Trust in Collaborative Automation in High Stakes Software Engineering Work: A Case Study at NASA

DAVID GRAY WIDDER, School of Computer Science, Carnegie Mellon University
LAURA DABBISH, School of Computer Science, Carnegie Mellon University
JAMES HERBSLEB, School of Computer Science, Carnegie Mellon University
ALEXANDRA HOLLOWAY, Jet Propulsion Laboratory, California Institute of Technology
SCOTT DAVIDOFF, Jet Propulsion Laboratory, California Institute of Technology

The amount of autonomy in software engineering tools is increasing as developers build increasingly complex systems. We study factors influencing software engineers' trust in an autonomous tool situated in a high stakes workplace, because research in other contexts shows that too much or too little trust in autonomous tools can have negative consequences. We present the results of a ten week ethnographic case study of engineers collaborating with an autonomous tool to write control software at the National Aeronautics and Space Administration to support high stakes missions. We find that trust in an autonomous software engineering tool in this setting was influenced by four main factors: the tool's transparency, usability, its social context, and the organization's associated processes. Our observations lead us to frame trust as a quality the operator places in their collaboration with the automated system, and we outline implications of this framing and other results for researchers studying trust in autonomous systems, designers of software engineering tools, and organizations conducting high stakes work with these tools.

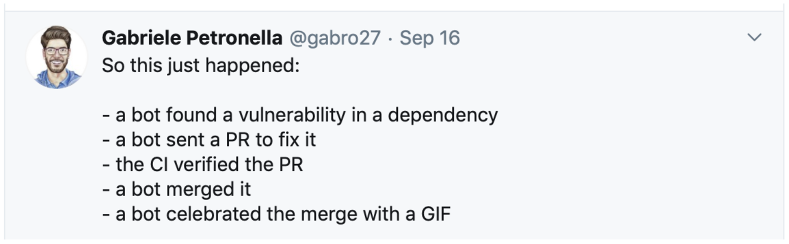
Additional Key Words and Phrases: trust; autonomous; tools; software engineering; ethnography; case study; space

ACM Reference Format:
David Gray Widder, Laura Dabbish, James Herbsleb, Alexandra Holloway, and Scott Davidoff. 2021. Trust in Collaborative Automation in High Stakes Software Engineering Work: A Case Study at NASA. In *CHI '21: ACM Conference on Human-Computer Interaction, May 08–13, 2021, Virtual Conference.* ACM, New York, NY, USA, 22 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Programming is increasingly conducted using tools which automate previously manual actions: such automated tools can help software engineers code [6], write tests [11], build [16], find bugs [26], repair [31], and deploy [33] their systems (also see Figure 1). Programmers necessarily become more reliant on automated tools as they build increasingly complex systems, and in response, designers extend the capabilities of tools, automating previously manual actions. Past work has shown that trust critically influences software engineering tool selection, use, and the decision of how much to rely on such tools [8, 37–39, 41]. This work has not uncovered, however, the factors which influence this trust in increasingly autonomous software engineering tools, nor how trust – a highly contextual and social phenomena – is developed and situated in context.

Research in contexts outside of software engineering finds that an inappropriate amount of trust in automation can lead to negative consequences (*e.g.*, performance reduction as a result of complacency [44]). We use a definition of


Gabriele Petronella @gabro27 · Sep 16
So this just happened:

- a bot found a vulnerability in a dependency
- a bot sent a PR to fix it
- the CI verified the PR
- a bot merged it
- a bot celebrated the merge with a GIF

Fig. 1. A tweet by an open source software engineer describing a fully automated bug fix. (permission obtained)

trust from prior work: "the attitude that an agent will help achieve an individual's goals in a situation characterized by uncertainty and vulnerability" [23, 32]. Many things could influence this attitude, including one's own interaction with the automated system or associated artifacts, or interactions with others about the system. In our context, *too little* trust could lead engineers to, for example, spend valuable time checking output from autonomously generated code which is entirely correct or skip using the tools all together. In contrast, *over*-trust on automated tools may lead to uncaught bugs, costing the loss of a multimillion-dollar mission, the resources spent launching and developing it, and the personal reputations of those who created it. To design tools which invoke neither under-reliance nor over-reliance in their users, it is important to understand the factors which influence this trust.

While much research has studied factors influencing trust in automated systems, we uncover three important gaps in the literature. Firstly, much of this work has been conducted in a laboratory, despite increasing evidence that trust is influenced by contextual organizational and social factors which are best studied *in situ* [23]. Secondly, much of the work on understating factors which influence trust on autonomous systems has been conducted on systems which users cannot modify or inspect, used to achieve a narrowly scoped goal, rather than autonomous tools with which they iteratively collaborate with to solve a creative problem. Finally, this work has not addressed influences of tool trust in the domain of software engineering, where trust should be influenced by the nature of the tools themselves and the ways they are uniquely integrated into key aspects of the software development workflow.

Our paper works to answer the question: **"How does software engineers' trust in an automated tool develop in a high stakes context?"** with a ten-week multi-method ethnographic study of software engineers building an experimental technology demonstration at The Center[1], a site of the National Aeronautics and Space Administration in the United States of America. We find that software engineers' trust in autonomous tools is influenced by four primary influences: Transparency, Usability, Social Context, and Process factors. Our study makes three primary contributions:

- an understanding of how 16 factors, including contextually dependent *social* and *organizational* factors, influence our users' trust in an autonomous tool which they collaborate with continuously to solve a creative problem;
- trust framed as a quality the operator places in their *collaboration with the automated system*, rather than a framing of trust placed in the system in isolation; and
- a study of the deployment of autonomous tools in software engineering, a context which has not been investigated with respect to factors influencing tool trust.

We discuss implications for the following key stakeholders:

[1]Pseudonym given and some details changed to preserve anonymity.

- for Human-Computer Interaction (HCI) researchers, we discuss implications of our framing of trust on future research on trust in similar collaborative autonomous systems, and how our results align with and fill gaps in the prevailing model of trust in autonomous systems;
- for software engineering tool designers, our results can help design tools which are and will be perceived as trustworthy;
- for software engineering organizations, our results can help foster socio-organizational conditions in which trust exchange (*i.e.*, social interactions where people exchange personal trust judgements) can easily occur.

2 RELATED WORK

2.1 Trust in Algorithms and Automation

As algorithms have proliferated into more aspects of life, there has been an increasing volume of research on how they are used, factors that influence trust, and system design that increases trust. Much work on trust in automation been conducted in laboratory or contrived settings, thus unable to situate organizational and social factors in context. In 2015, Hoff and Bashir reviewed 127 studies to update Lee and See's 2004 study of trust in autonomous systems [23, 32], and to build a model of factors influencing trust in autonomous systems. Their model identifies three layers of trust: dispositional trust (*e.g.*, culture, age, personality), situational trust (*e.g.*, riskiness of task, complexity of the system, subject matter expertise), and learned trust (*e.g.*, system performance, preexisting knowledge of system). As the authors themselves note, however, much of the empirical work focuses on a narrow set of systems (*e.g.*, combat systems), and call for a more complete examination of diverse types of automation in real-world environments. Additionally, Scaefer *et al.* performed a meta-analysis of quantitative factors influencing trust in automated systems, and found the need for further study on the role of the human operator and of organizational dynamics such as team training in trust development.

In the field of HCI, there's been interest in user perceptions of automated systems, however, a majority of this work removes or does not consider contextual influences on trust. For example, Rader *et al.* conducted an experiment to investigate the impact that different ways of explaining Facebook's news feed aggregation algorithm to participants had on their judgements of its trustworthiness [45]. They found that explanations helped users understand how it works and how much they could control it, but that explanations did not help users evaluate the algorithm's correctness, consistency, or sensibility. Kizilcec studied the effect of different levels of transparency on students' trust in an algorithmic grading system for an assignment in an online course, finding that transparency increased trust up until a point, after which adding more transparency eroded trust [27]. An experimental study by Binns *et al.* evaluated users' perceptions of justice in algorithmic decision making in different scenarios, and found that the facts of the case mattered more to participants than the explanations they were given [3]. Parasuraman and Miller conducted an experimental study on the effect of an autonomous system's etiquette on pilots and non-pilot's trust in a flight simulator task, finding that good etiquette improved trust [43]. Experimental studies like these can precisely quantify the relationship between participant trust and specific manipulated factors, but certain important factors which may affect trust depend on a complex relationship between users and their environment and their life use cases [23].

A set of recent work in HCI and ML on fairness and accountability of machine learning provides evidence for the strong need and increasing demand for information on algorithmic and autonomous systems in use to better support developers, organizations, and policy makers. Veale *et al.* asked public sector machine learning practitioners about their attitudes on fairness and accountability of machine learning in high stakes settings (*i.e.*, justice, taxation, child protection) and found that current research on the fairness and transparency of these algorithms was removed from the practical organizational and realities of where they would be applied [55]. Holstein *et al.* conducted interviews and surveys to understand practitioners' needs and challenges in developing fair machine learning systems, identifying the need for better understanding of how systems are used in context [24]. This work highlights the need for more ecologically valid studies of trust in autonomous systems, and ethnographic work in complex organizations to uncover factors which may not reveal themselves in contrived situations.

In addition, much of the previous work on attitudes towards algorithmic systems has focused on user affect towards algorithms which they cannot modify or have extended interaction with, rather than algorithmic tools designed to collaborate with users to generate a solution to a goal. Eslami *et al.* evaluated the effect of different explanations of an online advertising targeting algorithm on 32 participants recruited using Craigslist (classified advertisements website), and found that they preferred explanations which were interpretable, non-creepy, and linked specifically to their identity, but were then disillusioned with imperfections [18]. Six of their participants discussed trust, but the authors did not systematically analyze factors which influence trust. Again, the laboratory setting of this study allowed the authors to administer carefully designed tasks, but also poses a threat to ecological validity. Work by a similar set of authors investigated the perceptions that online users have of Yelp's review filtering algorithm, and found that (dis)agreement with this algorithm is determined largely by whether users stand to gain from the algorithm's filtering decisions, and that some users exploit transparency to defeat the filtering algorithm. [19]. We believe more research is needed to evaluate factors which affect trust in these complex, generative, and collaborative autonomous systems.

Finally, there are surveys of factors influencing trust to inform the design of physical robots. For example, Desai *et al.* surveyed literature to inform trusted robot design, and conducted a web questionnaire to evaluate trust in a robotically parked car in different hypothetical scenarios [14]. Hancock *et al.* performed a quantitative meta-analysis of studies measuring trust in human-robot interaction under different experimentally-manipulated conditions, and found that a robot's task performance was the key influence of trust [21]. In contrast to this work, we study a non-embodied system in its real life context.

2.2 Trust in Software Engineering Tools

To our knowledge, there is little work examining factors which lead software engineers to trust tools. There has been work, however, which underscores the importance of trust in determining other software engineering outcomes. Murphy *et al.* found that developers are more likely to use refactoring tools when they trust them, but did not examine factors which may lead to this trust [39]. Niedober *et al.* examined factors which affect pilots' and engineers' trust in an autonomous ground collision avoidance system throughout its development [41], but did not study engineers' trust the tools used to develop the system. Christakis *et al.* found that software engineers find program analysis tools difficult to use when, for example, they have inappropriate default settings, or are not used by the engineer's whole team [8]. The study did not measure trust specifically, but did suggest that cryptic error messages and false positive errors lead to decreased trust in the tool. Devanbu *et al.* found that software engineers tend to form beliefs from personal experience rather than from empirical evidence [15], and that these beliefs can originate from trusted peers. There is also quantitative evidence that projects with members which have previously abandoned a continuous integration tool on a past software project are more likely to do so on a new project, and lack of trust in the continuous integration tool may explain this effect [62]. Finally, a two-page position paper suggested possible factors which might lead to increased trust in software engineering recommender systems [37], based on studies of the adoption and use software engineering recommender systems, [38, 56], but the goal of the underlying studies was to investigate processes affecting their adoption and use rather than factors which lead engineers to trust them.

3 SETTING

3.1 The Center as an Extreme Case

Our study is set at The Center, a site of the National Aeronautics and Space Administration in the United States of America, which builds and operates planetary robotic spacecraft. The Center's history includes successful, multimillion-dollar Earth orbit and deep space planetary exploration missions, positioning it as a High Reliability Organization (HRO): a large organization executing work with a high possibility of, but little occurrence of, dramatic failure [47, 48]. The authors observe three primary qualities of HROs which The Center exhibits [9]. Firstly, a deference to expertise and past experience [60]: we observe that the experiences of "old timers" and their "war stories" (terms used by participants) include views about the right way of doing things, heavily influenced by what has worked in the past, are respected. This deference is also ingrained in less formal ways: hallways and gardens are adorned with chronologies and technical drawings of past missions and commissioned art inspired by them, and walls are hung with spare spacecraft parts within arms reach. Secondly, an inability to rely on trial and error as an organizational learning strategy [60]: space missions are necessarily "one shot", and we observe that consequently, The Center exhibits a conservative ethos to minimize risk, adopting new ideas only after careful study. Finally, a preoccupation with errors when they do occur [7]: for example, we attended an exhaustive root cause analysis presentation for the failure of a planetary rover which had vastly exceeded its designed life. This failure was not viewed as shameful, but this presentation was attended by units across The Center in order to prevent future similar failures. Yin *et al.* explain *extreme cases* as case studies which deviate from theoretical norms or usual occurrences, but whose extremes may reveal insight about normal cases [65]. We position The Center as an extreme case: we expect the extremely high stakes nature of the work, and the associated high reliability organizational culture to make factors affecting trust, and linkages between these factors and the culture, unusually salient and available for study.

3.2 Chief and its Autocoder

This study focuses on factors which affect engineers' trust in *Chief*[2], a control software framework designed to help build software for space missions. Central to Chief is the "Autocoder", a deterministic (*i.e.*, rule-based [22]) autonomous system which generates low level C++ code from a more abstract XML input file created by a software engineer. The generation rules were created and vetted and experienced control software engineers to ensure they do not themselves introduce errors.

Engineers interact with Chief iteratively (see Figure 2): they refine their XML input to iteratively create the Autocoder's output as they implement new components, act on new ideas, or find and fix bugs. The engineers can create the XML for this autonomous system in one of four ways, each with downsides. Most use a diagramming software package called MagicDraw[3] to graphically architect Chief systems (**Old Way**), a tool perceived as sluggish, complex, and error prone. However, after hearing many problems from users about MagicDraw, the Chief's creators built Chief Plus, a modeling language that is in early prototype stages with one test user (**New Way**), but which had not yet seen widespread adoption, leading some to approach it with caution. Others built occasionally error prone tools to generate XML (**Ad Hoc Way**), or wrote input XML by hand (**Tedious Way**) a cumbersome process which some participants report will

[2]Pseudonym given and some details changed to preserve anonymity.
[3]https://www.nomagic.com/products/magicdraw

inevitably lead to errors. Chief is made up of different parts, including a central framework which connects different components, a standard library of core components, and other components contributed by teams external to Chief's maintainers (implications of this discussed in 5.4.3). Chief is "Mission Proven" (see 5.4.2), having been used on at least one previous mission, a status which carries weight. To the pride but also stress of its creators, Chief has been chosen as the control system software package on an upcoming space mission.

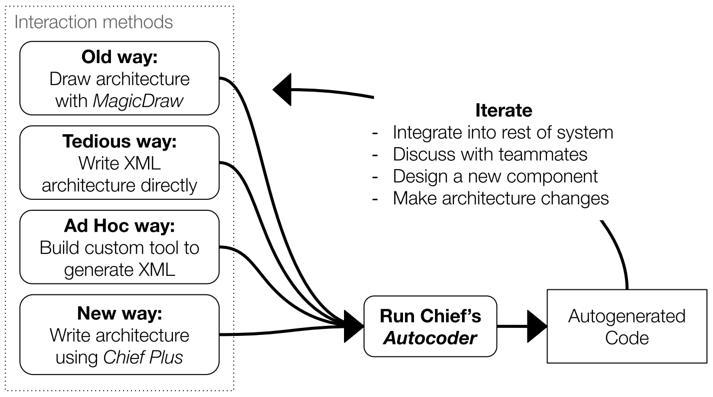

Interaction methods
Old way: Draw architecture with MagicDraw
Tedious way: Write XML architecture directly
Ad Hoc way: Build custom tool to generate XML
New way: Write architecture using Chief Plus
Iterate
- Integrate into rest of system
- Discuss with teammates
- Design a new component
- Make architecture changes
Run Chief's Autocoder
Autogenerated Code

Fig. 2. Steps to use Chief's Autocoder

4 METHODS

To maximize ecological validity to be able to uncover organizational factors, we conduct multi-method *ethnography*, "a qualitative empirical approach suited to understanding people and cultures, and their associated social and work practices" [52]. White *et al.* [61] cites Willis *et al.* [63] to discuss the relationship between ethnography and case study research: "used within an interpretivist framework, 'researchers do not seek to find universals in their case studies. They seek, instead, a full, rich understanding (verstehen) of the context they are studying."' Accordingly, we adopt this interpretivist paradigm (*i.e.*, [34]), believing that our results and the way we interpret them is inextricably linked to the setting in which they were derived, and consequently, that care and further study should be undertaken before attempting to generalize our results to other settings.

Our multiple methods allow us to investigate Trust in Chief in diverse contexts: those with long periods of experience using Chief (Semi-Structured Interviews, Sec 4.2), those new to Chief (Workshop Group Interviews, Sec 4.4), experienced users using a new Chief component for the first time (Think Aloud User Studies, Sec 4.3), and work practices of experienced Chief developers in situ (Participant Observation, Sec 4.5).

4.1 Recruitment

The team which created Chief was a partner in this research and enabled recruitment, providing access and introductions to personnel, documentation, and invitations to meetings about Chief. The team's supervisor provided the first author with a list of people in The Center who currently use Chief or had in the past. We then used snowball sampling to recruit further participants recommended by this first set, thereby building out a network of people who had used, learning to use, develop, manage or advise on the development of, or were otherwise associated with Chief. Participants at The Center (see Figure 1) had a software engineering background and were on a few different teams working on various missions. The first author also built relationships with long time employees of the The Center, who described its culture and practices, some of whom had no association with Chief.

			Years of experience...		
P#	Interview	User Study	on Chief	in the Center	programming
1	✓	✓	3	9	12
2	✓		2	29	34
3	✓		3	3	15
4	✓		3	3	10
5	✓		3	3	15
6	✓	✓	2	2	6
7	✓	✓	3	6	13
8	✓	✓	5	20	40
9	✓		7	48	53
10	✓	✓	3	9	18
11	✓	✓	5	6	37
12	✓		5	33	41
13	✓		1	32	35
14	✓	✓	2	2	35
15	✓	✓	7	28	32
16	✓		2	3	3
17	✓		2	5	6
		Average:	3.4	14.2	23.8

Table 1. Participant Information

4.2 Semi-Structured Interviews

The first author conducted semi-structured interviews on two occasions to probe deeply into factors which mediate engineer's trust in Chief in a one-on-one setting where participants feel comfortable to voice their unfiltered feelings [29]. The first were interviews with as many past or current users of Chief at The Center as possible. Topics included how engineers had learned Chief, how if at all they taught Chief to others, and what barriers they had faced in using Chief. The second set of interviews were conducted after the conclusion of a user study (described in 4.3), after participants had just used Chief Plus for the first time. Seventeen participants were individually interviewed, and these participants had an average of 3.4 years of experience using Chief, 14.2 years working at The Center, and 23.8 years of programming experience (see Figure 1). In both cases the interviews were audio recorded with consent, and interviewees were asked to reflect on aspects of their experience which affected their trust in the parts of Chief they had used, often by analogy or comparison to other software systems they had also used.

4.3 Think Aloud User Studies

To study how participants formed initial (dis)trust judgements in new tools, we designed and conducted think aloud user studies (*i.e.*, [40]) in which participants were presented with Chief Plus (New Way), a new prototype way of interacting with Chief's Autocoder. We constructed three tasks in the Chief *reference application*: an example system built for instructional purposes and as a starting point for real world implementations. Each task required users to replicate existing functionality in the new Chief Plus language, with each one having less scaffolding than the last.

Seven participants were chosen because of their strong affiliation with with Chief Plus's pilot users, and thus likely to adopt it in the future (see Figure 1). Participants were asked to think aloud as they completed these tasks, and with consent, their screens and verbalized thoughts were recorded. One participant declined consent to record so we only took detailed notes. Each participant took about an hour to complete these tasks; Participant 8 did not complete the third task. Users were subsequently interviewed about this first experience using Chief Plus (see above).

4.4 Chief Workshop & Group Interviews

In the beginning of the data collection period, the first author participated in a three-day Chief workshop instructed by Chief's designers to 16 largely student participants. This allowed him to learn how Chief worked to frame future inquiry, and to observe the designer-instructors' values surface through what they emphasised in the training. The workshop also provided opportunities to interact with four student groups working to complete a robot arm project with Chief. The first author performed group interviews with each group at the workshop's conclusion, labeled WSA, WSB, WSC, and WSD. Observations from the students provided a contrast to those from the more experienced engineers employed at The Center.

A majority of the participants attended this workshop because they intended to use Chief in satellite research at their home universities, except four who would use Chief as incoming interns to The Center (group WSD). There were 16 total participants with between three and five per project (and interview) group. Thirteen were undergraduate students, one was a Masters student, one was a PhD student, and one was a staff researcher. Compared to engineers' average of 3.4 years of Chief experience (see Figure 1), workshop participants had an average of four months of experience with Chief, including four participants who had no pre-workshop experience, and ten who had less than six months of experience. Participants had considerably more programming experience: at least one and a half years, a median of five years, and a maximum of 15 years.

At the conclusion of the three day workshop, the first author conducted group interviews with each of the four groups about their experience using Chief [2, 35]. Participants were encouraged to react to and build off of the thoughts of others, with the researcher offering minimal prompting to guide conversation. Interviews lasted between 13 and 30 minutes. Interviews were conducted out of earshot from workshop's instructors so that participants could speak freely about their opinion of Chief and their own misunderstandings or errors.

4.5 Participant Observation

Over ten weeks, the first author immersed himself in the development processes of Chief. He met with many others to chat about their work and experience at The Center and watch them perform their work as a participant observer [53]. For example, the first author observed team meetings to vet new Chief components, discussions about the impending start of project system testing, and concerns over potential threats to launch target. These observations included people managing or advising on the development of Chief, potential users of Chief, people familiar with similar software, but also people unconnected to Chief to gather additional context about the unique nature of The Center as a setting. The first author took field notes and collected any written materials relevant to these interactions, but did not record them in order to avoid making participants feel uncomfortable. At one point, the author even identified and proposed fixes for bugs in the Chief Plus system in collaboration with its lead engineer. These interactions, undertaken throughout the ten weeks, provided context, increased understanding, and therefore confidence in the first author's interpretation of the data collected.

4.6 Data Analysis

Throughout the ten-week data collection period, the authors gathered to discuss data and to memo intermediate emerging themes. At the end of this period, the first author printed out and reread all field notes, collected materials, and transcriptions of interviews and user studies (101 pages total). To ensure he correctly recollected important contextual details such as tone of voice, pace of dialogue, and onscreen actions taken during the user study, he also reviewed the interview and screen recordings (12.5 hours total).

The first author performed three rounds of iterative [58], thematic analysis on this data [5]. In the first round, an initial set of themes was identified from the individual interview data through a process of memoing and any reference to affect towards any program were identified. We wrote up these emerging themes, and presented and discussed them with participants to check resonance, and to the other authors to check comprehensibility. In the second round, references or allusions to things affecting trust including those to non-autonomous programs which participants brought up for the sake of comparison, were tagged with factors influencing trust. Given our definition of trust ("the attitude that an agent will help achieve an individual's goals in a situation characterized by uncertainty and vulnerability" [23, 32]), we were attuned to many different ways of expressing this attitude including direct statements referencing system helpfulness or utility or trust itself, reflections on causes of failure to complete a task, or tone of voice or comments directed at the system while attempting to complete a task. In the final round, these references to trust were systematized into the final 16 codes (see Figure 3), and the full dataset was coded using this scheme.

5 RESULTS: WHAT FACTORS MEDIATE TRUST IN CHIEF?

Our analysis revealed that four major factors influenced system trust. These factors and their components are shown in Figure 3. We find two major organizational trust factors (Social Context, Process Based), and two major categories of factors related to the developer's experience of system itself (Transparency, Usability).

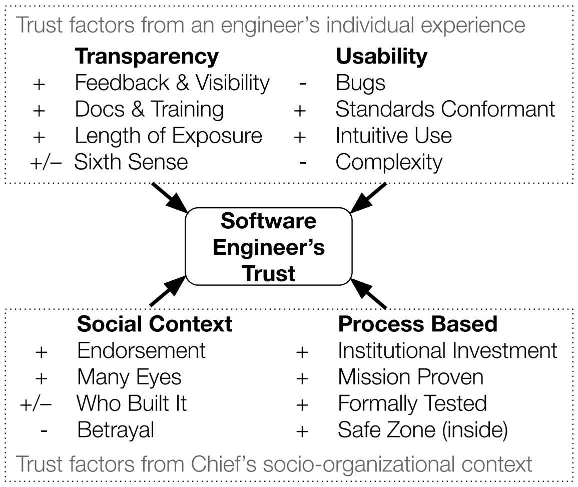


Fig. 3. Factors influencing software engineer's trust in Chief with positive possible influence marked "+", negative influence with "–".

5.1 Transparency

We observed that trust in Chief was influenced by an engineer's ability to understand Chief and its inner workings.

5.1.1 Feedback and Visible Internals. Many participants were not content to trust Chief without first inspecting its internal workings, and satisfying themselves that it worked correctly. For example, one participant stated that Chief made it more work to check that the system he produced with it worked correctly: *"If you develop the system without Chief, it would have been simpler to do. Chief is a large structure, and to [use it] you have to understand how your code works and how Chief works"* (P12). Another participant described how they had to wade through opaque code to get clarity: *"The context in which a port is called in is confusing. I only understood by reading the autogenerated code to understand what is happening"* (P4). Another participant stated that his lack of knowledge of Chief's construction prohibited him understanding possible implications for him using it: these quotes show Chief's autocoder did not exclusively simplify development by automating the previously handwritten code because the engineers were not content to trust this autogenerated code they had no experience with, without manually verifying that it was correct, thus adding an additional (and often time-intensive, as participants note that this autogenerated code is often complex, see also Section 5.2.3) step to build trust in the resulting system. Another participant provided a contrasting example, stating that he trusted open source tools more than Microsoft tools, because the former do not provide "visibility into the source code", underscoring that understanding the internal operation by inspecting source code is crucial to trust in our context. On the other hand, some participants trusted their understanding of the inner workings of Chief's Autocoder enough to modify it, for example, Participant 3 modified Chief's autocoding system. We observe that engineers in our context see an understanding the internal workings of the tools they use, and the output these tools generate, as a precondition to trusting these tools.

We observed that frequent, descriptive feedback from Chief had a positive effect on participant trust because it was commonly how participants expected to probe Chief to learn about its internal workings, and build trust that they had used it correctly. We observed that Chief users wanted to not only trust Chief, but also their ability to use it, in order to trust the code they generated together. We observed that engineers expected feedback from Chief Plus even if only to confirm that no errors occurred, as shown by one engineer attempting to see if the code he had just written was correct by compiling it: *"Usually for things like this I'll ask Google. But you can also ask the compiler, haha! Which I'll do now [compiles their code] Okay what happened? Did it work? No return from the compiler!"* (P11). This was echoed by Participant 14, who further explained that the lack of feedback *during* compilation made it hard to know if progress was being made. This shows that feedback, even when no errors occur, is a crucial way that participants build an understanding of, and learn to trust, Chief's inner workings. Workshop students also wanted feedback to build trust in their ability to correctly use Chief: three of the four workshop groups found Chief's error messages not clear enough to find and fix errors they made, for example: *"[The XML compiler] doesn't even tell you what is exactly wrong. [...] You just change random things and if it works its just works"* (WSB). The perceived poor feedback and requisite "guess and check", "trial and error" methods (quoted from WSB) added additional steps for work workshop students to build their trust in Chief over time, because workshop students observed that "random"(WSB) things fixed these issues, stopping them from learning what had solved the issue and consequently slowing their trust of the result of their iterative collaboration with Chief. In contrast, we observe that the more experienced engineers found the error messages they encountered in the user study to be generally helpful in pinpointing problems, suggesting that the link between feedback and trust is at least in part a function of experience (see 5.1.4).

5.1.2 Documentation & Training. Relevant documentation, and training which left space to have questions answered and provided the rationale behind the instructor's directions, helped participants develop trust in Chief. We observed that in the inspection process explained in Section 5.1.1, participants often had questions, and if they were easily answerable from documentation, we observed that this increased their trust. Reflecting on this process, one participant commented on this: *"I found the Chief User Guide very helpful, but only after struggling for a while. I didn't know how to express the questions I had in Chief language or how to effectively [...] search the documentation"* (P3). Workshop participants said that in absence of instruction, checklist style documentation would be necessary: *"The process of [adding] components [...] was something that wasn't as obvious before [instructors P10 and P15] walked through how you do that. When you don't have that, and when stuff doesn't work, you need a checklist of things to go through"* (WSC). Another group described the build process for Chief projects as difficult without a checklist style "cheat sheet" documentation: *"The process without the Chief cheat sheet thing, I don't know how to explain it except 'convoluted'. Having to go into mod.mk , change that, [...] then change the makefile, then do make, then do ai.something or other, then get a bunch of other files [...] then change something"* (WSA). This quote shows that the presence of documentation can make up for trust lost due to other negative qualities of a system.

Some workshop participants stated that knowing the rationale behind an instructor's direction would help them build trust in how they were told to use Chief: *"If we had gone through that makefile and said like 'so this is how its working' that would help us understand the process of development so much better. Say like 'make gen make, that's doing this', I feel like to some extent when we're doing development we don't know entirely whats going on with one of those commands, which makes it a little bit harder to look for bugs, to know what might have gone wrong"* (WSB). Similarly, a different member of this group stated that being given a bigger picture view of Chief from the outset would help them learn how to connect smaller pieces: *"Something that would have been helpful for me at least was at the beginning if [the instructors explained] how [components] played into the larger demo, but parts of the demo were revealed over time and we'd have to figure out how to connect them to each other. It didn't feel like I saw the big picture until late last night, and that was us on our own"* (WSB). Finally, some workshop participants did not interrupt their instructors to ask questions for fear of upsetting a packed schedule: *"There were way too many technical terms that we didn't understand [the instructors] just breezed over it until you ask about it specifically. A lot of people don't feel comfortable enough to ask to go in depth, especially because we were on a time crunch"* (WSB).

5.1.3 Sixth Sense. Participants appear to acquire a difficult-to-articulate "sixth sense" of which parts of Chief are trustworthy or not, built up over time working on similar problems in similar contexts: *"You kind of need to just know which components are rock solid and which ones you have to test and stuff"* (P10). One specific example was a technical decision about queue depths, the number of messages that a Chief component will support before ignoring further messages, where participants described using their sixth sense to find bugs resulting from a poorly chosen depth: another participant did not like having to rely on their sixth sense, and suggested that some guidance on queue depths could be provided: *"When debugging this, many ports have queue lengths. But what is the optimal queue length? Lots of trial and error. Perhaps there's room for [an official] way to tune queue lengths"* (P4). We note that sixth sense may itself be a consequence of experience (see Sec 5.1.4).

5.1.4 Long Exposure. Participants often cited the length of time they had had to use, inspect, and evaluate Chief as a reason they trusted the system. Less experienced workshop students suggested that they might be able to use Chief more confidently with longer use. Participants often offered the the length of time they had been using Chief as their first justification for why they trusted it: *"[Do you trust Chief?] Yeah, I've been using it for the past 2 years now"* (P1). This quote shows how two participants explained that in using Chief more has increased their confidence using it, one saying: *"[Do you trust Chief?] Yes, I've used it a lot. The amount of experience I've had with the tool has allowed me to be more confident in what it does"* (P7). The tone of voice in which such responses were given was indicative as well. The matter of fact tone participants used to relay the amount of experience they had had with Chief suggested their trust should be obvious as a result of this experience, thereby implying that if there were issues which should affect their trust in Chief, they would have discovered them by now.

Workshop participants felt that more experience would help them debug Chief in spite of its poor feedback (see also 5.1.1): *"This [error message] language is geared towards someone who is basically using it every day. Like, this, the debugging especially, unless you have experience you really don't know what is going on. [...] If you don't have experience with it, there's not really any way you can actually know which random switch you have to make to make it work"* (WSB) This quote illustrates the converse: less experienced workshop students, who have not had experience deciphering error messages, trust their use of Chief less.

5.2 Usability

We also observed than an engineer's ability to easily use Chief was associated with trust in the system.

5.2.1 Standards Conforming. Parts of Chief which conform to externally or internally codified or conventional standards were trusted more, and that participants frequently compared Chief to other systems to point out cases where Chief behaved differently than the expected "standard". One particular example participants mentioned frequently was the way that Chief implements serialization, the method by which a system converts an object into a stream of bytes to store the object or transmit it: *"[What was hard when learning to use Chief?] Probably trying to get my head around [it's] s concept of serialization. When you say serialization I think of JSON and whatnot. That's not what serialization is in Chief"* (P9), showing that Chief's serialization does not align with their expectation based on the common JSON format. Another participant specifically mentioned that the implementation of serialization appears to also be *internally* inconsistent, *i.e.*, implemented differently in different parts of Chief: *"One of the most difficult things about Chief, not for me but for users in general is the Serialize infrastructure. The consistency was switched up, and the Serialization stuff was difficult to grasp"* (P8), whereas another notes that a function in the serialization infrastructure contains a "code smell" (*i.e.*, a pattern in source code of considered to pose maintainability issues [42]) by having a Boolean parameter: *"[Does Chief react in the way you'd expect?] Well I guess there's some aspects of the framework that are a little strange and hard to use. One example that comes to mind is there's some serialization function that has a Boolean argument"* (P11).

Another example of a deviation from standards in the Chief system is the way it handles buffers: *"Buffer manager is a piece of code saying 'give me a buffer', but if you return them out of order it will crash. No one who understands how a buffer manager should work would expect it to behave this way. Once you understand it's a stack based buffer manger, that requires inverse order return. Once you understand this caveat, fine. No one understands this caveat"* (P10). Other participants spoke about how using two different shell versions within Chief is very confusing, and that there should be a consistent standard throughout: *"[Chief Plus's build system] is written in C Shell, which is an abomination. So if C Shell is gonna creep back in, we're gonna have this dual shell mode [another shell is already used in other parts of Chief] which is gonna confuse me. We should have consistency"* (P10)

Other participants desired more clear standards to build trust in their use of Chief, one engineer saying: *"I don't know where the [Chief Plus] format is documented. For instance, XML has a required format"* (P14). Similarly, a workshop student spoke about an "unclear line" between making a component "active" or "passive", but which was presented by instructors as a choice with clearly defined determinants: *"I had to ask [about] the reasons it was supposed to be active and passive. The servos should be active because the command dispatcher was connected to it. But from the maps that we were understanding, it could have been passive. Having that line be unclear kind of confused me in some aspects, because it* *can be passive in what you're doing, but because this could also happen, and because this is also happening, you should make it active"* (WSB). In summary, participants trusted parts of Chief which behaved in accordance with codified or conventional standards more.

5.2.2 Bugs. While systems which have many bugs may obviously be trusted less than those with fewer, we observed an important nuance: participants hated it when bugs surprise them, so they hate "unknown bugs" much more than "known bugs". Reflecting on the new Chief Plus prototype, one participant describes unknown bugs as "bear traps": *"Before I use [Chief Plus], I want to see it become more mature. The rumors are there are a lot of bear traps to tiptoe around, similar to as in MagicDraw"* (P15), and another made reference to Donald Rumsfeld's "unknown unknowns": *"I do worry that that the core Chief components have not been unit tested to a large degree. Some unknown unknowns"* (P14). Another said he trusts Chief because of the absence of unknown bugs: *"I trust the Chief Autocoder. I've been using it for 3 years, and in my experience I rarely discovered a bug or issue"* (P7). In the user study, one participant discovered an unknown bug while reading Chief Plus's source code: *"Oh look! There's a comment here: 'This line fails for no apparent reason.' What?! [laughs exasperatedly]"* (P10). Bugs still exist in Chief, but participants seemed reassured when they knew where the bugs were: *"Yes, I trust Chief because [...] I have a good understanding of its deficiencies"* (P8). Another trusted Chief, despite the presence of bugs: *"[Do you trust Chief?] I mean, I think mostly. Although [...] there's still bugs in it"* (P11). In summary: the presence or suspected presence of unknown bugs has a negative impact on trust, but the presence of known bugs need not.

5.2.3 Complexity. Highly complex Chief systems were trusted less. Three participants criticised the complexity of MagicDraw, one saying: *"It seemed complicated to connect components to the topology using MagicDraw. I don't think I could have hooked that up myself, if that was a requirement. Seemed to have lots of human procedural steps you have to know to hook up things in MagicDraw"* (P5). Another participant complained that frequently used functionality in MagicDraw is buried seven menus deep. Another participant described how he'd be more likely to trust a small system which does fewer things well rather than an ambitious but complex system: *"The development of tools like Microsoft Word is focused on adding more and more features, and making it more and more cumbersome and hard to use, and not making sure that the basic features work. I want the basics to be simple and fast and reliable and easy to use. I don't want multiple conflicting ways to do the same thing"* (P11).

In the Chief context, another participant invited a simplification of the code which results from the Autocoder: *"To keep things simple and fast is important. The autocoded code was way longer than it [should be] in my experience as a realtime software developer, and not the most efficient. The complexity of the stuff that is autocoded [is bad], you write functions that get magically called from autocoded things"* (P12).

Complexity resulting from frequent context switches caused workshop participants to be concerned they may make a mistake, affecting their trust in their own ability to use Chief: *"The number of steps you have to go through to make even like the simplest component, is pretty extreme. You have to touch six different files, and each one is in its own directory almost. [...] That's just a lot of tabs we have to keep on switching through"* (WSB). Another group agreed, suggesting this may cause mistakes: *"Having a lot of moving parts, that adds to the potential of having bugs during that setup process. It makes it more likely that when you're doing all these different operations that you'll make a mistake somewhere, or have a directory that's incorrect or something, or forget a piece"* (WSC), whereas workshop group B explained how they similarly made mistakes this way: *"if you end up in the wrong directory, you mess something up. And have to redo a lot of things"* (WSB).

5.2.4 Intuitive Use. Chief systems which could be used intuitively were more trusted. User study participants appeared to have a sense of "the way it should work" when completing the tasks using Chief Plus, which they were using for the first time: *"Despite my lack of knowledge about Chief Plus, for me it was intuitive"* (P14). Users explained that they trusted systems which react in a way that aligns with their expectations, which they derive from their past experience: *"If you understand what you're doing, GCC will do what you expect. If you tell it to do something, it'll do what you say, I rarely worry about it doing something unexpectedly"* (P10). Other participants described "rituals", undocumented capabilities that departed from their mental models to achieve the desired functionality out of a system: *"The hardest part was teaching [interns] MagicDraw, like the procedure for when you change the component interface, you have to rebuild it in this way, do a clean, and rebuild everything. If you want to compile with the kernel and such. The most difficult part was the rituals you had to do with MagicDraw"* (P16), whereas others pointed out the amount of repetition required to do simple tasks: *"[There was] lots of repetition. Lots of generic ports that usually go to the same place. It took a long time to add maybe 12-15 components. The ports that I really cared about didn't take very long to set up, but adding all the commanding and logging and telemetry ports took forever. It's slow to do things in MagicDraw. There's just a lot of repetitive motion"* (P4). One workshop group said how buffer delays are set doesn't align with "tradition", only discovering how after consulting a manual: *"I would traditionally just use a buffer to build a message and just send it. My first instinct would be to define one in the function, then send it. But you're supposed to define a single buffer as a member and do various construction things to it. [...] I don't know how I would have learned that that was there, other than reading a reference manual"* (WSA).

5.3 Social Context

We observed trust in Chief was also dependent on other people's interaction with Chief, including team members, other users, or downstream users of an engineer's code.

5.3.1 Many Eyes. Systems and parts of Chief which had more users, and thus "more eyes" checking them, were trusted more. Participants presumed that having many users meant that many eyes have checked the code or system, and report or help fix those bugs, shown in: *"I trust the bigger well established packages that lots of people use more. You have millions of users acting as testers. The more eyes the more I'd trust it to be solid."* (P15), and *"When a lot of people are banging on it, they [bugs] get discovered and fixed"* (P11). They also believed the reverse to be true: *"I get the opinion that [software tool] doesn't have enough eyes on it to be trusted. [However, another tool] works because it has a thousand eyes on it"* (P10) This was often spoken about in a way similar to "social proof", as shown by: *"If other people have used it and not run into big problems, then I'd trust it more"* (P6), *"[I trust this tool because of] how widely its used"* (P1), and *"I trust the GCC compiler. Its had thousands and thousands of users successfully using it"* (P8), and they also mentioned open source software specifically as a common case of the "many eyes" indicator of trust, as shown in: *"Its open source. So you have multiple people who know compilers looking at the code. So if there's bugs, they'll usually put a patch in quickly"* (P14) and *"Having [this tool] on GitHub and having the community be active and report their issues, is the reason why I trust it"* (P7).

5.3.2 Who Built It? The person or organization that built the code or system in question was an important factor in participants' perceptions of its trustworthiness. These perceptions often reflected participants' past experience with systems built by that author, or their personal experience interacting with the individual or team who built it.

Chief, because it was built at The Center by some of our own participants, was often trusted because of the participant's close relationship with those developing it: *"[Do you trust Chief?] I guess I do, because I work with very smart people who work on it"* (P14), and another had even helped develop Chief themselves: *"[Do you trust Chief?] I also wrote a lot of it, my own input into it, I know how its structured, and I have experience writing [control] code and I* *know we have good developers writing it"* (P15). However, not everyone felt this way, with one participant distrusting Chief's Autocoder specifically because it was built at The Center: *"I don't trust our in house tools. I don't trust the Chief Autocoder"* (P8)

In contrast, our participants especially distrusted tools built by Microsoft: *"In general, I am less trusting of Microsoft tools"* (P14), and: *"[What tool don't you trust as much?] Like, uh, any Microsoft tool. Like, Word"* (P11). Parts of Chief are developed through a collaboration with a university as a final project for Masters of Software Engineering students, a fact mentioned with skepticism (?) by one of our participants: *"Chief Plus is developed by [colleague's name] and a sea of Masters students at [university]"* (P10). While participants spoke about the benefit of open source software having "many eyes" on it in Section 5.3.1, another participant spoke about a potential downside of not knowing the authors personally: *"Its open source, sometimes I think would someone put something bad in it? But, in general, yes I trust GCC [a compiler]"* (P14). In summary, participants used their knowledge of software's authorship to inform their trust judgements about it.

5.3.3 Betrayal. Systems which introduced errors in an easy to miss manner, likely to be caught by others (rather than the original engineer) were trusted less. Because of the high stakes, one shot, nature of the work The Center performs, the tools and systems used often had the potential to damage our participants reputation, thereby betraying their trust. For example, incorrectly interacting with Chief's Autocoder using MagicDraw could result in errors which were easy to miss, and only likely to be caught by someone else, but easily traceable to the original developer. MagicDraw appeared to modify files when it shouldn't, in a way that violated our participant's trust that it would not modify files autonomously without user input: *"I open up my MagicDraw doc, but I don't save it, yet GitHub reports the file has changed. And so in the background, MagicDraw is saving stuff without telling you"* (P7).

The Chief Plus Autocoder does not have a "spellchecker", thus propagated typos an engineer makes to places in the system which might not be caught by that engineer, thus putting the engineer's reputation at risk: *"Wait, I'm confused... Oh god! So if I miss type something, the compiler just converts this directly into my XML! I miss typed the word and it just jammed it into XML with no errors! It should check! Someone needs to modify the compiler, so that it can catch misspellings. Because the error will pop up in code you didn't write!"* (P10). Similarly, another participant spoke about how MagicDraw propagates engineer's errors in a way that will only be caught late in the development process, thus exposing their errors to others and putting their reputation at risk: *"If you do something wrong in MagicDraw's GUI, it generates a topology or a module XML which is also slightly wrong. That wouldn't be found until we actually try to run the software on the testbed for testing"* (P16).

5.3.4 Endorsement. Tools and systems which had been used or verbally endorsed by engineer's team members were trusted more, possibly due to social proof [10]: *"[Do you trust Chief?] Yes, I know other people are using it. Talking about it with my group, everyone's using it, the community itself really helps"* (P7). Participant 7 explained that when he was new to Chief, his peers helped him learn to use Chief, but also to trust Chief. This is an instance of "social proof", a psychological phenomenon where people use the behavior of others a cue for the appropriate way to behave in a situation [10]. Another participant provided an example of a negative endorsement of the prototype Chief Plus's "bear traps" conveyed through both formal meetings and less formal rumors: *"The rumors are that there are a lot of bear traps [in Chief Plus] to tiptoe around. [...] Sitting in meetings, I got the impression it was fragile, and that there were issues that had to be sorted out"* (P15). We note that social Endorsement, or lack thereof, was also used to communicate other factors that can also be discovered individually such as Bugs (see Sec 5.2.2).

5.4 Process Based

Trust was also given to a system through a formal trust-granting process or routine that was part of The Center's style of work. We refer to this as process based trust.

5.4.1 Formally Tested. Systems that were tested more thoroughly were more trusted. Participants reported knowing some parts of Chief needed further testing before they could be fully trusted: *"[Do you trust Chief?] I mean, I think mostly? Although parts of it could be better tested. [...] But large parts of it are thoroughly tested. There are just some classes of bugs that just sit there until testing happens"* (P11). Another participant reported that insufficient testing is completely disqualifying from a trust perspective: *"I don't trust Chief's Autocoder. It hasn't been tested as much as a real tool. We tend to discount testing and such here"* (P8).

5.4.2 Mission Proven. Systems which had served successfully on a space mission (dubbed "Mission Proven") were also more trusted. This Mission Proven status carried revered importance for both hardware and software components and systems in The Center. Participants informally kept track of what components and systems were Mission Proven. They reported trusting Chief more because Chief's auto-generated code had been Mission Proven on a previous satellite. This status is so important that it is one of the first characteristics mentioned in materials promoting Chief.

When asked why participants trust Chief, participants often gave the fact it was Mission Proven in a tone of voice that suggested it was the only justification for their trust that was needed: *"Yes, we've successfully [used] Chief on several spacecraft"* (P10) as one example, and: *"[Do you trust Chief?] Yes, I guess I do. Its flying on [satellite name]"* (P14) as another. Another participant described this Mission Proven status in a genealogical sense: *"[Do you trust Chief?] Mostly yes ... because it has heritage, we've used it successfully [in previous missions]"* (P11), and other employees of The Center would talk about processes and tools which are still used today after successful use on many past missions.

5.4.3 Safe Zone. When components of systems were admitted into a conventionally understood "safe zone", they were trusted much more than those which are not. Some components are considered generic and well tested enough to be considered "core" and maintained by Chief's creators themselves and are generally trusted more, others are contributed by other teams who have used Chief in the past and trusted less. Participants reported trusting components inside the safe zone, and viewing other components as suspect: *"So Chief is two things: its a standard library and a framework. The framework is rock solid, its been tried and testing. The standard library I look at with skepticism except for core components. Some of the other components, I don't trust to be reliable [control] software"* (P10). Another participant on the team responsible for Chief's development explained how his team has absorbed components from other teams into this safe zone: *"We've inherited components in other projects, and reused them, and they have heritage"* (P7). Another member of Chief's development team mentioned that components inside the safe zone are more rigorously tested: *"[Do you trust Chief?] I mean, I think mostly. Although parts of it could be better tested, but large parts of it are thoroughly tested"* (P11).

5.4.4 Institutional Investment. The fact that The Center chose to invest in Chief was a signal to engineers that it could be trusted. The Center made investment decisions with a great deal of care and study, because missions were often expensive and risky. Participants saw funds invested in Chief as a form of official anointment and strong indicator of trust. *"Do you trust Chief? Yeah, It's trusted to run a 40 million dollar experimental [mission]"* (P15). Participants recognized both the dollar figure of this trust ("40 million"), and the fact that this trust is potentially risky ("experimental"). However, another participant, said of the Chief Plus system: *"Here's the problem at [The Center], we don't have much to invest in* *these kind of technologies"* (P8), perhaps showing that he doesn't think the prototype Chief Plus system has yet been invested in enough by The Center to merit his trust just yet.

6 DISCUSSION AND IMPLICATIONS

Our study has possible implications for further research on trust in collaborative and creative autonomous systems, for software engineering tool designers seeking to build tools their users trust, and for organizations wanting to enable their members to appropriately calibrate their trust in the tools they use.

6.1 Implications for Research on Trust in Automated Tools: Trusting the *Collaboration*, (not the System)

We observed engineers use Chief iteratively, re-running Chief, tuning their input to iteratively refine output as they attempt to implement new ideas, change goals, or fix bugs. It is in this context that we believe Chief is best viewed as a collaborator, and consequently, that trust is best thought of as something the operator places in the ***human-automated system collaboration, which includes themselves***, rather than trust being something an operator places in Chief. We believe this conception of trust fits our context: the software for one space mission at The Center can take a decade to develop, launch, and maintain. We heard of decade long missions having software mishaps repaired using some of the same tools they were built with, so it is within this context that we believe engineers instead assess their trust in their ability to *work with* Chief, regardless of how goals may develop in the future, rather than their trust *in Chief* to successfully complete a particular, preordained goal. We believe that this conception of trust also fits our data: for example, our participants often offer explanations relating to their own ability to work with Chief as primary factors influencing their trust, such as the length of time they've been working with Chief, the extent to which the way they want to use Chief aligns with their expectations and their ability to understand Chief and its output. We believe this framing of collaborative trust can be used to consider more influences on trust in future studies of creative, operator-autonomous system collaborations as they become increasingly powerful and common as autonomous systems proliferate knowledge work contexts. For example, Siedel *et al.*'s recent line of work investigates how game designers used autonomous tools to help design a virtual world, and how such tools enabled designers to create more complex games, emphasizing benefits of repeated iteration and creativity these tools can enable [49–51]. As autonomous systems support more open ended goals, trust will increasingly depend on collaboration between the operator and the system, which necessarily must be studied in context. Our proposed framing contrasts with the papers reviewed in Hoff and Bashir, which position the human operator as having a narrowly scoped and known goal for their use of automation, with clearly defined criteria for when and whether this goal has been achieved. For example, combat identification aids, which help weapons operators identify enemy targets, were the most commonly studied system used in their review, are systems with a clearly defined goal, and results can clearly be decided as either success or failure.

Separately, our study corroborates many experimental results used in Hoff and Bashir's model of trust human operators place in automated systems, and our ethnographic approach allows us to situate these factors within a real life socio-organizational context, a need they recognize [23]. These include the length of exposure operators have to the system [66], the system's reputation [36], nature and quality of feedback provided by the system [57], and the transparency of the system [17]. Contradictorily, some work (*i.e.*, [20, 46]) finds that trust in automation decreases as risk increases, but other work suggests the reverse (*i.e.*, [54]). Hoff and Bashir state that future work is needed, but propose "under high risk conditions, operators may have a tendency to reduce their reliance on *complex* automation", and in our high risk context we indeed find that additional complexity, both in the way one must interact with Chief and the output it generates, reduces trust.

6.2 Designing Trust-Supporting Organizations

Our ethnographic method revealed how a tool's social context can influence trust. Here we discuss how future work could examine whether and how our results transfer to other organizations support the transfer of trust judgements about autonomous tools which can be trusted and those which cannot.

Our finding that endorsement (or lack thereof) influences trust in Chief suggests that mechanisms to make known who uses what tool, and what their experience was, is a way to help a tool's newcomers calibrate their trust, noting that past work has already noted that the interchange of such experiences already affect tool adoption [64]. Relatedly, our finding that the reputation of who built a tool can influence trust suggests that organizations developing tools for internal use may clearly identify who built what tool, and what other tools those people made. We also observed that Chief's creators view their personal reputation is tied to that of Chief, so making tool authorship visible may further incentivize the creation of good tools.

Our finding that The Center's institutional investment in Chief positively impacts trust suggests that organizations may make known to their members the approval process for adopting new tools or software. For example, we observed that The Center subjects new external tools to rigorously security vetting before allowing their use internally, and exposing the criteria use for similar vetting in other organizations may help increase members trust in the vetted tool.

Our Mission Proven result is particularly unique to context, but other High Reliability Organizations can look for similar routines and signals that trust was granted to a tool or system when it had been used in previous missions without incident, and make these apparent. However, given that HROs often defer to what has worked in the past, this may lead to the limitation that new tools can't get adopted or they have to come up with other ways to 'prove' they will work on a mission. We find that parts of Chief admitted to a "safe zone" were trusted more than experimental components which were not. This suggests a way in which organizations can split parts of software they develop or tools they use into those which are highly vetted, and those which are less, as a way to preserve ability to experiment with new things while retaining trust and high reliability.

6.3 Designing Trusted Software Engineering Tools

To our knowledge, this is the first study on factors which influence trust in software engineering tools. Since trust has been shown to impact software engineering tool adoption [38, 56], further study can examine whether and how our results may transfer to other contexts to help software engineering tool designers evaluate and improve their tool's trustworthiness and adoption. Below, we discuss how providing visibility into their inner workings, improving training and documentation to include the *why* not just the *how*, publicly exposing known bugs, and highlighting when past experience may be out of date increased users' trust in Chief.

Our study suggests that tools with easily understood internals, for example, by open sourcing and documenting the code, may be trusted more. However, companies may be reluctant to release source code of proprietary algorithms but may risk being seen as opaque and ethically questionable as a result [30], so future work can examine ways to build trust by exposing as much of the internal workings of proprietary autonomous software tools without revealing their source code. Additionally, our results suggest that aligning with existing codified standards or expected norms may help trust when such constraints prevent full transparency.

Our study suggests that training and documentation on *how* to use tools are not enough: to trust the tool, software engineers also expect to understand *why* by including the not just the rationale for what they are told to do, but also why certain design decisions were made, a result, suggesting that not only should automated systems provide explanations for their behavior to incur trust [17], but that their human creators must too. We note that other research underscores the important of supporting "why" questions in debugging [28], and that the structure of frameworks can support debugging [12]. Crucially, our results also suggest that training and documentation, when done right, can make up for trust lost elsewhere, such as through overcomplexity.

Perhaps counter intuitively, our research suggests that the presence of *Bugs* in software engineering tools need not always have a negative impact on trust: the surprise discovery of "unknown bugs" negatively impact users' trust, whereas a knowledge of Chief's deficiencies helped build trust. This aligns with research on interpersonal trust which shows that surprise violation of expectations by teammates adversely affected trust [1]. As a result, software engineering tool companies may consider ways to expose known bugs to their users, such as making bug tracking dashboards public, in order to build trust. We note that this is already common in open source software, many of which have public lists of known issues [4]. Studies on the trustworthiness of open source software in comparison to proprietary software find that reliability and functionality are the two most frequently cited factors affecting trustworthiness [13], and there have even been attempts to build tools to measure the trustworthiness of open source software components [25]. This concurs with our observations that participants often trust open source software, which have more users to find bugs thus improving perceived reliability (see 5.3.1) and provide visibility into the source code (see 5.1.1).

Software tools designed for other software engineers are often less than usable [59], and our study underscores the importance of making using such tools intuitive through its impact on trust. For example, our participants performed "rituals" (see 5.2.4) to achieve desired functionality, and others found cases where they had to rely on a "Sixth Sense" to use Chief correctly. Identifying, documenting, and ultimately fixing these issues can improve tool trust.

Our research shows that trust tended to increase as users became more experienced. However, research on other frequently changing frameworks found past experience can lead users to misdiagnose errors due to previous experience, with one user stating past experience should not be trusted [12]. Thus, to ensure the positive trust outcomes as users become more experienced that we observe, tool designers should clearly mark when changes between tool versions require different use.

7 CONCLUSION

We conducted a ten-week multi-method ethnographic study of the factors which influence users' trust as they collaborate with an autonomous tool in a high stakes context. We report how transparency, usability, social context, and process based trust factors are situated in context.

We discuss how our observations lead us to frame trust as a quality the operator places in their collaboration with the automated system, and we outline implications of this framing and other results for researchers studying trust in autonomous systems, software engineering tool designers, and organizations conducting high stakes work with these tools.

8 ACKNOWLEDGEMENTS

We thank our participants and the leaders of Chief's development for enabling this study and sharing their experiences in the spirit of improving Chief. We thank our reviewers from successive submissions, our shepherd, Julie and John Widder, Pranav Khadpe, and Anastasia Sosnovskikh for their helpful feedback to improve our paper. The research was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration (80NM0018D0004).

REFERENCES

[1] Ban Al-Ani, Erik Trainer, David Redmiles, and Erik Simmons. 2012. Trust and surprise in distributed teams: towards an understanding of expectations and adaptations. In *Proceedings of the 4th international conference on Intercultural Collaboration*. 97–106.

[2] Rosaline Barbour. 2008. *Doing focus groups*. Sage.

[3] Reuben Binns, Max Van Kleek, Michael Veale, Ulrik Lyngs, Jun Zhao, and Nigel Shadbolt. 2018. 'It's Reducing a Human Being to a Percentage': Perceptions of Justice in Algorithmic Decisions. In *Proc. Conf. Human Factors in Computing Systems (CHI)*. ACM, 377.

[4] Tegawendé F Bissyandé, David Lo, Lingxiao Jiang, Laurent Réveillere, Jacques Klein, and Yves Le Traon. 2013. Got issues? who cares about it? a large scale investigation of issue trackers from github. In *2013 IEEE 24th international symposium on software reliability engineering (ISSRE)*. IEEE, 188–197.

[5] Virginia Braun and Victoria Clarke. 2012. Thematic analysis. (2012).

[6] Frank J. Budinsky, Marilyn A. Finnie, John M. Vlissides, and Patsy S. Yu. 1996. Automatic code generation from design patterns. *IBM systems Journal* 35, 2 (1996), 151–171.

[7] James G Casler. 2014. Revisiting NASA as a high reliability organization. *Public Organization Review* 14, 2 (2014), 229–244.

[8] Maria Christakis and Christian Bird. 2016. What developers want and need from program analysis: an empirical study. In *Proc. International Conf. on Automated Software Engineering (ASE)*. IEEE, 332–343.

[9] Marlys K Christianson, Kathleen M Sutcliffe, Melissa A Miller, and Theodore J Iwashyna. 2011. Becoming a high reliability organization. *Critical care* 15, 6 (2011), 314.

[10] Robert B Cialdini. 1993. Influence: The psychology of persuasion. (1993).

[11] David M Cohen, Siddhartha R Dalal, Jesse Parelius, and Gardner C Patton. 1996. The combinatorial design approach to automatic test generation. *IEEE software* 13, 5 (1996), 83–88.

[12] Zack Coker, David Gray Widder, Claire Le Goues, Christopher Bogart, and Joshua Sunshine. 2019. A qualitative study on framework debugging. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 568–579.

[13] Vieri Del Bianco, Luigi Lavazza, Sandro Morasca, and Davide Taibi. 2011. A survey on open source software trustworthiness. *IEEE software* 28, 5 (2011), 67–75.

[14] Munjal Desai, Kristen Stubbs, Aaron Steinfeld, and Holly Yanco. 2009. Creating trustworthy robots: Lessons and inspirations from automated systems. (2009).

[15] Premkumar Devanbu, Thomas Zimmermann, and Christian Bird. 2016. Belief & evidence in empirical software engineering. In *Proc. International Conf. on Software Engineering (ICSE)*. IEEE, 108–119.

[16] Paul M Duvall, Steve Matyas, and Andrew Glover. 2007. *Continuous integration: improving software quality and reducing risk*. Pearson Education.

[17] Mary T Dzindolet, Scott A Peterson, Regina A Pomranky, Linda G Pierce, and Hall P Beck. 2003. The role of trust in automation reliance. *Human-Computer Studies* 58, 6 (2003), 697–718.

[18] Motahhare Eslami, Sneha R Krishna Kumaran, Christian Sandvig, and Karrie Karahalios. 2018. Communicating algorithmic process in online behavioral advertising. In *Proc. Conf. Human Factors in Computing Systems (CHI)*. ACM, 432.

[19] Motahhare Eslami, Kristen Vaccaro, Min Kyung Lee, Amit Elazari Bar On, Eric Gilbert, and Karrie Karahalios. 2019. User Attitudes towards Algorithmic Opacity and Transparency in Online Reviewing Platforms. In *Proc. Conf. Human Factors in Computing Systems (CHI)*. ACM, 494.

[20] Neta Ezer, Arthur D Fisk, and Wendy A Rogers. 2008. Age-related differences in reliance behavior attributable to costs within a human-decision aid system. *Human Factors* 50, 6 (2008), 853–863.

[21] Peter A Hancock, Deborah R Billings, Kristin E Schaefer, Jessie YC Chen, Ewart J De Visser, and Raja Parasuraman. 2011. A meta-analysis of factors affecting trust in human-robot interaction. *Human factors* 53, 5 (2011), 517–527.

[22] Frederick Hayes-Roth. 1985. Rule-based systems. *Commun. ACM* 28, 9 (1985), 921–932.

[23] Kevin Anthony Hoff and Masooda Bashir. 2015. Trust in automation: Integrating empirical evidence on factors that influence trust. *Human Factors* 57, 3 (2015), 407–434.

[24] Kenneth Holstein, Jennifer Wortman Vaughan, Hal Daumé III, Miro Dudik, and Hanna Wallach. 2019. Improving fairness in machine learning systems: What do industry practitioners need?. In *Proc. Conf. on Human Factors in Computing Systems (CHI)*. ACM, 600.

[25] Anne Immonen and Marko Palviainen. 2007. Trustworthiness evaluation and testing of open source components. In *Seventh International Conference on Quality Software (QSIC 2007)*. IEEE, 316–321.

[26] Brittany Johnson, Yoonki Song, Emerson Murphy-Hill, and Robert Bowdidge. 2013. Why don't software developers use static analysis tools to find bugs?. In *Proc. International Conf. on Software Engineering (ICSE)*. IEEE Press, 672–681.

[27] René F Kizilcec. 2016. How much information?: Effects of transparency on trust in an algorithmic interface. In *Proc. Conf. Human Factors in Computing Systems (CHI)*. ACM, 2390–2395.

[28] Andrew J Ko and Brad A Myers. 2004. Designing the whyline: a debugging interface for asking questions about program behavior. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. 151–158.

[29] Ugur Kuter and Cemal Yilmaz. 2001. Survey methods: Questionnaires and interviews. *Choosing Human-Computer Interaction (HCI) Appropriate Research Methods* (2001).

[30] Kyriakos Kyriakou, Pınar Barlas, Styliani Kleanthous, and Jahna Otterbacher. 2019. Fairness in proprietary image tagging algorithms: A cross-platform audit on people images. In *Proceedings of the International AAAI Conference on Web and Social Media*, Vol. 13. 313–322.

[31] Claire Le Goues, ThanhVu Nguyen, Stephanie Forrest, and Westley Weimer. 2011. Genprog: A generic method for automatic software repair. *IEEE Transactions on Software Engineering* 38, 1 (2011), 54–72.

[32] John D Lee and Katrina A See. 2004. Trust in automation: Designing for appropriate reliance. *Human Factors* 46, 1 (2004), 50–80.

[33] Marko Leppänen, Simo Mäkinen, Max Pagels, Veli-Pekka Eloranta, Juha Itkonen, Mika V Mäntylä, and Tomi Männistö. 2015. The highways and country roads to continuous deployment. *Ieee software* 32, 2 (2015), 64–72.

[34] Yvonna S Lincoln, Susan A Lynham, and Egon G Guba. 2011. Paradigmatic controversies, contradictions, and emerging confluences, revisited. *The Sage handbook of qualitative research* 4 (2011), 97–128.

[35] Howard Lune and Bruce L Berg. 2017. *Qualitative research methods for the social sciences.* Pearson.

[36] Poornima Madhavan and Douglas A Wiegmann. 2007. Effects of information source, pedigree, and reliability on operator interaction with decision support systems. *Human Factors* 49, 5 (2007), 773–785.

[37] Gail C Murphy and Emerson Murphy-Hill. 2010. What is trust in a recommender for software development?. In *Proc. Workshop on Recommendation Systems for Software Engineering*. ACM, 57–58.

[38] Emerson Murphy-Hill, Gail C Murphy, Joanna McGrenere, et al. 2015. How Do Users Discover New Tools in Software Development and Beyond? *Computer Supported Cooperative Work (CSCW)* 24, 5 (2015), 389–422.

[39] Emerson Murphy-Hill, Chris Parnin, and Andrew P Black. 2011. How we refactor, and how we know it. *Transactions on Software Engineering* 38, 1 (2011), 5–18.

[40] Brad A Myers, Amy J Ko, Thomas D LaToza, and YoungSeok Yoon. 2016. Programmers are users too: Human-centered methods for improving programming tools. *Computer* 49, 7 (2016), 44–52.

[41] David J Niedober, Nhut T Ho, Gina Masequesmay, Kolina Koltai, Mark Skoog, Artemio Cacanindin, Walter Johnson, and Joseph B Lyons. 2014. Influence of cultural, organizational and automation factors on human-automation trust: A case study of Auto-GCAS engineers and developmental history. In *International Conf. on Human-Computer Interaction*. Springer, 473–484.

[42] Steffen Olbrich, Daniela S Cruzes, Victor Basili, and Nico Zazworka. 2009. The evolution and impact of code smells: A case study of two open source systems. In *2009 3rd international symposium on empirical software engineering and measurement*. IEEE, 390–400.

[43] Raja Parasuraman and Christopher A Miller. 2004. Trust and etiquette in high-criticality automated systems. *Commun. ACM* 47, 4 (2004), 51–55.

[44] Raja Parasuraman, Robert Molloy, and Indramani L Singh. 1993. Performance consequences of automation-induced'complacency'. *The International Journal of Aviation Psychology* 3, 1 (1993), 1–23.

[45] Emilee Rader, Kelley Cotter, and Janghee Cho. 2018. Explanations as mechanisms for supporting algorithmic transparency. In *Proc. Conf. Human Factors in Computing Systems (CHI)*. ACM, 103.

[46] Bako Rajaonah, Nicolas Tricot, Françoise Anceaux, and Patrick Millot. 2008. The role of intervening variables in driver–ACC cooperation. *Human-Computer Studies* 66, 3 (2008), 185–197.

[47] Karlene H Roberts and Gina Gargano. 1989. *Managing a high reliability organization: A case for interdependence.* Managing complexity in high technology industries: Systems and people. New ....

[48] Gene I Rochlin, Todd R La Porte, and Karlene H Roberts. 1987. The self-designing high-reliability organization: Aircraft carrier flight operations at sea. *Naval War College Review* 40, 4 (1987), 76–92.

[49] Stefan Seidel, Nicholas Berente, and John Gibbs. 2019. Designing with Autonomous Tools: Video Games, Procedural Generation, and Creativity. (2019).

[50] Stefan Seidel, Nicholas Berente, Aron Lindberg, Kalle Lyytinen, and Jeffrey V Nickerson. 2018. Autonomous tools and design: a triple-loop approach to human-machine learning. *Commun. ACM* 62, 1 (2018), 50–57.

[51] Stefan Seidel, Nicholas Berente, Benoit Martinez, Aron Lindberg, Kalle Lyytinen, and Jeffrey V Nickerson. 2018. Autonomous tools in system design: Reflective practice in Ubisofts Ghost Recon Wildlands project. *Computer* 51, 10 (2018), 16–23.

[52] Helen Sharp, Yvonne Dittrich, and Cleidson RB De Souza. 2016. The role of ethnographic studies in empirical software engineering. *IEEE Transactions on Software Engineering* 42, 8 (2016), 786–804.

[53] James P Spradley. 2016. *Participant observation.* Waveland Press.

[54] Charlene K Stokes, Joseph B Lyons, Kenneth Littlejohn, Joseph Natarian, Ellen Case, and Nicholas Speranza. 2010. Accounting for the human in cyberspace: Effects of mood on trust in automation. In *Proc. Symposium on Collaborative Technologies and Systems*. IEEE, 180–187.

[55] Michael Veale, Max Van Kleek, and Reuben Binns. 2018. Fairness and accountability design needs for algorithmic support in high-stakes public sector decision-making. In *Proc. Conf. Human Factors in Computing Systems (CHI)*. ACM, 440.

[56] Petcharat Viriyakattiyaporn. 2009. *An active help system to improve program navigation.* Ph.D. Dissertation. University of British Columbia.

[57] Lu Wang, Greg A Jamieson, and Justin G Hollands. 2011. The effects of design features on users' trust in and reliance on a combat identification system. In *Proc. Human Factors and Ergonomics Society Annual Meeting*, Vol. 55. SAGE Publications Sage CA: Los Angeles, CA, 375–379.

[58] Robert Philip Weber. 1990. *Basic content analysis.* Number 49. Sage.

[59] Thomas Weber, Alois Zoitl, and Heinrich Hußmann. 2019. Usability of Development Tools: A CASE-Study. In *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. IEEE, 228–235.

[60] Karl E Weick. 1987. Organizational culture as a source of high reliability. *California management review* 29, 2 (1987), 112–127.

[61] Julie White, Sarah Drew, and Trevor Hay. 2009. Ethnography versus case study. *Qualitative Research Journal* 9, 1 (2009), 18–27.

[62] David Gray Widder, Michael Hilton, Christian Kästner, and Bogdan Vasilescu. 2019. A conceptual replication of continuous integration pain points in the context of Travis CI. In *Proc. European Software Engineering Conf./Foundations of Software Engineering (ESEC/FSE)*. ACM, 647–658.

[63] Jerry W Willis, Muktha Jost, and Rema Nilakanta. 2007. *Foundations of qualitative research: Interpretive and critical approaches.* Sage.

[64] Shundan Xiao, Jim Witschey, and Emerson Murphy-Hill. 2014. Social influences on secure development tool adoption: why security tools spread. In *Proceedings of the 17th ACM conference on Computer supported cooperative work & social computing*. 1095–1106.

[65] Robert K Yin. 2017. *Case study research and applications: Design and methods.* Sage publications.

[66] Nirit Yuviler-Gavish and Daniel Gopher. 2011. Effect of descriptive information and experience on automation reliance. *Human Factors* 53, 3 (2011), 230–244.