual-process models is the inclusion of the sufficiency principle, which states that perceivers are economy-driven. In other words, people will stop processing a situation once a desired level of confidence has been met; otherwise, the perceiver will continue to process information until sufficiency has been achieved.

Several experimental findings have highlighted the utility of the HSM in the context of trustworthiness of computer code. Alarcon and colleagues [16], for example, found that properties of code (e.g., readability and source of the code) served as cues programmers used to inform their trust toward the code. Specifically, when readability and perceived source were degraded, trust levels declined and human processing speed was shortened, despite that the code compiled successfully and was free from performance errors. These findings align with Alarcon and colleagues' interpretation of heuristic processing. However, when the organization of the code was degraded, participants found code from a reputable source was trustworthy. Participants spent more time examining the code (i.e., an indication of systematic processing) if it was from a reputable source, coming to a more accurate assessment. These results have been replicated across several studies [17–19]. Similarly, Alarcon and colleagues [20] found changing only the comments within the code—thus leaving the source code intact—influenced trust perceptions and manual time spent on code review that was similar to prior studies. Although these results have demonstrated the impact of trust in code, research is only beginning to explore the effects of code repair by different agents, such as computer programs that repair other programs [21].

2.4. Trust Biases

According to the HSM, biases act as heuristics that influence the way in which data is perceived and encoded [15,22]. Researchers have examined the similarities and differences between human–human and human–automation trust. Madhavan and Weigmann [23], for example, hypothesized a theoretical model comparing human–human to human–machine trust. The model integrates theoretical differences in trust judgements, monitoring behaviors, and schemas. Madhavan and Weigman drew from previous research in human–automation trust to construct the theoretical model. They suggested users have preconceived notions about automation, such as it performing perfectly. Additionally, users are generally more observant of automation errors, and their trust perceptions are based on the performance of the automation rather than knowledge when interacting with humans. For example, advice from a computer is perceived as more objective than advice from a human, despite being the same advice [24], indicating a bias towards computers. Additionally, people are more sensitive to errors made by automation, with sharper declines in trust when a computer made an error than when a human made one [25]. Interestingly, when participants received no information about a system's reliability, participants failed to use the system after errors occurred. Thus, according to previous findings, when people's automation bias (expectation that systems are always reliable) is violated, trust appears to decline quickly when paired with automation compared to when paired with a human. Based on the research presented above, we contend two factors will influence trust perceptions of program repair: (1) comments in the overall code, and (2) whether the repairs were performed by a human or an automated repair program (indicating a bias towards automation or human performance).

2.5. Comments

As mentioned above, comments are an important aspect of computer code, as comments help to improve readability and understandability [26]. For example, documentation comments, which are comments written right before a method or in the body of a method, assist coders in understanding the code [27,28], which increases transparency. Importantly, the trust in automation literature has focused on transparency as a key antecedent of trust in automation. For example, Lyons et al. [29] found trust in an automatic ground collision avoidance system in fighter jets increased when information about the functioning of the system was displayed on the screen. Similarly, Alarcon et al. [16] found transparency was a key factor in whether programmers would reuse code that was generated by other programmers, and research has consistently demonstrated transparency as a key factor in perceptions of software [14,20]. However, the relationship of comments and trust in code remains equivocal. Aman, Amasaki, Yokogawa, and Kawahara [30] found longer documentation comments were more prone to changes and required more repairs after release, indicating poorer code was being released with longer comments. Additionally, bad comments can influence the perception of the code, even when the code is functionally correct. Alarcon et al. [20] found manipulating comments alone in functional code reduced the trustworthiness and intentions to reuse the code, despite the source code being functional and error-free. However, comments are generally encouraged in code, as they help to aid in understanding and thus increase transparency. As such, code containing comments should be easier to understand when those comments communicate fixes to a patch.

Hypothesis 1. *Code repairs with Comments in the header will be perceived as more trustworthy.*

Hypothesis 2. *Code repairs with Comments in the header will be reused more.*

2.6. Code Reputation and Cognitive Biases

The source of the code repair may influence perceptions of code trustworthiness. The reputation factor found by Alarcon et al. [16] is defined as "perceived code trustworthiness cues based on external information [such as] the source of the code, information available in reviews, and the number of current users of the software" [20] (p. 112). In the automation literature, de Vries and Midden [31] found that a system's reputation affected participants' expectancies of the system. Specifically, when the performance details of the automation were absent, participants relied on reputational cues to form perceptions of trust and dictate trust behaviors. Thus, the reputation of a system influences the extent to which humans trust and use that system. Additionally, in the computer science literature, researchers have explored differences in correctness perceptions of machine-generated and human-generated patches [32,33], although this research focused on correctness rather than trust in the system. Trust is inherently different as it refers to the willingness to be vulnerable to the system, rather than the correctness of each patch. These studies demonstrate the reputation, and specifically the source, associated with the code repairs, such as a human or automation, can influence the perceptions of the code repairs.

Parasuraman et al. [34] described various levels of automation ranging from low (i.e., the human controls and performs all aspects) to high (i.e., the automation performs actions autonomously). In the instance of automated code repair, the system engages in higher-level decision-making and problem solving, which may come at a cost. Automation highest on the decision execution scale takes direct actions, such as GenProg, which reduce the programmer's cognitive load. The programmer, however, has less control and predictability of a system's behavior [35], which may reduce trust in the system [36]. The ability to select, assess, choose, and adaptively deploy appropriate actions are foundational automation attributes, yet these decision execution capabilities come with inherent expectations and a need for transparency regarding the accuracy of the system's decision logic [37]. Automated systems may be capable of performing a task (e.g., code repair), but the end user must trust the system to perform the task accurately for the system to be utilized to its full potential. However,

trust in automation may differ from trust towards humans, as people typically are more willing to accept errors from other humans (i.e., the automation bias) [38].

Programmers trust other human programmers when they accept code revisions from them [20]. Accepting a code revision may be based on aspects of the referent such as reputation of the repository or the experience and/or expertise of the programmer that made the changes. However, there may be differences in how programmers trust automated repair programs compared to humans that repair software. For example, when users were paired with automation, automation faults reduced their self-confidence in ability to perform the task and reduced trust perceptions in the automation [39]. In contrast, when told they were paired with a human partner, faults did not affect self-confidence, even though all participants were paired with an automated system. These results again illustrate the biases in favor of humans when mistakes are made, as human partners did not have as extreme trust reductions after faults as automation did. Additionally, it may depend on the experience of the human partner. Smith and colleagues [40] found automated repair program patches and novice patches both overfit and the automated repair programs performed no worse than the novice developers. These results indicate the difference between the automated repair programs and humans may also be moderated by experience.

Lastly, Ryan and colleagues [21] found differences in perceptions of automated code repair when participants were told the repairs were done by a human versus GenProg. Specifically, participants' trust intentions declined significantly when they were told that the repairs made were done by a human compared to when participants were told that the repairs were made by GenProg. Unbeknownst to participants, however, all repairs were made by GenProg. In this study, participants trusted GenProg the same after reviewing stimuli, but their trust towards the human decreased, which may have occurred due to the violation of their expectancies. That is, GenProg refactors code when repairing it, and the changes to the code—although technically functional—fail to resemble changes people make in the real-world. Thus, the comparison Ryan and colleagues' made between the effects of human versus automated code repairs on programmer trust were somewhat limited, as both the human and GenProg conditions contained GenProg repairs. The current study seeks to remedy this shortcoming by giving participants actual human repairs or actual GenProg repairs to review.

One source of bias may be the type of repairs the program makes in comparison to the repairs humans make. Nakajima, Higo, Yokoyama, and Kusuoto [41] found human developers made more changes to functions, program structure, and added/deleted more program statements than GenProg. In contrast, GenProg only passes cases based on the test cases, whereas humans repairing the code may catch other faults in the code during the repair process. Indeed, this may be the reason for the discrepancy between the numbers of changes GenProg makes compared to the number of changes a human makes. Programmers use deeper and broader knowledge structures of the program's purpose, modifying the program by intended use rather than by test cases. This should provide both higher trustworthiness perceptions and also lead to higher reuse intentions.

Hypothesis 3. *Code repairs done by a human will be perceived as more trustworthy than code repairs done by GenProg.*

Hypothesis 4. *Code repairs done by a human will be reused more than code repairs done by GenProg.*

As noted by Madhavan and Weigman [23], humans hold certain biases towards automation. In particular, humans have lower perceptions of system trustworthiness when they have high self-confidence to complete the task [42]. In addition, trust has been defined as a social construct and humans have a tendency to over-trust humans, as distrusting other humans has a more negative connotation than distrusting a machine [43]. Similarly, human programmers and their changes to code may be perceived as more familiar than a machine and its changes, increasing trust [23]. In the context of the HSM of code, the source of the code repairs can act as a cue that triggers heuristic processing for evaluating whether or not to trust code when little other information is available.

As such, the biases mentioned will result in lower intentions to trust automated program repair (i.e., GenPog) than a human programmer.

Hypothesis 5. *Intention to trust the referent will be higher in the human condition than in the automated repair program condition.*

3. Method

3.1. Participants

A total of 51 programmers were recruited to participate in the current study and were compensated $50 (USD) for participating. Participants were required to have a minimum of four years of programming experience and experience with the C programming language. The sample was primarily male (68.6%) with a mean age of 27.72 (SD = 7.75), a mean of 8.21 (SD = 5.22) years total programming experience, and 27.45% stated they used C on a weekly basis.

3.2. Measures

3.2.1. Trustworthiness

We used four items to asses overall trustworthiness perceptions of the repairs (i.e., assessing perceived trustworthiness, maintainability, performance, and transparency perceptions). The items were chosen because they are the main constructs in the trust in code research that can be ascertained from the code itself [16]. Participants indicated their perceptions of trustworthiness with the item "How trustworthy do you find this repair?"; "How maintainable do you find this repair?"; "How transparent do you find this repair?"; and "How well do you think this repair will perform?" on a scale ranging from 1 (*Not at all Trustworthy*) to 7 (*Very Trustworthy*). The items all inter-correlated well and the scale had adequate reliabilities at each time point (see Table 1).

3.2.2. Trust Intentions

We adapted Mayer and Davis' [44] trust intentions scale to assess intentions to trust the referent (i.e., the software repairer). For a description of the referents, see the procedure below. The scale consisted of four items. All items were rated on a 5-point Likert scale (1 = *Strongly Disagree* to 5 = *Strong Agree*). The first and third items were reverse coded. We adapted the scale to reflect the referent being assessed. An example item is "I would be comfortable giving [human or automated code repair referent] a task or problem which was critical to me, even if I could not monitor their actions." Participants rated their intentions to trust the referent once before beginning the experiment and once after they had finished reviewing all code stimuli. Additionally, participants were asked with a single item whether they would endorse the code repair for use with "Use" or "Don't Use" as response options. This provided a single measure of programmers' intention to trust each of the code repairs should they be using the repaired code.

3.3. Stimuli

Participants reviewed and assessed repairs made to source code written in the C programming language. The source code was taken from the ManyBugs benchmark [45] and then run through Genprog to produce repairs. While Genprog does suffer from overfitting [46], the bugs we show closely match previous Genprog patches that were rated by experts as either "correct" (the patch matches or nearly matches the actual human-created patch) or "plausible" (the patch was rated by human experts as a viable patch for the bug). The analysis of the original Genprog patches is available online [47]. As such, all the patches were 100% reliable. The study utilized a 2 × 2 between-subject design, with 5 within-subject trials. The between-subject factors consisted of the Commenting (i.e., comments in the headers of the code vs. no comments in the headers of the code) and the Source of the repairs (i.e.,

human generated repairs vs. automation generated repairs). The 5 within-subject trials consisted of 5 different pieces of source code and their repairs in a diff. Diffs are utilities that display the differences between two files (see Figure 1). In the current study, the diffs were displayed such that the left side of the screen displayed the code prior to repair, and the right side of the screen displayed the code after the repair. Participants were randomly assigned to one of the four conditions and completed each of the five trials.

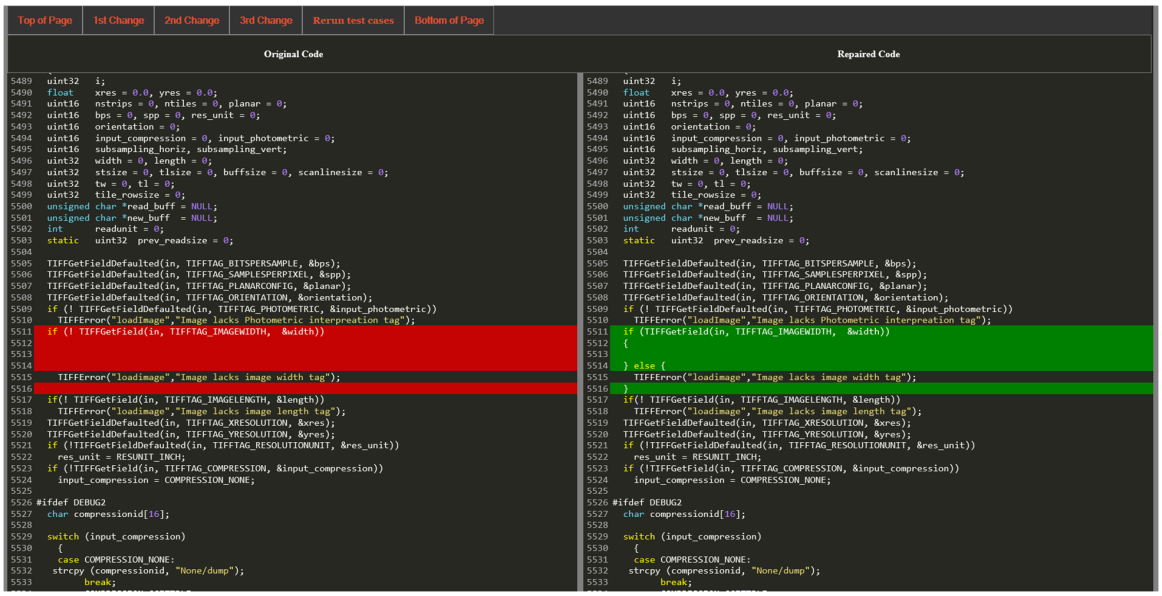

Figure 1. Example of diff stimuli.

3.4. Procedure

After being greeted, participants filled out background demographic and personality surveys, and completed training on how to review the code. After completing the surveys, participants were then read a description of GenProg or "Bill." For the GenProg condition, a brief description about GenProg and how it creates and inserts code patches was read. For the human condition, a summary of an experienced computer programmer that worked for a local government contractor was provided. We used the name "Bill" to refer to the human computer programmer. We provided a description of the human to keep the experiment balanced regarding prior information. The descriptions of both are provided in the Supplementary Materials. After being read the referent condition script, participants rated their intentions to trust the referent (Bill or GenProg). Participants then viewed each diff and rated each on trustworthiness. After the experiment was completed, participants then rated their intentions to trust the referent a final time. Participants were then debriefed and provided financial remuneration.

4. Results

Two participants completed the trust intentions scale prior to hearing the back story, and one participant answered survey questions unrelated to the specific code repairs. As such, two participants were excluded from the trust intentions analyses (N = 49) and one participant was excluded from the trustworthiness and reuse analyses (N = 50). Additionally, the human repair no commented condition had a 100% use endorsement rate, and the human repair commented condition had a 35.22% endorsement rate. Upon further inspection of the code, there was only a minimal change to the code. The first was replacing a logical "if" statement, while the second was combining two existing lines into one line of code. As such, there was only one minor technical change to the code. The high use rates due to the small change to the code probably led participants to decide it was a safe change, resulting in the 100% use rate. As such, we deleted code four from all analyses. Cronbach's alphas, correlations,

means, and standard deviations for trustworthiness and trust intentions at their respective time points are illustrated in Table 1.

Table 1. Means, standard deviations, and zero-order correlations of study variables.

Variable	*M*	*SD*	1	2	3	4	5	6
1. Trustworthy.T1	4.52	1.67	(0.75)					
2. Trustworthy.T2	4.60	1.50	0.58 **	(0.71)				
3. Trustworthy.T3	4.51	1.55	0.42 **	0.54 **	(0.80)			
4. Trustworthy.T5	4.27	1.92	0.46 **	0.42 **	0.59 **	(0.85)		
5. TrustIntention.T1	3.09	0.67	0.16	−0.05	0.29	0.44 **	(0.45)	
6. TrustIntention.T5	2.58	0.88	0.49 **	0.46 **	0.51 **	0.66 **	0.51 **	(0.76)

$N = 42$ to 50. Reliabilities for each scale are reported on the diagonal in parentheses; ** $p < 0.01$.

4.1. Trustworthiness

We used the *nlme* package [48] in R [49] to conduct linear mixed effects models to explore differences in trustworthiness perceptions between Source and Commenting factors across the four time points. The *F*-statistic and Type III sum of squares were considered when interpreting the effects. We tested several error variance–covariance structures and chose the model with the best fit based on the Akaike Information Criterion (AIC) and Bayesian Information Criterion (BIC), as per Liu, Rovine, and Molenaar [50]. The first order auto-regressive error variance–covariance structure fit the data the best (AIC = 700.03, BIC = 762.02), indicating assessments closer in time were more closely associated than assessments occurring further away in time.

Results of the full-factorial model are displayed in Table 2. We observed a significant main effect of Source [$F(1, 46) = 10.29$, $p = 0.002$]. As illustrated in Figure 2, programmers perceived code repair by GenProg (*EMMean* = 3.95, *SE* = 0.25) to be significantly less trustworthy than code repaired by a human (*EMMean* = 5.02, *SE* = 0.25). We found no significant main effect of Commenting, no significant main effect for Time, and there were no significant interactions (see Table 2). The entire cell estimated marginal means and standard errors are reported in Table S1 of Supplementary Material for interested readers.

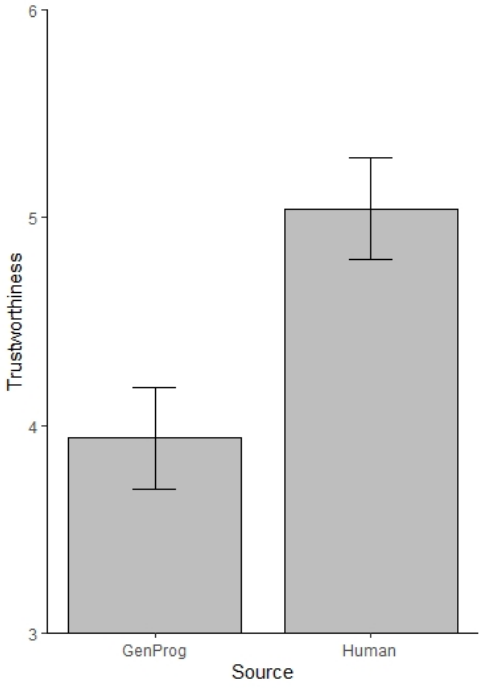

Figure 2. Perceived trustworthiness across source of the code repair.

Table 2. Mixed effects regression model for differences in trustworthiness across source and commented factors.

Predictor	*df*	*F*
Time	3, 131	0.51
Source	1, 46	10.29 *
Commented	1, 46	1.09
Time × Source	3, 131	1.54
Time × Commented	3, 131	0.92
Source × Commented	1, 46	0.43
Source × Commented × Time	3, 131	0.38

$N = 50$. Time was entered as an ordered factor. The error structure adopted was First-order autoregressive. Source = Human versus GenProg. Commented = Comments vs. No Comments. * $p < 0.01$.

4.2. Trust Intentions

Next, we conducted a repeated measure analysis of variance (RM ANOVA) on the trust intentions scale using the R *afex* package [49,51]. We used an RM ANOVA because the correlations between measurement points are constrained with only two time points. Results indicated a significant main effect for Source [$F(1,42) = 11.62$, $p < 0.001$], and a main effect for Time [$F(1,42) = 17.24$, $p < 0.001$]. However, all other effects were non-significant (see Table 3). Similar to perceived trustworthiness, participants had higher trust intentions towards the human programmer (*EMMean* = 3.15, *SE* = 0.12) compared to GenProg (*EMMean* = 2.54, *SE* = 0.12; see Figure 3). For those interested, the entire cell estimated marginal means and standard errors are reported in Table S2 of Supplementary Material.

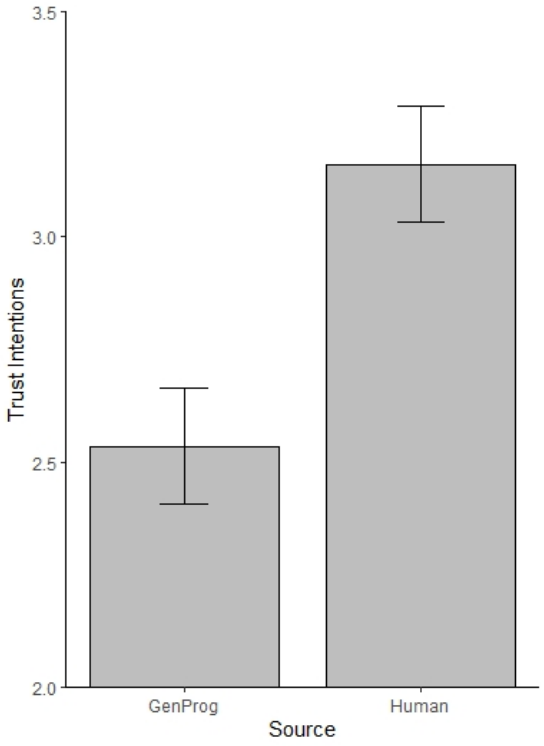

Figure 3. Perceived trust intentions across source of the code repair.

Table 3. Repeated measures ANOVA showing differences in trust intentions by source and commented factors.

Predictor	*df*	*F*	η^2_p
1. Time	1, 42	17.24 *	0.29
2. Source	1, 42	11.62 *	0.22
3. Commented	1, 42	1.37	0.03
4. Time × Source	1, 42	1.28	0.03
5. Time × Commented	1, 42	0.14	0.00
6. Source × Commented	1, 42	0.22	0.01
7. Source × Commented × Time	1, 42	0.02	0.00

$N = 49$. Two participants were removed because of missing data. Time was entered as an ordered factor. Source = Human versus GenProg. Commented = Comments vs. No Comments. * $p < 0.01$.

4.3. Use Endorsement

We used a generalized linear mixed effects model to analyze the effects of Source, Commenting, and Time on participant endorsement of code for use, as use endorsement was a binary outcome variable. We used the Type-III Wald's χ^2 statistic from the *RVAideMemoire* package in R [49,52] to interpret the main effects and interaction term. Source had a significant influence on reuse [Wald χ^2 (1) = 9.02, $p = 0.003$]. Figure 4 illustrates participants were more likely to reuse repairs from a human (*EMMean* = 75.90, *SE* = 0.07) rather than GenProg (*EMMean* = 41.00, *SE* = 0.08). There was no main effect of Commenting or Time. Additionally, none of the interactions were significant (see Table 4). The entire cell estimated marginal means and standard errors are reported in Table S3 of Supplementary Material for interested readers.

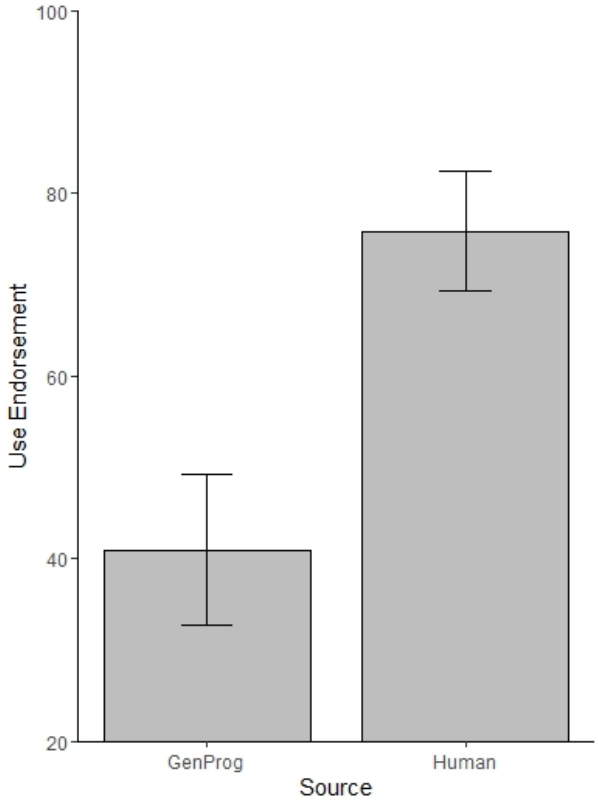

Figure 4. Probability of use endorsement across source of the code repair.

Table 4. Mixed effects regression model showing differences in reuse for source and commented factors.

Predictor	*df*	Wald's χ^2
Time	3	2.69
Source	1	9.02 *
Commented	1	1.00
Time × Source	3	4.16
Time × Commented	3	0.88
Source × Commented	1	0.25
Source × Commented × Time	3	0.11

$N = 50$. Time was entered as an ordered factor. The error structure adopted was first-order autoregressive. Source = Human versus GenProg. Commented = Comments vs. No Comments. * $p < 0.01$.

4.4. Qualitative Coding

We qualitatively coded participants' remarks about each piece of code for reputation, transparency, and performance to better understand the perceptions of the referent repairs (human or GenProg). Additionally, we coded any remarks about the code itself. Table 5 illustrates the results of the qualitative coding. As noted in the table, participants made very few positive reputation remarks concerning the programmer (Bill or GenProg), with only six positive reputation remarks made across a total of two pieces of code. More remarks were negative than positive about the referent in terms of code repair, but there did not appear to be a substantial difference between the human and GenProg conditions. In contrast, participants had 50% more negative transparency remarks about GenProg. In addition, participants also had twice as many positive transparency remarks about human repairs than the GenProg repairs. Lastly, participants had twice as many negative remarks about GenProg's performance compared to the human condition. However, positive remarks about performance did not appear to differ.

Table 5. Qualitative counts of remarks made about reputation, transparency and performance and remarks about the code itself.

	Reputation				**Transparency**				**Performance**				**Code**			
	Human		**GenProg**		**Human**		**GenProg**		**Human**		**GenProg**		**Human**		**GenProg**	
Code	**(+)**	**(−)**	**(+)**	**(−)**	**(+)**	**(−)**	**(+)**	**(−)**	**(+)**	**(−)**	**(+)**	**(−)**	**(+)**	**(−)**	**(+)**	**(−)**
1	0	3	2	1	3	7	0	13	2	6	6	10	0	0	0	0
2	0	0	0	0	3	12	6	14	5	13	8	10	0	2	0	1
3	4	2	0	1	4	17	3	14	8	6	6	16	0	0	0	0
4	0	3	0	2	7	6	1	15	5	7	5	20	1	0	0	0
5	0	5	0	5	6	11	0	25	14	10	3	34	1	1	1	1
Total	4	13	2	9	23	53	10	81	34	42	28	90	2	3	1	2

(+) = Positive remark. (−) = Negative remark.

5. Discussion

The current study explored biases toward code repaired by an automated code repair process (GenProg) in comparison to the same code repaired by a human. Results indicate programmers found human repairs more trustworthy, were more willing to be vulnerable to a human, and intended to reuse human repairs more compared to GenProg. Interestingly, including the comments in the header of the code had no effect on trustworthiness perceptions, trust intentions, or use. Overall, the current study illustrated the effects of biases against automated code repair and elucidates some of the past findings on trust towards automated repair tools [21].

5.1. Source

The source of the repairs had a significant influence on all variables assessed in the current study. This is not surprising as previous research in both computer science and psychology [14,16,20] has demonstrated the source of the code (i.e., reputation) as an important factor that influences reuse and trust perceptions in code. The current study also supports the hypotheses of Madhavan and Weigmann [23] that humans hold biases in perceptions against automation when they perceive they can perform the task themselves, as participants consistently rated GenProg lower than a human. In the current study, participants perceived a human as more trustworthy (i.e., trust intentions) even before seeing the code repairs. Participants also perceived the repairs as more trustworthy and were more likely to use repairs from a human rather than a computer-generated repair, despite all the repairs from both the human and automated code repair process accurately fixing the test cases. This may be because previous research has demonstrated the types of repairs made by GenProg are repairs that are similar to novices [40].

Qualitative analyses indicated participants perceived human code repairs as more transparent. This is an issue with programs such as GenProg which have to refactor the source code to perform the code repairs [5]. In addition, the changes that GenProg typically makes are not written in a manner that is necessarily intuitive to humans. Automated code repair processes do not attempt to replicate code written by humans, but rather create a patch that passes the test cases. As such, the changes often look odd to programmers, especially those who do not have experience with the process used by GenProg to repair the code. Interestingly, positive mentions of performance did not show a bias. However, negative perceptions of performance showed a clear trend against GenProg. These results as seen in negative comments toward automation are similar to previous research by Jian et al. [43] that found participants were more likely to use distrust words when describing interactions with computers than when describing interactions with humans. However, in contrast to that study, the current study found roughly the same amount of trust terms toward the human and automation. One explanation for the current findings may be participants' ability to monitor the behaviors by reviewing the repairs themselves in the diffs. It may also be associated with the programmer's way of writing certain types of functions and repairs, as there are many ways to write a program. One alternative may also be that programmers do not understand how GenProg operates "under the hood." Specifically, the process by which GenProg uses extant information and creates patches that pass test cases may not be understood by programmers, and this lack of transparency leads to a lack of trust [29]. Future research should investigate whether transparency is indeed the most important factor leading to differences between trust towards human and automated repair tools in software evaluation contexts. This could be done by systematically manipulating transparency characteristics in these contexts to investigate the possible interaction between transparency and source of repair, which would further elucidate the role of automation bias on trust in code. Importantly, transparency may interact with other variables in popular models of human–automation trust. For example, Rusnock, Miller, and Bindewald [53] present a model of human–automation trust. Transparency of the automation will likely moderate the relationship of automation performance on trust and automation predictability on trust.

In the context of large psychological theories, the biases against GenProg can be explained with Madhavan and Weigman's [23] model. Participants may have been more critical of automated code repair because the participants may have felt they could adequately repair the source code (i.e., stimuli), and thus not need the automation. In this context, programs such as GenProg suffer when they are unclear in their fixes because the user may feel it is more expeditious to simply perform the task him/herself [54]. As such, any deviation from actions programmers typically perform is perceived as untrustworthy. Although the code repairs fixed any errors and resulted in all test cases passing, the participants may have noticed other aspects of the code that could run more efficiently or could be written better. If these changes were absent, it could result in negative remarks towards the referent. This is especially true in the "Bill" condition, as humans are perceived as more adaptable and able to make changes outside the scope of the task [23]. Additionally, in the current study, workload, which

has been hypothesized to influence reliance behaviors [53], was relatively low as the participants were only performing the experimental task (and not teaming with a code repair assistant [human or automation] to repair code themselves). As such, programmers may be more likely to utilize the automated code repair in tasks associated with increased workload.

5.2. Commenting

We also assessed the influence of comments placed in the code headers. Comments in the header had no effect on any of the trust dependent variables. In retrospect, the comments in the header may have not had an effect on trust intentions towards the referent as the comments were provided with the code prior to the referent making the repairs and after the referent made the repairs. In other words, the referent did not make any of the comments. As such, participants may not have assumed the comments to have been made by and thus did not alter their trust toward the referent. Comments failed to influence trustworthiness perceptions or reuse, which was contrary to our hypotheses. This may have occurred for several reasons. First, comments are useful for understanding the code [26]. However, the comments in the current study described aspects of the overall code rather than explaining changes associated with the code repair. In other words, neither GenProg nor "Bill" made any comments as to their changes. As such, comments may not have facilitated processing or understanding, as participants only attended to code aspects that were changed or relevant to the change. Second, comments in the headers act as section breaks and assist in understanding the larger architecture [27,30] but may not facilitate understanding when the code or code repair is relatively terse. Although the architectures used in the current study were large, participants were guided to only focused on certain sections with relatively few repairs to retain parsimony. As such, the comments in the sections do not facilitate understanding as participants were able to read the few lines of code and ascertain the changes themselves. Participants may not have even read the comments in the headers, given their task of understanding the repairs and not the overall architecture. Future research may wish to place comments in locations within the diff so that programmers can attribute them to a referent more intuitively, allowing a clearer assessment of the effect of comments on trust towards code repair referents. Future work may also use different techniques to determine whether or not participants actually read the comments within the code (e.g., eye-tracking, having participants answer post-task questions to confirm they indeed read the code).

5.3. Implications

The current study has several implications for theory and practice. First, although automated code repair processes have come a long way in the last 20 years, biases still exist against automation. Despite the repairs made by GenProg passing all test cases, participants still trusted the repairs less and used the repairs less than human repairs. Research should further explore the reasons for these differences. It may be that differences in how the changes are made influence the trust perceptions, which can be altered by engineers to make the repairs more human-like. However, if it is the nature of the referent, i.e., GenProg or human, then the biases humans hold should be explored and training may help to increase the trust and use in the system, which is related to our second point. Second, research has demonstrated GenProg makes repairs similar to novice programmers [40]. Although all the patches in the current study repaired the code to pass all test cases, there may have been other issues that experienced programmers would not have missed such as overfitting. Third, GenProg lacks transparency in the process it uses to come to a conclusion, or in the changes it makes to the code. Research in the psychology field have demonstrated transparency as a key factor in trusting automation [1]. Importantly, transparency should be added to modeling methods in the literature. As discussed above, transparency likely moderates much of the automation predictors of trust in the modeling methods such as predictability and performance [1]. In other words, it is not enough to know the automation is performing the task well but it is also necessary to know *why* it is performing the task well. Fourth, there are clear biases against automated code repair processes. Across all dependent

variables, the automated repair was viewed as less trustworthy. This is especially relevant in trust intentions of the referent. Participants had significantly lower trust intentions towards GenProg than the human condition, even before seeing the repairs. Although these biases are modeled in popular modeling methods [53] via propensity to trust automation, they can have moderating influences on other aspects of the model. Finally, by expanding the experimental design to include actual changes from both GenProg and a human programmer, the present study has elucidated some of the outstanding issues from Ryan and colleagues' [21] work.

5.4. Limitations

The current study is not free from limitations. First, the trust intentions scale demonstrated lower than acceptable reliability estimates. It should be noted the trust intentions measure was developed for assessing managers in an organizational context [44]. As such, it may not be suitable in an automation environment. However, it should be noted there is no automation trust intentions scale in the literature to date that assesses both human and automation referents. Also, the trust intentions internal consistency value increased over time, once participants gained more information about the code repair source. Participants should answer trust intention items more consistently with increased knowledge about the source.

Second, participants in the current study only viewed the diffs associated with the code repairs. It may be that seeing the code repairs happen in real time may affect perceptions of the referent and the repair itself. For example, GenProg may take a long time to find a solution to a problem and may not come to the same solution in the same amount of time. This may also influence trust perceptions.

Third, it is possible that the GenProg patches may have been overfitted, despite our best efforts to ensure correct or plausible patches. While we do not believe this would result in a large difference between GenProg and humans when shown the same patches, it is possible that code reviewers may examine the computer-generated code more closely, resulting in them recognizing a potential issue.

Fourth, it should be noted that in the current study all repairs performed by both human and automation repaired the test cases. As other researchers have noted, the reliability of the automation influences the trust in the system. By only displaying instances where the referent repaired the program for all test cases, we have limited our results to instances where the referent performs well, often referred to as reliability. Indeed, in models of automation performance is an important factor in trust [36].

Lastly, reliance on automation is important in environments where cognitive resources are limited. As such, reliance on systems such as GenProg may be dictated by how much time the programmer has to inspect the code. In the current study, participant's sole task was to inspect the code repairs and the HTML format moved them to the appropriate area of the architecture where the repair occurred. In real life scenarios, reliance may fluctuate depending on how many resources the user has available.

Supplementary Materials: The following are available online at http://www.mdpi.com/2079-8954/8/1/8/s1, Table S1: Estimated Marginal Means and Standard Errors for Trustworthiness by Conditions and Time, Table S2: Estimated Marginal Means and Standard Errors for Trust Intentions by Conditions and Time, Table S3: Marginal Means and Standard Errors for Reuse by Conditions and Time.

Author Contributions: Conceptualization, G.M.A., R.F.G., C.W. and T.J.R.; Methodology, R.F.G. and C.W.; Software, R.F.G. and C.W.; Investigation, G.M.A., S.A.J., T.J.R.; Data Curation, G.M.A., A.M.G., T.J.R.; Writing-Original Draft Preparation, G.M.A., C.W., R.F.G.; A.C.; S.A.J.; A.G.; Writing-Review & Editing, G.M.A., C.W., R.F.G.; A.C.; S.A.J.; A.M.G.; 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.

Ethical Statements: The current study was reviewed by the Air Force Research Laboratory Institutional Review Board and was approved for human subjects research approval number FWR20170149H.

References

1. Lee, J.D.; See, K.A. Trust in automation: Designing for appropriate reliance. *Hum. Factors* **2004**, *46*, 50–80. [CrossRef]
2. Britton, T.; Jeng, L.; Carver, G.; Cheak, P.; Katzenellenbogen, T. *Reversible Debugging Software*; Technical Report for University of Cambridge Judge Business School: Cambridge, UK, 2013.
3. German, A. Software static code analysis lessons learned. *Crosstalk* **2003**, *16*, 19–22.
4. Arcuri, A. On the automation of fixing software bugs. In Proceedings of the 30th International Conference on Software Engineering, Leipzig, Germany, 10–18 May 2008; pp. 1003–1006.
5. Weimer, W.; Nguyen, T.; Le Goues, C.; Forrest, S. Automatically finding patches using genetic programming. In Proceedings of the 31st International Conference on Software Engineering, Vancouver, BC, Canada, 16–24 May 2009; pp. 364–374.
6. Gazzola, L.; Mariani, L.; Micucci, D. Automatic Software Repair: A Survey. In Proceedings of the 2018 IEEE/ACM 40th International Conference on Software Engineering, Gothenburg, Sweden, 27 May–3 June 2018; p. 1219.
7. Martinez, M.; Monperrus, M. Astor: Exploring the design space of generate-and-validate program repair beyond GenProg. *J. Syst. Softw.* **2019**, *151*, 65–80. [CrossRef]
8. Wickens, C.D.; Li, H.; Santamaria, A.; Sebok, A.; Sarter, N.B. Stages and levels of automation: An integrated meta-analysis. In Proceedings of the Human Factors and Ergonomics Society Annual Meeting, San Francisco, CA, USA, 27 September–1 October 2010; Volume 54, pp. 389–393.
9. 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. Cog. Eng. Decis. Mak.* **2017**, *11*, 107–121. [CrossRef]
10. Banker, R.D.; Kauffman, R.J. Reuse and productivity in integrated computer-aided software engineering: An empirical study. *MIS Q.* **1991**, *15*, 375–401. [CrossRef]
11. Lim, W.C. Effects of reuse on quality, productivity, and economics. *IEEE Softw.* **1994**, *11*, 23–30. [CrossRef]
12. Albayrak, Ö.; Davenport, D. Impact of maintainability defects on code inspections. In Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, Bolzano-Bozen, Italy, 16–17 September 2010; ACM: New York, NY, USA, 2010; pp. 50–53.
13. Beller, M.; Bacchelli, A.; Zaidman, A.; Juergens, E. Modern Code Reviews in Open-Source Projects: Which Problems Do They Fix? In Proceedings of the 11th Working Conference on Mining Software Repositories, Hyderabad, India, 31 May–1 June 2014; ACM: New York, NY, USA, 2014; pp. 202–211.
14. 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, Waikoloa Village, HI, USA, 3–6 January 2018.
15. 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]
16. 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]
17. Capiola, A.; Nelson, A.D.; Walter, C.; Ryan, T.J.; Jessup, S.A.; Alarcon, G.M.; Gamble, R.F.; Pfahler, M.D. Trust in Software: Attributes of Computer Code and the Human Factors that Influence Utilization Metrics. In Proceedings of the International Conference on Human-Computer Interaction, Orlando, FL, USA, 26–31 July 2019; Springer: Cham, Switzerland, 2019; pp. 190–196.
18. 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 Rrelationship. In Proceedings of the International Conference on Human-Computer Interaction, Las Vegas, NV, USA, 15–20 July 2018; Springer: Cham, Switzerland, 2018; pp. 370–376.
19. 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, Waikoloa Village, HI, USA, 4–7 January 2017.
20. 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]

21. Ryan, T.J.; Alarcon, G.M.; Walter, C.; Gamble, R.; Jessup, S.A.; Capiola, A.; Pfahler, M.D. Trust in Automated Software Repair. In Proceedings of the International Conference on Human-Computer Interaction, Orlando, FL, USA, 26–31 July 2019; Springer: Cham, Switzerland, 2019; pp. 452–470.
22. Chaiken, S.; Maheswaran, D. Heuristic processing can bias systematic processing: Effects of source credibility, argument ambiguity, and task importance on attitude judgment. *J. Personal. Soc. Psychol.* **1994**, *66*, 460–473. [CrossRef]
23. Madhavan, P.; Wiegmann, D.A. Similarities and differences between human–human and human–automation trust: An integrative review. *Theor. Issues Ergon. Sci.* **2007**, *8*, 277–301. [CrossRef]
24. Dijkstra, J.J. User agreement with incorrect expert system advice. *Behav. Inf. Technol.* **1999**, *18*, 399–411. [CrossRef]
25. Dzindolet, M.T.; Pierce, L.G.; Beck, H.P.; Dawe, L.A.; Anderson, B.W. Predicting misuse and disuse of combat identification systems. *Mil. Psychol.* **2001**, *13*, 147–164. [CrossRef]
26. Tenny, T. Program readability: Procedures versus comments. *IEEE Trans. Softw. Eng.* **1988**, *14*, 1271–1279. [CrossRef]
27. Aman, H. An Empirical Analysis of the Impact of Comment Statements on Fault-Proneness of Small-Size Module. In Proceedings of the 2012 19th Asia-Pacific Software Engineering Conference, Hong Kong, China, 4–7 December 2012; IEEE: Washington, DC, USA, 2012; pp. 362–367.
28. Aman, H.; Amasaki, S.; Sasaki, T.; Kawahara, M. Empirical Analysis of Change-Proneness in Methods Having Local Variables with Long Names and Comments. In Proceedings of the 2015 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, Beijing, China, 22–23 October 2015; IEEE: Washington, DC, USA, 2015; pp. 1–4.
29. 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]
30. Aman, H.; Amasaki, S.; Yokogawa, T.; Kawahara, M. A Doc2Vec-Based Assessment of Comments and Its Application to Change-Prone Method Analysis. In Proceedings of the 2018 25th Asia-Pacific Software Engineering Conference, Nara, Japan, 4–7 December 2018; IEEE: Washington, DC, USA, 2018; pp. 643–647.
31. De Vries, P.; Midden, C. Effect of indirect information on system trust and control allocation. *Behav. Inf. Technol.* **2008**, *27*, 17–29. [CrossRef]
32. Le, D.X.B.; Bao, L.; Lo, D.; Xia, X.; Li, S.; Pasareanu, C. On Reliability of Patch Correctness Assessment. In Proceedings of the 2019 IEEE/ACM International Conference on Software Engineering, Montréal, QC, Canada, 25 May–1 June 2019; IEEE: Washington, DC, USA, 2019; pp. 524–535.
33. Wang, S.; Wen, M.; Chen, L.; Yi, X.; Mao, X. How Different is it between Machine Generated and Developer Provided Patches? An Empirical Study on the Correct Patches Generated by Automated Program Repair Techniques. In Proceedings of the 2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, Porto Galinhas, Brazil, 19–20 September 2019; ACM: New York, NY, USA, 2019; pp. 1–12.
34. Parasuraman, R.; Sheridan, T.B.; Wickens, C.D. A model for types and levels of human interaction with automation. *IEEE Trans. Syst. Man Cybern. Part. A Syst. Hum.* **2000**, *30*, 286–297. [CrossRef]
35. Chen, J.Y.; Barnes, M.J. Human–agent teaming for multirobot control: A review of human factors issues. *IEEE Trans. Hum.-Mach. Syst.* **2014**, *44*, 13–29. [CrossRef]
36. Schaefer, K.E.; Chen, J.Y.; Szalma, J.L.; Hancock, P.A. A meta-analysis of factors influencing the development of trust in automation: Implications for understanding autonomy in future systems. *Hum. Factors* **2016**, *58*, 377–400. [CrossRef]
37. Arkin, R.C.; Ulam, P.; Wagner, A.R. Moral decision making in autonomous systems: Enforcement, moral emotions, dignity, trust, and deception. *Proc. IEEE* **2012**, *100*, 571–589. [CrossRef]
38. Mosier, K.L.; Skitka, L.J. Human Decision Makers and Automated Decision Aids. In *Automation and Human Performance: Theory and Applications*; Parasuraman, R., Mouloua, S., Eds.; Lawrence Erlbaum: Mahwah, NJ, USA, 1996; pp. 201–220.
39. Lewandowsky, S.; Mundy, M.; Tan, G.P.A. The dynamics of trust: Comparing humans to automation. *J. Exp. Psychol. Appl.* **2000**, *6*, 104–123. [CrossRef] [PubMed]
40. Smith, E.K.; Barr, E.T.; Le Goues, C.; Brun, Y. Is the Cure Worse Than the Disease? Overfitting in Automated Program Repair. In Proceedings of the 2015 Joint Meeting on Foundations in Software Engineering, Bergamo, Italy, 30 August–4 September 2015; ACM: New York, NY, USA; pp. 532–543.

41. Nakajima, H.; Higo, Y.; Yokoyama, H.; Kusumoto, S. Toward Developer-Like Automated Program Repair—Modification Comparisons between GenProg and Developers. In Proceedings of the 2016 23rd Asia-Pacific Software Engineering Conference, Hamilton, New Zealand, 6–9 December 2016; IEEE: Washington, DC, USA, 2016; pp. 241–248.
42. Waern, Y.; Ramberg, R. People's perception of human and computer advice. *Comput. Hum. Behav.* **1996**, *12*, 17–27. [CrossRef]
43. Jian, J.Y.; Bisantz, A.M.; Drury, C.G. Foundations for an empirically determined scale of trust in automated systems. *Int. J. Cogn. Ergon.* **2000**, *4*, 53–71. [CrossRef]
44. Mayer, R.C.; Davis, J.H. The effect of the performance appraisal system on trust for management: A field quasi-experiment. *J. Appl. Psychol.* **1999**, *84*, 123–136. [CrossRef]
45. Le Goues, C.; Holtschulte, N.; Smith, E.K.; Brun, Y.; Devanbu, P.; Forrest, S.; Weimer, W. The ManyBugs and IntroClass Benchmarks for Automated Repair of C Programs. *IEEE Trans. Softw. Eng.* **2015**, *41*, 1236–1256. [CrossRef]
46. Martinez, M.; Durieux, T.; Sommerand, R.; Xuan, J.; Monperrus, M. Automatic repair of real bugs in java: A large-scale experiment on the defects4j dataset. *Empir. Softw. Eng.* **2018**, *22*, 1936–1964. [CrossRef]
47. LASER-UMASSS/AutomatedRepairApplicabilityData. Available online: https://github.com/LASER-UMASS/AutomatedRepairApplicabilityData/blob/master/ManyBugs.csv (accessed on 27 February 2020).
48. Pinheiro, J.; Bates, D.; DebRoy, S.; Sarkar, D.; R Core Team. Nlme: Linear and Nonlinear Mixed Effects Models. Available online: https://CRAN.R-project.org/package=nlme (accessed on 6 February 2019).
49. R Core Team. *R: A language and Environment for Statistical Computing*; R Foundation for Statistical Computing: Vienna, Austria, 2018.
50. Liu, S.; Rovine, M.J.; Molenaar, P. Selecting a linear mixed model for longitudinal data: Repeated measures analysis of variance, covariance pattern model, and growth curve approaches. *Psychol. Methods* **2012**, *17*, 15–30. [CrossRef]
51. Singmann, H.; Bolker, B.; Westfall, J.; Aust, F.; Ben-Shachar, M.S. afex: Analysis of Factorial Experiments. Available online: https://CRAN.R-project.org/package=afex (accessed on 6 February 2019).
52. Hervé, M. RVAideMemoire: Testing and Plotting Procedures for Biostatistics. Available online: https://CRAN.R-project.org/package=RVAideMemoire (accessed on 6 February 2019).
53. Rusnock, C.F.; Miller, M.E.; Bindewald, J.M. Observations on Trust, Reliance, and Performance Measurement in Human-Automation Team Assessment. In Proceedings of the 2017 Industrial and Systems Engineering Conference, Pittsburgh, PA, USA, 20–23 May 2017; pp. 368–373.
54. Riley, V. Operator Reliance on Automation: Theory and Data. In *Automation and Human Performance: Theory and Applications*; Parasuraman, M., Mouloua, M., Eds.; CRC Press: Boca Raton, FL, USA, 1997; pp. 19–35.