Humans in the Loop: People at the Heart of Systems Development

Helen Sharp

Abstract Despite increased automation in the process, people are (still) at the heart of software systems development. This chapter adopts a sociotechnical perspective and explores three areas that characterize the role of humans in software systems development: people as creators, people as users, and people in partnership with systems. Software is *created* by specialist developers such as software engineers and non-specialists such as "makers." Software developers build communities and operate within several cultures (e.g., professional, company, and national), all of which affect both the development process and the resulting product. Software is *used* by people. Users also operate within communities and cultures which influence product use, and how systems are used feeds back into future systems development. People and systems are interdependent: they work in *partnership* to achieve a wide range of goals. However, software both supports what people want to do and shapes what can be done.

1 Introduction

Digital humanism aims to put humans at the center of the digital world, arguing that technology is for people and not the other way around. Other chapters in this volume (e.g., Winter in this volume) advocate human-centered systems development which suggests that humans' needs should be the driving force for development and that humans and groups should be better integrated into the system development cycle.

This chapter echoes that perspective but turns the spotlight back onto the people who contribute to the development of digital artifacts and how the human tendency to form communities, and their respective cultures, influences and shapes the artifacts they produce. We focus more specifically on the role that people have in the development and use of systems: who are they, what is their role, and how do

H. Sharp (✉)
Faculty of Science, Technology, Engineering and Mathematics, The Open University, Milton Keynes, UK
e-mail: helen.sharp@open.ac.uk

H. Werthner et al. (eds.), *Introduction to Digital Humanism*,
https://doi.org/10.1007/978-3-031-45304-5_23

humans shape the digital artifacts that they encounter. A key feature of this work is that systems development is seen as sociotechnical, i.e., an approach that makes explicit the fact that people and technology are interdependent (Klein, 2014), a perspective that is increasingly pertinent to digital humanism.

Digital artifacts rely on software, and software is fundamental to virtually everything people do nowadays. Apart from phone apps that keep people in touch with their loved ones, allow bills to be paid, and keep track of their fitness levels, there are also global software-based projects, from instrumentation of the James Webb telescope out in space (e.g. see NASA, 2023) to modeling the spread of viruses across the world (e.g. Wang et al., 2021), controlling neighborhood traffic (e.g. see WDM, 2023), and tracking animals in danger of extinction (e.g. Kulits et al., 2021). Software development is at the core of digital artifact design and implementation, controlling its behavior, how it interacts with users and the environment, determining how trustworthy or secure it is, and whether it supports what the human user is trying to do.

This chapter will focus on software and software development and aims to consider who is involved, what is their role, and how does the sociotechnical nature of software systems development affect the software produced. It is divided into three sections, exploring people as creators of software systems; people as users of software systems; and the partnership between people and software systems.

2 People as Creators of Software Systems

When thinking about people as creators of software systems, the first group that comes to mind are the professionals—specialist software designers and builders. But there is also a growing set of people who are not specialists yet who are involved directly in developing and implementing software systems. Whether specialists or non-specialists, people who create software systems are not acting on their own. Instead, they sit within a community of designers, developers, users, and other stakeholders who contribute to creation in one way or another. These communities may be professional (e.g., user experience designers), organizational (e.g., employees of a company), or personal (e.g., based on ethnicity), but they all influence systems development. For instance, in a study of designers in Botswana, researchers found that sociocultural factors of the designers influenced both the design process and the designed artifact (Lotz & Sharp, 2020; Sharp et al., 2020).

This section explores these two groups (specialists and non-specialists) and how they influence and inform systems development.

2.1 Specialists such as Software Engineers

Professional software engineers are one subset of specialist creators, although this seemingly homogeneous group is made up of yet more subgroups, such as commercial software developers, open-source developers, and freelance software developers, for example. Members of these groups work within a community and a network of stakeholders, technical components, and standards. Their work is influenced by different cultures and their own experience and those around them (Sharp et al., 2000).

Modern software development is a very complex endeavor and relies quite extensively on building from existing components such as language library assets, interface components, patterns, and design system languages, often created by different groups. A piece of software must be embedded in its technical environment and is dependent on digital and physical attributes of the device and of the environment within which it operates, e.g., Internet connectivity and access to digital assets. This complexity means that software developers and their work are highly dependent on others: local others and distant others.

The community aspect of software development is often overlooked. Software developers may operate in teams, which is one kind of community, but they also form very close communities across companies, disciplines, and continents (see Fig. 1). These may coalesce around programming languages or tools, or in specific domains such as finance or physics, or in particular locations. Members of these communities support each other with solutions to problems, guidance on technical matters, and documentation, for example. And it goes beyond that—communities are very influential. When we were looking at object-oriented development in the 1990s, one of the research questions we had was how did object-oriented technology emerge and become widespread. We didn't look at the official history but instead tried to follow strands of evidence in contemporaneous literature. From that investigation came the view that the community of object-oriented advocates built a significant following through community events so that when a commercial-strength object-oriented language emerged (C++), there was a ready-made appetite among developers for it to spread very rapidly (Robinson & Sharp, 2009).

Developer communities support each other in various practical ways through sharing solutions and propagating information. But the impact that social processes have goes beyond this. For instance, resilience of sociotechnical systems relies on people (Furniss et al., 2011a). While some aspects of resilient performance are visible through written procedures or policies, others are "hidden" within adaptations made by people every day. To illustrate this, Furniss et al. (2011b) provide several examples from a hospital study. One of these relates to a batch of infusion pumps that were prone to triggering a false alarm. A workaround, i.e., lubricating the relevant part with alcohol gel, was developed by the nurses but was not captured in any procedures nor reported to anyone beyond the immediate team. Instead, people adapted their behavior to account for the situation until the batch had been used and

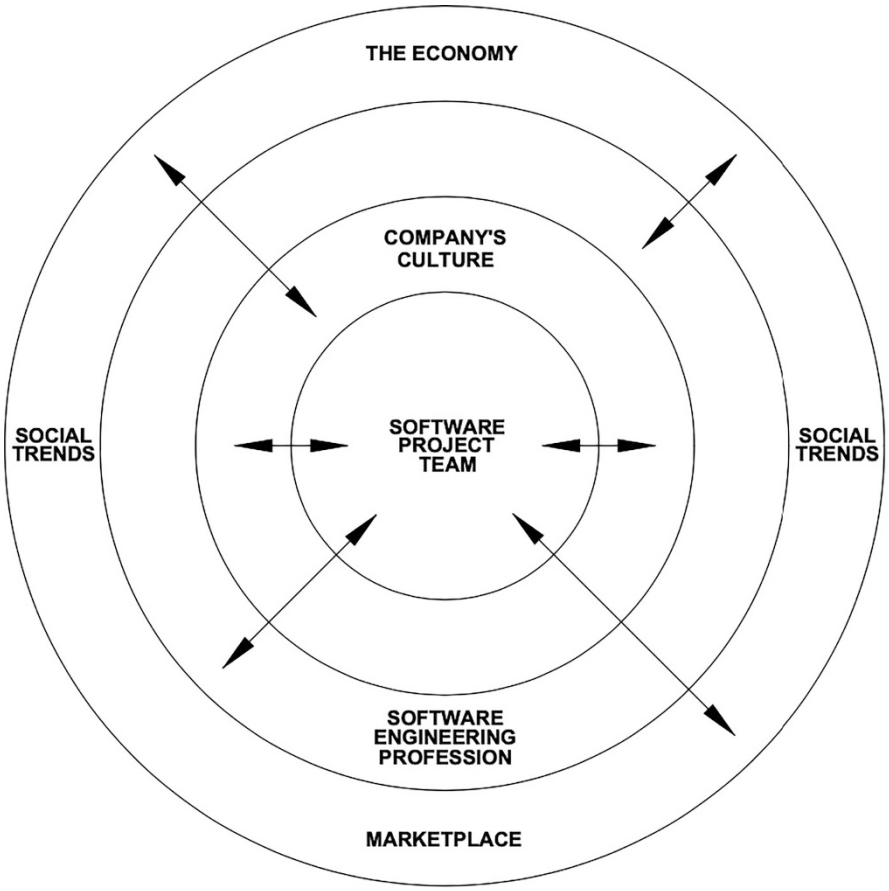

Fig. 1 Communities and cultures affecting software development (© Tim Clarke)

work could return to "normal." Examples of how people's actions keep systems working are also found in software development (Lopez et al., 2023).

A further example of how developers are influenced by their environment is given in work by Lopez et al. (2022), who studied the security behavior of software developers. They were driven to understand why known security vulnerabilities were still being embedded in software: why aren't the developers countering these known issues when the software is built. At the time, the finger was often pointed at software developers asking why they didn't "just do it." Findings showed that decisions that have an impact on security within code were not always made by developers and their teams but instead reflect the attitudes and priorities of companies and their clients. This provided evidence that the cultures of the company, the client, and the team all affected technical outcomes, not just the individual's experience and expertise.

2.2 Non-specialists such as Domain Experts

There are many ways in which non-specialists can and do contribute to systems development. For instance, a quick search on Google will show that crowdsourcing of ideas for new technologies and systems is common-place. Two non-specialist examples are discussed in this section as system creators: domain experts and "the public."

All software development requires developers to engage with the domain of application, i.e., where the software will be deployed, and the kind of functionality it is designed to embody. In some cases communicating the intricacies of that domain can be particularly challenging if the domain is specialist, for example, in scientific discovery (Chawla, 2015). In this case, it is common for the domain specialists themselves to create the software, and while they are specialists in their own field, they are not necessarily software or technology specialists. This demonstrates one group of non-specialist creators: domain experts. While some software may be produced simply to support the developer's own requirements, it is common for this software to be valuable to others and such software may be taken up in the wider community, supported by crowdsourced documentation (Pawlik et al., 2015).

The Maker Movement (Anderson, 2013; Hatch, 2014) is very much about opening up the world of "making" to a wide range of people, some of whom were already hobbyists, but others are new to making. The Maker Movement (MM) aims to make Do-It-Yourself making accessible to whoever wants to take part (Anderson, 2013). It explicitly aims to encourage people to make as well as consume artifacts. At its core is to collaboratively craft physical and digital artifacts using a diversity of machines, tools, and methods. The availability of affordable, powerful, and easy-to-use tools, coupled with a renewed focus on locally sourced products and community-based activities, has fueled this interest and made the movement feasible. A network of makerspaces has enabled the maker movement to become widespread and popularized worldwide.

The main principles of MM are to make, share, give, learn, play, participate, support, and change. Note that this is not just about making your own things but also sharing and supporting others. Websites such as instructables.com and makezine.com illustrate the outcomes of this ethos and demonstrate what can be achieved when people share and build on each other's ideas and creations. Although the pandemic dampened the opportunity to gather physically, the movement is still growing. The use of pre-formed kits such as Arduino[1] and e-textiles,[2] together with ready-made components in the form of software development kits, patterns, and libraries, also encourages a wide engagement.

This movement illustrates the power of community and of individual abilities to create artifacts that focus on things that they want. It also represents a change in

[1] Arduino is an open-source electronics platform based on easy-to-use hardware and software; see arduino.cc for more information.

[2] A field of electronics that combines electronic widgets (lights, batteries, sensors) with textiles.

mindset where people can see that they have the chance to shape technology to fit their own purpose rather than be driven by it.

3 People as Users of Software Systems

You might think it goes without saying that software is built to be used, and so people are users of software. It is worth remembering that software is intended to support peoples' goals and that software should be developed with users in mind, but this section goes beyond these simple platitudes. Here we explore two sociotechnical aspects of people as users that are relevant to systems development: that users are inventive and appropriate technology for their own context and that user feedback and behavior with existing systems influence future systems development.

3.1 Taking Account of Users and What They Do

During the 1970s, when software and its applications were becoming more widespread, the need to pay attention to the design of the interface so that it was usable by humans was recognized. The early focus on "man-machine interface" quickly evolved during the 1980s into human-computer interaction (HCI), a term that is still used today although its scope has increased considerably over the decades.

From the beginning, HCI drew on a range of disciplines including cognitive psychology, linguistics, computer science, and sociology. The goal was to design software that would take account of human characteristics such as attention and learning and influences on human behavior such as group processes and attitudes. HCI's focus was on how to take account of these traits in the design of interfaces and systems. For instance, an understanding of attention led to suggestions on how best to structure information, so that users could find what they needed more effectively, and how to use space and color on screens to direct users' attention to the salient points for the task in hand.

HCI also recognized the need for an iterative approach to development so that a range of expertise could be brought into play, and the emerging design may be checked with users. This evaluation of early prototypes and designs with users became the focus of the HCI design process in which emerging designs and prototypes are shown to and evaluated with intended user groups, and the results fed back into redesign (Preece et al., 1994).

The more recent term interaction design captures a focus that is much wider than that of the early HCI days. Interaction design today recognizes that the context of use has expanded away from "one user-one computer" and a wide range of different disciplines needs to be drawn upon in deciding what interactive products to develop and how to design them, including psychology and computer science but also product design, social sciences, and cognitive ergonomics. Interaction design also

recognizes the centrality of people and humans as users, co-designers, and creators in the design of interactive products. It is defined as "designing interactive products to support the way people communicate and interact in their everyday and working lives" (Rogers et al., 2023).

Alongside this change of emphasis is the recognition that designing *for* users isn't enough but that creators need to also design *with* users so that technologies can be truly human-centered (see Winter in this volume). But even with the best human-centered development process, users have a habit of appropriating technology for their own uses and it's not clear how it will be used until it is in the hands of the user population. Although software may be designed for particular purposes, people are very good at adapting the software for their own use and in molding it to their own context. This phenomenon prompted the introduction of "in-the-wild" studies of use and evaluation of early designs (Rogers & Marshall, 2017). The idea behind this approach is that systems are evaluated in situations that reflect as much as possible the context in which they will be deployed. While users may say that they will do things in a particular way, it's often the case that they do something different when faced with the situation "for real." The old adage that "what I do and what I say I do are not the same thing" applies in systems design and use too. Indeed, sociology of technology suggests that the usefulness of software is actively and socially constructed by users rather than merely perceived as a property of technology (Pinch & Bijker, 1987). For instance, studying ERP systems (enterprise resource planning systems), Abdelnour-Nocera et al. (2007) found that the context and local culture shaped the utility and usability of systems after they have been deployed.

The importance of encouraging developers to watch users interacting with their creations was realized a long time ago, and the importance of iteration is reflected in modern software development through the agile approach (Ashmore & Runyan, 2015; Zuber et al. in this volume). Agile software development recognizes the need for regular interactions with users and customers so that feedback is provided as the product evolves. In many cases, products are released regularly into real use so that value is delivered to the business often, and feedback may be based on real use.

Insight into how technologies are appropriated by communities of users can be gained by observing products in use and getting regular feedback. However, uncovering cultural norms such as assumptions, customs, beliefs, and habits of user communities is challenging, yet their impact on technology use is significant. For instance, Chavan and Gorney (2008) describe scenarios in which the use of mobile phones is influenced by cultural norms where technologies are shared and hence privacy and security are compromised. If cultural norms of communities are explored during the design process, then this kind of unexpected (to the designers) behavior could be accounted for. This example is one of many that led researchers to recognize that different approaches to design for indigenous communities may be needed (Winschiers-Theophilus & Bidwell, 2013), so that the role that technology plays can be better understood.

3.2 Software Use Influences Future Development

Human-centered development of new systems is important, but much technology design and development are based on evolving existing systems rather than being completely new inventions. Given what we said in the previous section, then it's not a surprise that existing use of systems influences how they evolve in the future. And we've got better at knowing how to collect data and derive information that allows us to do this. Nowadays, there is an inextricable interdependence between development and use.

The Lean UX approach is one example of software development where even an idea can be checked within the context of real use before development proceeds too far. A "minimum viable product" (MVP) is released for real use and user behavior is monitored to see whether and how the product is used. Gothelf and Seiden (2016, pp. 76–77) provide a simple example of an MVP produced by a company who thought that their customers would like a monthly newsletter. To test out this assumption before spending a lot of resource on developing it, they spent half a day producing a sign-up form available online. This MVP allowed them to collect evidence to support or refute their assumption, based on user feedback.

Another approach that focuses on online systems and obtains evidence of users' behavior is A/B testing (Kohavi et al., 2020). In A/B testing, different versions of the same system are delivered to different sets of users, and their performance is tracked according to a defined set of evaluation criteria. The performance of the two designs then helps to decide which one to implement more widely. The name "A/B testing" is based on the idea that there are two alternative designs—"A" and "B"—and that their quality is being tested via an experiment. Setting up appropriate criteria and implementing an experiment are not simple although this technique is used widely. Users are identified and assigned to groups randomly. They don't usually know that they are taking part in such an experiment, so next time you're online and the website or app looks different, perhaps you're helping the designers learn how users interact with their creations!

Other sources of information about how software is used come from customer reviews, which can affect the popularity and success of a product (Harman et al., 2012). App reviews from social media can also provide concrete improvements, for example, Twitter has been suggested as a good source of app reviews (Mezouar et al., 2018). However, it's not straightforward to extract useful information that can be acted upon by developers (Dabrowski et al., 2022). These areas are still the subject of research.

4 People in Partnership with Software Systems

People and software work in partnership in a way that is more than just "using" software: what people can *achieve*, how ideas and understanding *evolve*, and how people *behave* are all influenced by the software systems we use. For instance, what can be achieved through computer-aided design (CAD) systems has revolutionized the way in which buildings are designed and maintained by providing a high level of accuracy and detail that can be modified repeatedly and easily. Software visualization and manipulation of images that supports the analysis of satellite photos have allowed our understanding of how human activity has affected the planet to evolve. How software shapes our behavior is illustrated by the effect of accurate navigation systems on walking and cycling for leisure. Before these came along, such activities needed to be carefully planned and landmarks noted. With the support of accurate GPS tracking and detailed maps, it's possible to find where you are (almost) anywhere in the world and how to get to your destination. You just go out of the door and follow the prompts!

Taking this last point further, software-based systems not only influence our behavior in line with our original goals; it can also help people to change their habits. Persuasive technologies (Fogg, 2009) are explicitly designed to do this. For instance, behavior around domestic energy use can be changed by providing regular feedback to consumers about their energy consumption. Behavior of groups of people may also be changed by sharing consumption figures across a neighborhood. These kinds of persuasive technologies illustrate that the partnership between people and software can have a positive impact across large groups.

However, persuasion can sometimes be less positive. Many online sites are designed to persuade users to do something beyond their initial goal, such as buying an additional product or signing up for services other than those originally intended. At times, this is seen as a fair marketing technique, provided it is transparent, e.g., if purchasing a train ticket to visit a new city, it is likely that the user will also want accommodation, so why not offer them some options? Unfortunately, sometimes a more deceptive approach is implemented where the user feels tricked into "opting in" for something they didn't want. This approach has been referred to as deceptive or dark patterns (see www.deceptive.design).

On the whole, the partnership between software and people has positive outcomes, but although it is very malleable, software has its constraints, and it can have unintended consequences. These constraints may come from technical issues, such as the speed of rendering an image, or from the underlying design of a system, such as mismatches between data formats that prevent integration. Research into technical areas has and will continue to break new ground and what is a constraint today will have been solved next week, but it will be replaced by some other constraint and the cycle will progress. Constraints caused by design issues will also be resolved over time (although significant frustration may continue in the meantime), as more is learned about what users do and want to do.

Unintended consequences are particularly interesting to digital humanism. Baeza-Yates and Murgai in this volume, for example, highlights the ubiquity of bias on the Web. Another example of unintended consequences is provided by recommender systems, which are also discussed by Knees and Neidhardt in this volume. Recommender systems help a user to identify products or services that they might not otherwise have found. These systems are used by online retailers and streaming services, for example, to suggest the items someone might want to purchase or films they might like to download. They analyze data collected about a user such as previous searching or downloading behavior and aim to predict what that user may like; recommenders may also compare across their bank of users to make predictions. Exactly how these recommendations are arrived at depends on the algorithm used, which is a combination of filtering and prediction strategies. However, these systems can lead to the user's preferences simply being self-reinforcing, i.e., the same kind of articles, products, or views are presented again (Pariser, 2011), rather than introducing new ideas. People are prone to biases of various kinds, e.g., confirmation bias, and recommender systems can exacerbate this unless designed deliberately to introduce a novel perspective. If this happens, silos of opinion or creativity can form, which constrain rather than expand a person's outlook. The phenomenon of "algorithm hate," where users become dissatisfied or puzzled by the recommendations they receive, is an active area of research (Smith et al., 2022). However, if users were more aware of this tendency, then changing their behavior would lead to a change in recommendations.

5 Conclusions

Software systems development is a sociotechnical endeavor with people at its heart. Software both supports people in achieving their goals and shapes what can be done; in turn, the ways in which people use software affects how the software behaves and shapes its future evolution. Together, software and its users work in partnership to extend our capabilities, and despite its limitations, software systems have helped us achieve significant advances.

Moreover, both software creators and software users sit within communities, each of which has its own cultural norms that inform and influence software's development and use. Software creation, whether by professional developers or non-specialists, depends on ready-made components and technical assets created by others. Increasingly, non-specialist creators and users are influencing the technologies available.

Making people aware of the fact that software development and use are becoming increasingly interdependent may provide unexpected opportunities within digital humanism.

Discussion Questions for Students and Their Teachers

1. What consequences arise for digital humanism from the sociotechnical nature of the relationship between people and software this chapter describes?
2. Consider the number of software systems you use in a day. Which of them support you and which of them shape your behavior? Choose one that you use regularly and consider how your day would be affected if it wasn't available.
3. Investigate platforms for crowdsourcing ideas—how could crowdsourcing be used to good effect in the quest for digital humanism?
4. Discuss whether and how the communities you belong to influence your relationship with software—as a user or as a creator.

Learning Resources for Students

1. Bergman, O. and Whittaker, S.(2016). *The Science of Managing Our Digital Stuff*. MIT Press

   This book provides an account of how users manage all their digital stuff that seems to keep increasing each day. It explains why users persist with seemingly old-fashioned methods when there are alternative, maybe better approaches that have been designed by software companies.
2. CHASE conference proceedings (the list of papers is available through www.chaseresearch.org)

   This is an annual conference that showcases up-to-date research into the cooperative and human aspects of software development. The papers concentrate on software creators and include a range of issues related to human characteristics and their impact on software development.
3. Hatch, M. (2014) *The Maker Movement Manifesto*. McGraw Hill

   This introduces the Maker Movement and how anyone can get involved in making things and sharing what they have produced. It outlines the fundamentals behind the movement although in places it is a little evangelical.
4. Kohavi, R., Tang, D., and Ya, X. (2020). *Trustworthy Online Controlled Experiments: a practical guide to A/B testing*. Cambridge University Press

   This book was written by three experienced practitioners who have been running online experiments, also referred to as A/B testing, at scale for many years. It is readable and accessible to a wide range of readers and provides valuable detail backed up with specific examples that show the impact that applying this approach successfully can have.
5. Rogers, Y., Sharp, H. and Preece, J. (2023). *Interaction Design: beyond human-computer interaction*. Hoboken: Wiley

   This book provides a good introduction to a wide range of subjects within Interaction Design. Written by a cognitive scientist, an information scientist, and a software engineer, it brings together perspectives from three of the disciplines that have influenced Interaction Design. See id-book.com for more detail.
6. Segal J. and Morris C. (2008) Developing scientific software, *IEEE Software*, 25(4), 18–20

This introduction to a special issue on scientific development is a short piece exploring the nature of this kind of software creation. Further papers in the special issue delve into more detail about developing software for scientific discovery.

7. Sharp, H., Robinson H., and Woodman, M. (2000) 'Software engineering: community and culture', *IEEE Software*, 17(1), 40–47

   This is a relatively short read that explains how community and culture impact software development. The work is based on ethnographic studies of development.

Acknowledgments Many of my colleagues have shaped the views expressed in this chapter. In particular, I'd like to mention Yvonne Rogers, Jenny Preece, Tamara Lopez, and Hugh Robinson. The development of this chapter was supported by UKRI/EPSRC EP/T017465/1.

References

Abdelnour-Nocera, J., Dunckley, L., & Sharp, H. (2007). An approach to the evaluation of usefulness as a social construct using technological frames. *International Journal of HCI, 22*(1), 157–177.

Anderson, C. (2013). *Makers*. Random House Business Books.

Ashmore, S., & Runyan, K. (2015). *Introduction to agile methods*. Addison Wesley.

Chavan, A. L., & Gorney, D. (2008). The dilemma of the shared mobile phone---culture strain and product design in emerging economies. *Interactions, 15*(4), 34–39.

Chawla, D. S. (2015). The Unsung heroes of scientific software. *Nature, 529*, 115–116.

Dabrowski, J., Letier, E., Perini, A., & Susi, A. (2022). Analysing app reviews for software engineering: a systematic literature review. *Empirical Software Engineering, 27*, Article 43.

Fogg, B. J. (2009) A behavior model for persuasive design. In *Proceedings of the 4th International Conference on Persuasive Technology (Persuasive '09)*. ACM, New York, NY, Article 40.

Furniss, D., Back, J., Blandford, A., Hildebrandt, M., & Broberg, H. (2011a). A resilience markers framework for small teams. *Reliability Engineering & System Safety, 96*(1), 2–10.

Furniss, D., Blandford, A., & Mayer, A. (2011b). Unremarkable errors: Low-level disturbances in infusion pump use. In *Proceedings of the 25th BCS Conference on Human-Computer Interaction (BCS-HCI '11)* (pp. 197–204). BCS Learning & Development Ltd.

Gothelf, J., & Seiden, J. (2016). *Lean UX*. O'Reilly.

Harman, M., Jia, Y., & Zhang, Y. (2012) App store mining and analysis: MSR for app stores. In *Proceedings of the 9th IEEE Working Conference on Mining Software Repositories (MSR 12)* (pp. 108–111).

Hatch, M. (2014). *The maker movement manifesto*. McGraw Hill.

Klein, L. (2014). What do we actually mean by 'sociotechnical'? On values, boundaries and the problems of language. *Applied Ergonomics, 45*, 137–142.

Kohavi, R., Tang, D., & Ya, X. (2020). *Trustworthy online controlled experiments: A practical guide to A/B testing*. Cambridge University Press.

Kulits, P., Wall, J., Bedetti, A., Henley, M., & Beery, S. (2021). ElephantBook: A semi-automated human-in-the-loop system for elephant re-identification. In *ACM SIGCAS Conference on Computing and Sustainable Societies (COMPASS '21)* (pp. 88–98). Association for Computing Machinery.

Lopez, T., Sharp, H., Bandara, A., Tun, T., Levine, M., & Nuseibeh, B. (2022). Security responses in software development. *ACM Transactions on Software Engineering and Methodology, 32*(3), 1–29.

Lopez, T., Sharp, H., Wermelinger, M., Langer, M., Levine, M., Jay, C., & Nuseibeh, B. (2023) Accounting for socio-technical resilience in software engineering. In *Proceedings of CHASE 2023*, Melbourne, IEEE.

Lotz, N., & Sharp, H. (2020). Challenges for interaction design education in the South: A case study of Botswana. *Journal of International Development, 32*(1), 62–84.

Mezouar, M. E., Zhang, F., & Zou, Y. (2018). Are tweets useful in the bug fixing process? An empirical study on Firefox and Chrome. *Empirical Software Engineering, 23*, 1704–1742.

NASA. Accessed June 16, 2023., from https://jwst.nasa.gov/content/forScientists/publications.html

Pariser, E. (2011). *The filter bubble: What the Internet is hiding from you*. Penguin.

Pawlik, A., Segal, J., Petre, M., & Sharp, H. (2015). Crowdsourcing scientific software documentation: A case study of the NumPy documentation project. *Computing in Science and Engineering, 17*(1), 28–36.

Pinch, T., & Bijker, W. (1987). The social construction of facts and artifacts. In W. Bijker, T. Hughes, & T. Pinch (Eds.), *The social construction of technological systems* (pp. 17–50). MIT Press.

Preece, J., Rogers, Y., Sharp, H., Benyon, D., Holland, S., & Carey, T. (1994). *Human-Computer Interaction*. Addison-Wesley.

Robinson, H., & Sharp, H. (2009). The emergence of object-oriented technology: the role of community. *Behaviour and Information Technology, 21*(3), 211–222.

Rogers, Y., Sharp, H., & Preece, J. (2023). *Interaction Design: Beyond human-computer interaction* (6th ed.). Wiley.

Rogers, Y., & Marshall, P. (2017). *Research in the wild*. Morgan & Claypool.

Sharp, H., Lotz, N., Mbayi-Kwelagobe, L., Woodroffe, M., Rajah, D., & Turugare, R. (2020). Socio-cultural factors and Interaction Design in Botswana: Results of a video diary study. *International Journal of Human-Computer Studies, 135*, 102375.

Sharp, H., Robinson, H., & Woodman, M. (2000). Software engineering: Community and culture. *IEEE Software, 17*(1), 40–47.

Smith, J. J., Jayne, L., & Burke, R. (2022). Recommender systems and algorithmic hate. In *Proceedings of the 16th conference on recommender systems (RecSys '22)* (pp. 592–597). ACM.

Wang, H., Miao, Z., Zhang, C., Wei, X., & Li, X. (2021). K-SEIR-Sim: A simple customized software for simulating the spread of infectious diseases. *Computational and Structural Biotechnology Journal, 19*, 1966–1975.

WDM. (2023). Accessed June 16, 2023, from https://www.wdm.co.uk/software

Winschiers-Theophilus, H., & Bidwell, N. J. (2013). Toward an Afro-centric indigenous HCI paradigm. *International Journal of Human-Computer Interaction, 29*(4), 243–255.