# The Relationship Between User-Developer Communication and Software Bias

A Research Paper submitted to the Department of Engineering and Society

Presented to the Faculty of the School of Engineering and Applied Science
University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree
Bachelor of Science, School of Engineering

Anna Williamson
Spring, 2023

On my honor as a University student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-related assignments.

Signed: _____ Date: _____
Anna Williamson

Approved: _____ Date: _____
S. Travis Elliott, Department of Engineering and Society

## Introduction

Recently the public has become more aware of the presence of bias in software systems and the form this bias takes. For example, the development of the AI chatbot called chatGPT has sparked public concern over technology biased around political views (CBS News, 2023). This topic has implications for all software developers and poses an important question: how are biases introduced into software? This STS report specifically focuses on biases introduced through user-developer communication. The goal of this report is to answer the question: how does user-developer communication foster bias in software? The presence of bias in artificial intelligence related to facial identification is often discussed in modern media and this bias also permeates other, less reported, software systems. Friedman et al. (1996) examine some of the more subtle biases in graphical user interfaces (GUIs), educational software, and other less cutting-edge software systems (p. 49). Software is ingrained in society and inherent biases in software work to reinforce and perpetuate societal inequities. Creating a bias-free and fair society necessitates developing software systems that are bias-free. Additionally, biases in the software development process can lead to the development of a product that does not meet the needs of the target user group and does not fulfill its intended purpose. In order to create useful software that satisfies the target user group and that prevents the perpetuation of societal biases, it is important to understand how exactly bias influences the development of software systems and how developers can guard against this unwanted influence. This paper first provides some background by defining bias, and defining requirements engineering and why it's important for software development. Next, the software product CONFIG is introduced, which was a software system that ultimately failed due to poor user-developer communication and a flawed requirements engineering process. The Social Construction of Technology (SCOT) is then

suggested as a method for analyzing the combined effects of users and developers on software during the software development cycle and how these groups are able to shape software products. SCOT is used to discuss the development of two "virtual cities" in the Netherlands and explain how biased software can strengthen societal biases. Finally, an argument is made to conduct more research into the intersection of bias and requirements engineering with the goal of creating useful, effective, and bias-free software.

**Background**

When creating a piece of software, developers communicate with a group of potential users in order to understand what users might want from the software. This process of communicating with the user-group is called requirements elicitation. Requirements elicitation involves conducting interviews, sending out questionnaires, or using other methods to communicate with a user-group in order for developers to determine what a software system should do. However, flawed communication between users and developers during requirements elicitation can lead to biased software systems.

Communication between users and developers about software can be complex. There are many obstacles preventing complete and useful communication between developers and users, including user and developer bias, lack of communication channels, indirect communication that leads to misunderstood feedback, differing levels of system understanding between users and developers, and various social factors that prevent honest feedback. All of these obstacles prevent user satisfaction with the final software product. In the industry of software development, understanding the desired product is critical since maintenance and future changes can comprise 90% of the total software cost (Dehaghani & Hajrahimi, 2013, p. 63). Misinterpreting or assuming prior knowledge of user needs can lead to the development of a

product that is not used by the target group and ultimately fails. Interpreting user feedback incorrectly or collecting user feedback in a biased manner can allow bias to enter the software. Friedman et al. (1996) present a very useful definition for understanding bias in software systems by stating that a software is biased "if it *systematically* and *unfairly discriminates* against certain individuals or groups" and "assigns an undesirable outcome… on grounds that are unreasonable or inappropriate" (p. 48). This definition is concerned with the impact of a software product on its users and whether or not the effective outcome of the software is the same for different types of users. An unbiased software should be able to fully meet the needs of its entire target user group and therefore provide a desirable outcome to all of its users when suitable. However, bias is often built into software systems during the software development cycle. In their paper discussing the impact of gender on software development and requirements engineering, Nunes, Moreira, and Araujo (2023) write that "software does not equally serve everyone" and that software favors "characteristics… observed in those that are represented during development" (p. 1). This quote suggests that software itself is a biased tool and that this bias reflects the biases present during the development process. Individuals or groups who are consulted effectively during requirements elicitation will have their needs met by the software product and so the software will be biased towards these individuals or groups. Since software is so ingrained in society, subtle biases in everyday software can significantly harm certain groups and perpetuate inequity.

The idea of including users in the software development process is not new. Developers understand that users play an important role in the development cycle and should be consulted. However, not much research has been dedicated to understanding how developers and users communicate and exactly what impact users can have on the final software product. Gallivan and

Keil (2003) analyze the system CONFIG, created by an anonymous company in 1980, and reveal that more user participation in the development process does not necessarily lead to a better product (p. 45). Developers of CONFIG worked to include users throughout the process by holding periodic meetings with a subset of the user group, integrating a feedback system into the software, and frequently conducting telephone surveys to gauge opinion on the new system (Gallivan & Keil, 2003, p. 46). However, they still found that users interacted very little with the finished system. Users were pressured by the company to accept the product and therefore moderated their feedback. They also assumed that developers were already aware of "obvious" problems with the software. The use of indirect feedback, i.e., collected through a telephone surveyor before reaching developers, served to warp original feedback collected from the user. Finally, developers were affected by overconfidence bias and did not consider that the system was not actually useful for users. CONFIG was abandoned in 1992 after an expensive redeployment initiative that did not improve its usage numbers (Gallivan & Keil, 2003, p. 45). This is an extreme example to motivate continued research into the communication pipeline between users and developers, focusing on how user needs are interpreted and applied to the developing software. Due to the developers' bias for developing the software and due to ineffective communication between users and developers, a software was created that wasn't useful or needed by the end users.

In contrast to the CONFIG example, which was a financially expensive and notable failure in the software world, it is often difficult to identify subtle biases in more conventional or every-day software systems. For example, Friedman et al. (1996) point out the mainly male characters in computer games, the GUI designs that have become increasingly difficult for the visually-impaired to navigate, and the assumption of prior knowledge in understanding software

systems (p. 49). Biases held by developers and users can influence the requirements elicitation process and effect the end software system.

**Introduction and Analysis – SCOT**

Social construction of technology (SCOT) can be used to understand the role users and developers play in introducing bias to a software system. User-developer communication in software development can be examined through the lens of Pinch and Bijker's Social Construction of Technology (SCOT) (1984), with an emphasis on the simultaneous influence of both users and developers on the software system. Social Construction of Technology is a technological lens that emphasizes the ability of various interest groups to influence the development of a technology. SCOT argues that the ways in which societal groups interpret, interact, and use a technology define that technology and its function. Society determines how a technology is perceived, potentially adopted, and changed. This theory is useful for understanding the role of user-developer communication relating to bias because it takes into account the personal experiences that shape the developer and user groups. Each developer and user will work to mold the technology, either consciously or unconsciously, to benefit themselves. Thus, the development of the software will be influenced by these two interest groups.

There are many interest groups involved with a particular software, however this paper specifically examines the user and developer groups. This relationship is shown in Figure 1, where the financial sponsor group denotes a general entity that commissions the development of the software product. The financial sponsor has an initial strong influence on the developers and the resulting software product as they commission the software product and define its function. However, the focus of this paper is on the influence of the users and developers. SCOT can be

used to examine how users and developers shape technology, especially through the requirements elicitation process.

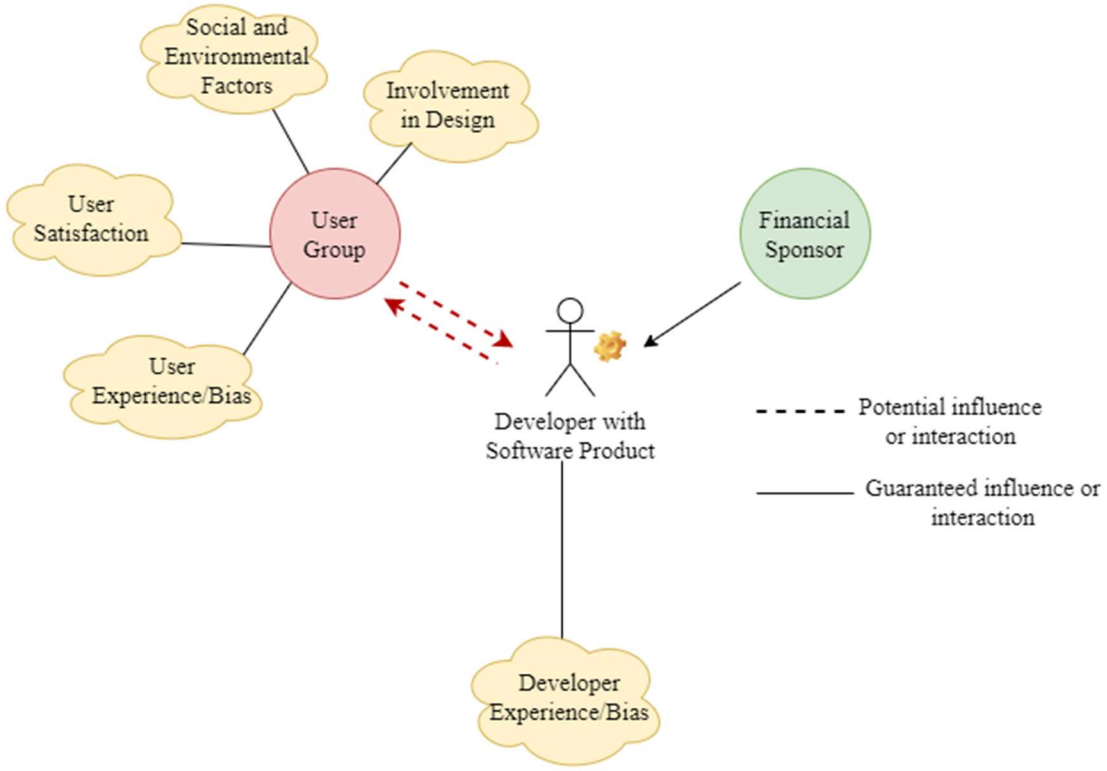


Figure 1: Adapted Social Construction of Technology (SCOT) model for the Influence and Interaction of the User Group in the Development of a Software Product: A depiction of the user group and financial sponsor influencing the final software product. There is a focus on the user-developer communication pipeline, which is not a guaranteed structure in the design system but connects a subset of the final user group with the developers. (Adapted by Anna Williamson from W. Carlson, 2009).

An excellent example of how users shape technology can be seen through the design of "electronic virtual cities" (2004, p. 33). Oudshoorn et al. examine the role of the user in the design process and the introduction of bias into these virtual cities. The Digital City of Amsterdam (DDS) in the Netherlands was created to encourage political discussion by allowing citizens to access discussion forums and political information from a computer. New Topia was another virtual city project that allowed citizens to navigate a pixelated "town" through a

television and read news, play games, and post messages. Both of these projects were originally pitched as technology for "everyone." However, as the projects progressed, it became clear that the developers were unaware of the true needs of their user group. In the DDS project, designers were motivated to develop innovative technology and therefore added new features to the software that made it difficult for computer-illiterates to navigate and ultimately decreased the benefit of the software. The developers' personal ideas of what the software should do influenced the decision to make the software more complex and add additional features. This change satisfied the developer interest group, but caused problems for the software later on. Ineffective testing with computer-illiterate users ensured that this bias wasn't caught during development. The developers did conduct user testing however the test-group was recruited from a technology conference, which guaranteed that all testers had experience using computers. The software was satisfactory for this particular user interest group and their prior experience working with computers shaped the final software product. Since most individuals with prior computer experience at the time were men, this led DDS to become a masculine technology, with only 18% of its users in 1996 being female (Oudshoorn et al., 2004, p. 44). This demonstrates the importance of understanding how user-developer communication shapes technology. The prior experiences of the test group ensured that they had the necessary skills to understand the software and so did not bring any issues to the attention of the developers. The biases of the developers when choosing a test group then influenced the end software product – DDS was built to be more easily used by people with prior computer experience.

The New Topia project encountered a similar issue where developers believed that "entertainment and socializing" (Oudshoorn et al., 2004, p. 50) were motivators for using the new technology. User testing was conducted on company employees, thus ensuring that all

testers had some experience using computers and were motivated to interact with the system. This user interest group was satisfied with the software product and their role as testers allowed them to directly influence the development of the software. After the software was released, it became clear that it was used primarily by young men. Women disliked the system because it resembled "computer games" (Oudshoorn et al., 2004, p. 50) and were more interested in a system that could provide local news and community information. In both of these projects, the designers did not have a strong grasp of their target user group and any potential barriers to different groups using the software. They also conducted limited user-testing that was used to validate the complete system instead of to determine user motivations, desires, and interests. By choosing small user groups for testing that did not represent the full target user group of "everyone," the software was molded to satisfy only these narrow user groups. Oudshoorn et al. also suggest that these projects relied on the I-methodology, which describes a developer's tendency to rely on their own needs and interests instead of consulting the target user-group. This demonstrates the developers' subjective role in developing software – a developer's own biases and interests are built into software. By introducing the developers' own needs and interests into the software product, the software becomes biased for the developer and may become biased against certain user groups.

These two examples of developing "virtual cities" represent common mistakes in the world of software development. Developer bias, user bias, and a lack of clear communication between users and developers during the requirements elicitation stage of software development can have negative effects on a software product. As seen in these examples, the end software can become biased against certain user groups and not meet their needs, thus the software only partially fulfills its intended purpose. Users' and developers' experiences shaped the technology

in a way that is modeled through the Social Construction of Technology theory. By incorporating users' and developers' personal needs into the software, the software is influenced by individuals in society. As in the examples above, it is important for developers to be aware of which users exert an influence on the technology. Since developers completed testing with only a small subset of the target user group, these testers exerted much more influence over the final software product.

Developers should be deliberate and aware of which users they collect feedback from, how many users from various social groups offer feedback, how situational and social pressures affect user feedback, how they interpret and apply user feedback, and how both developer and user biases can carry into the final software product. For example, Gallivan and Keil (2003) state that "…minority group members… are often reluctant to share their views publicly…" (p. 63), which affects what user needs are incorporated into the final software product. If the personal needs of minority users aren't incorporated into software, then the software will be biased towards other, more vocal groups. Software developers may be unaware of the biases they promote in the development of a software product. They may also be unaware of the potential for bias during user interaction. For example, developers who only survey or receive feedback from a certain group of users will be biased to create a system based on the needs of this specific group instead of for the larger target group. While more user influence in the development process is generally positive, it is essential for developers to examine how this influence should be exerted.

**Discussion**

Identifying and measuring bias can be difficult and there is a significant lack of research into how user-developer communication throughout the requirements elicitation process

influences bias in software. As shown in Figure 2, the amount of published research in this area has only recently increased. Research in this field is growing, however the link between requirements engineering and bias in software systems is still not well understood or analyzed on a large scale. Developers don't have a concrete guide for requirements elicitation that takes into account unintentional biases. More research should be conducted in order to understand the best methods for user-developer communication that minimize bias. Developers should be aware of potential biases in themselves and the user group in order to combat such biases in the software. Improved user-developer communication and representative user groups are also important to creating unbiased software systems.

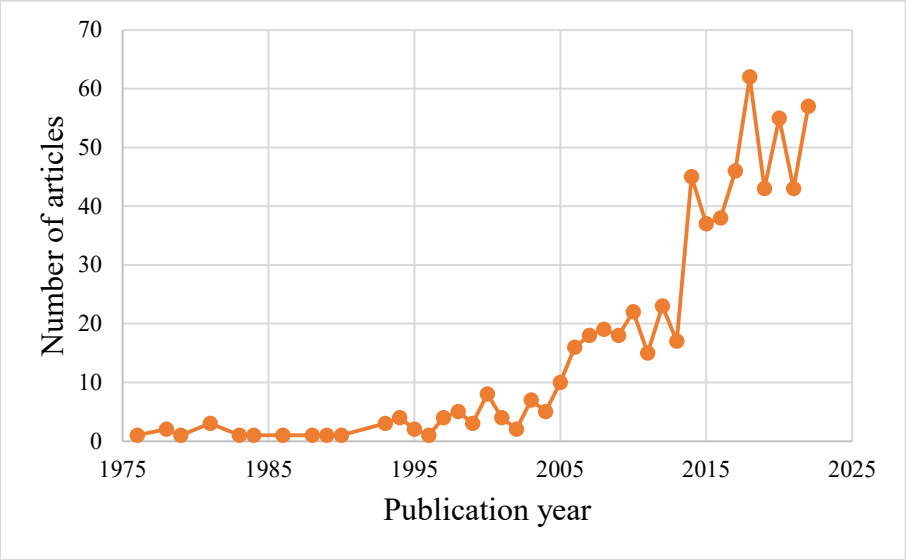


Figure 2: Published Articles in the Association for Computing Machinery (ACM) Digital Library Related to Requirements Engineering and Bias: Filtering articles in the ACM Digital Library by "requirements engineering" AND "bias" yields evidence of the amount of yearly research in this field (Created by Williamson, 2022).

As society continues to integrate software systems into daily life, developers and users should both be aware of potential biases in these systems and how they can affect different user groups. Bias can be introduced into a software product throughout its development, including during user-developer communication. It is important to examine how user-developer communication occurs so that bias can be recognized and prevented and developers can create

fair, unbiased software products. Developing software systems that are unbiased is important for society since software has a far-reaching impact on individuals. The bias for male users in DDS worked to reinforce past inequity surrounding computer literacy and furthered stereotypes about gender and technical aptitude. In the same way, modern software influences the lives of individuals in subtle ways that reinforce societal biases. In order to create an unbiased society, software systems must be unbiased. Since user-developer communication can introduce biases into a software, developers should be aware of this potential and work to combat it.

The anticipated outcome of this paper is both to encourage more research into this area as well as bring the issue of subtle software biases to the attention of developers. By making developers aware of the possibility of introducing bias into their software, developers can guard against biased requirements elicitation practices and work to communicate with a representative user group. Other practices such as collecting direct vs indirect feedback and ensuring accessible communication channels between developers and users are also important, but not well understood. More research into the best methods for user-developer communication is necessary to understand how various practices contribute to or minimize bias.

**Conclusion**

Requirements elicitation, or the process of developers communicating with users, is an important step in software development that can introduce bias into the software. User bias, developer bias, and flawed communication between users and developers are all ways in which biases can be built into a software. For example, the CONFIG project involved flawed user-developer communication that ultimately resulted in a software product that wasn't useful for the target group. Developers were biased towards building the software and users were affected by various social pressures impacting the feedback they provided so that the software project was

ultimately a failure. The Social Construction of Technology theory is useful for understanding the role of developers and users in creating and influencing software. These two interest groups are able to mold the software in order to satisfy their needs. Collecting feedback from an unrepresentative user group and applying the developer's personal interests to a software results in a biased software system that may not be beneficial to all the users in the target user group. The examples involving "virtual cities" DDS and New Topia demonstrate the dangers of conducting unrepresentative user testing and using I-methodology to develop software. In order to guard against developing biased software, developers should be aware of the possibility for user-developer communication to introduce bias into software. More research into exact methods for conducting effective user-developer communication in order to prevent bias is necessary. However, simply being aware of any personal biases a developer might have and communicating clearly with a representative user group is helpful for minimizing bias in software development.

## REFERENCES


Aydemir, F. B., & Dalpiaz, F. (2018). A roadmap for ethics-aware software engineering. *Proceedings of the International Workshop on Software Fairness, 40*, 15-21. https://doi.org/10.1145/3194770.3194778

CBS News. (2023). *CHATGPT and large language model bias | 60 Minutes*. CBS News. Retrieved March 14, 2023, from https://www.cbsnews.com/video/chatgpt-and-large-language-model-bias-60-minutes/

Dehaghani, S., & Hajrahimi, N. (2013). Which factors affect software projects maintenance cost more?. *Acta Informatica Medica, 21*(1), 63-66. https://doi.org/10.5455/aim.2012.21.63-66

Friedman, B., Brok, E., Roth, S. K., & Thomas, J. (1996). Minimizing bias in computer systems. *SIGCHI Bulletin, 28*(1), 48-51. https://doi.org/10.1145/249170.249184

Friedman, B., & Nissenbaum, H. (1996). Bias in computer systems. *ACM Transactions on Information Systems, 14*(3), 330-347. https://doi.org/10.1145/230538.230561

Gallivan, M. J., & Keil, M. (2002). The user-developer communication process: A critical case study. *Information Systems Journal, 13*(1), 37-68. https://doi.org/10.1046/j.1365-2575.2003.00138.x

Jantunen, S., Dumdum, R., & Gause, D. (2019, May 25-31). *Towards new requirements engineering competencies*. International Conference on Software Engineering, Montreal, QC, Canada.

Johanssen, J. O., Kleebaum, A., Bruegge, B., & Paech, B. (2019). How do practitioners capture and utilize user feedback during continuous software engineering?. *IEEE International*

*Requirements Engineering Conference, 27*, 153-164. https://doi.org/10.1109/RE.2019.00026

Mohanani, R. (2016, May 14-22). *Implications of requirements engineering on software design: A cognitive insight*. International Conference on Software Engineering, New York, NY, United States.

Nunes, I., Moreira, A., & Araujo, J. (2023). GIRE: Gender-inclusive requirements engineering. *Data & Knowledge Engineering, 143*, 1-19. https://doi.org/10.1016/j.datak.2022.102108

Oudshoorn, N., Rommes, E., & Stienstra, M. (2004). Configuring the user as everybody: Gender and design cultures in information and communication technologies. *Science, Technology, & Human Values*, 29(1), 30–63. https://doi.org/10.1177/0162243903259190

Pagano, D., & Bruegge, B. (2013). User involvement in software evolution practice: A case study. *International Conference on Software Engineering, 35*, 953-962. https://doi.org/10.1109/ICSE.2013.6606645

Pinch, T., Bijker, W. (1984). The social construction of facts and artefacts: Or how the sociology of science and the sociology of technology might benefit each other. *Social Studies of Science, 14*(3), 399-441. https://doi.org/10.1177/030631284014003004

Saiedian, H., & Dale, R. (2000). Requirements engineering: Making the connection between the software developer and customer. *Information and Software Technology, 42*(6), 419-428. https://doi.org/10.1016/S0950-5849(99)00101-9

Williamson, A. (2022). *Adapted Social Construction of Technology (SCOT) model for the influence and interaction of the user group in the development of a software product.* [Figure 1]. *Prospectus* (Unpublished undergraduate thesis). School of Engineering and Applied Science, University of Virginia. Charlottesville, VA.

Williamson, A. (2022). *Published articles in the Association for Computing Machinery (ACM) Digital Library related to requirements engineering and bias.* [Figure 2]. *Prospectus* (Unpublished undergraduate thesis). School of Engineering and Applied Science, University of Virginia. Charlottesville, VA.

Zalewski, A., Borowa, K., & Kowalski, D. (2020). On cognitive biases in requirements elicitation. In S. Jarzabek, A. Poniszewska & L. Madeyski (Eds.), *Integrating research and practice in software engineering* (pp. 111-123). Springer Cham.