The Trouble with Troubleshooting

Tracey Booth
Centre for HCI Design
City, University of London
London, UK
tracey.booth.1@city.ac.uk

Jon Bird
Centre for HCI Design
City, University of London
London, UK
jon.bird@city.ac.uk

Simone Stumpf
Centre for HCI Design
City, University of London
London, UK
simone.stumpf.1@city.ac.uk

Abstract

Tinkering is an empowering and creative approach to making but it is a poor strategy for fixing bugs. We report new empirical findings which suggest that a trial and error tinkering approach is not an effective troubleshooting strategy and that a more structured, systematic approach can help makers locate and resolve their problems more effectively. There is an opportunity for HCI to develop tools to support makers' troubleshooting, a role it currently plays in the end-user programming domain.

Author Keywords

Making; tinkering; troubleshooting.

ACM Classification Keywords

H.5.m. Information interfaces and presentation (e.g., HCI): Miscellaneous;

Introduction

Tinkering is a foundation of making [2], which aims to encourage people who are not experts in programming or electronics to 'have a go', and to discover and learn from their hands-on experiences, including their mistakes. For example, the Maker Summit report argues that, "*Educators should offer assistance with difficult technical tools and processes, but stay at a distance; as a mainly self-directed learning experience, the project should allow learners to make mistakes and learn how to fix them on their own* [1, p.10]. However, in this provocation, we argue that one of the most difficult

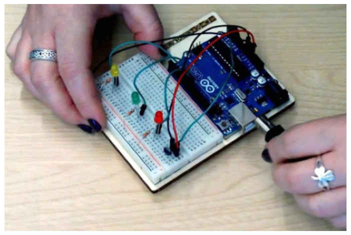

Figure 1. A participant constructing a prototype circuit in our study. The task involved connecting a temperature sensor to an Arduino and writing a short program to read the sensor and then visualize the values using LEDs.

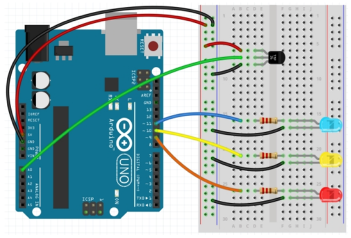

Figure 2. Fritzing layout of the simplest circuit configuration for the study task.

technical processes to learn is how to find and fix mistakes. Knowing where to look for bugs, what evidence to look for, how to test for the existence of bugs and how to interpret the results are all crucial skills in successful troubleshooting [6] and currently makers are not given enough assistance in developing these skills. We report new empirical findings which suggest that a trial and error tinkering approach is not an effective troubleshooting strategy and that a more structured, systematic approach can help makers locate and resolve problems more effectively. A key challenge is to develop tools that support makers to systematically fix mistakes while still enabling them to creatively tinker.

Study

To gather empirical data about problems that makers encounter when developing physical computing devices, we conducted an empirical, think-aloud study involving 20 adults. All had experience of using Arduino, but none were professional physical computing developers. Each participant attended an individual session in which they were given 45 minutes to construct an Arduino prototype that used three LEDs to visualize values read from a temperature sensor (Fig.1 and Fig.2). We gave them all necessary equipment and a specification of the required behavior of the physical computing device; they also had access to the Internet and were allowed to use any online resources. We video-recorded these sessions and later transcribed the recordings.

Our first analysis of these data [3] discovered that participants experienced many obstacles when developing the prototype and as a result they made lots of mistakes, resulting in bugs that they then needed to fix. On average, participants spent most of their time trying to fix bugs that they had introduced. Although participants were able to fix 70% of their bugs, the ones they were unable to resolve had severe implications for task success. Of the 14 participants who failed the task, 10 did so due to unresolved circuit bugs – faults in circuit construction. Circuit bugs which proved difficult to resolve included connecting component legs to the wrong Arduino pins, inadequately seated connections, and the omission of necessary components, e.g. resistors. We also discovered that when trying to fix their bugs, participants frequently introduced new bugs, as has been observed in other end-user programming domains [5]. We found that some circuit bugs proved particularly difficult for participants to localize, and were subsequently misdiagnosed as errors in the program, or in a different part of the circuit.

How do makers troubleshoot?

In a new analysis of our study data, we focused on participants' troubleshooting of circuit bugs, to identify why some were fixed whereas others were not. We analyzed troubleshooting by all those participants (14/20) who noticed and attempted to resolve at least one circuit-related problem – the type of bug most frequently responsible for task failure [3]. Of these 14 participants, only two successfully completed the task. The six participants not included in the analysis either experienced only programming bugs (4) or no bugs at all (2). A participant was considered to be troubleshooting a bug if they noticed the symptoms and indicated some concern and subsequently made an attempt to investigate the problem. A simple example of troubleshooting is a when a participant who, holding the sensor at run time, was surprised that an LED did not light up and so then inspected the wiring to see why this was the case.

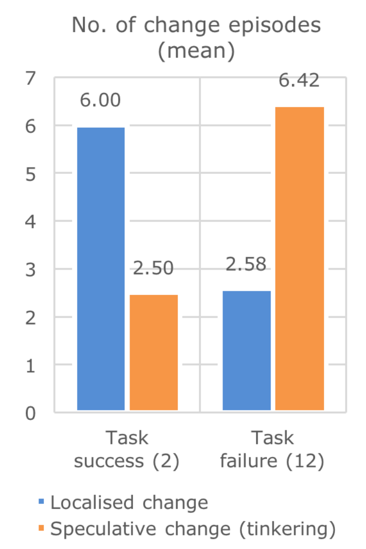

Figure 3. A comparison of the way that successful and unsuccessful participants made changes in an attempt to resolve their bugs. Those who successfully completed the task usually localized errors before making changes, whereas those who failed the task 'tinkered' more – making speculative changes in the hope of resolving their problem. It's important to note that successful participants sometimes tinkered, and unsuccessful participants sometimes localized their bugs before making changes.

Most circuit bugs only became apparent when run-time behavior alerted the participant to a potential problem. Participants then carried out a range of troubleshooting activities to fix the bug. Some were systematic and carried out activities such as inspection and testing to recognize, localize and identify the bug, changing the circuit to fix the bug and further inspection and testing to verify the fix. However, much of the troubleshooting we analysed involved trial and error tinkering, which, in this context, we define as making speculative changes (to the program or circuit) in the hope that it will fix the problem and thereby reveal the cause. For example, P01 had miswired the temperature sensor, which resulted in unpredictable behavior. They made several speculative changes that they hoped might identify and resolve the cause of this, including changing the voltage supply to the sensor: *"So, I might try seeing what it does at 3.3 volts and see if that changes things. Why the flip not?!"*. Crucially, this tinkering not only did not resolve the problem, it introduced a new circuit bug for them to troubleshoot.

In contrast, a useful strategy when troubleshooting either software or hardware is to isolate the part of a system that is suspected to be responsible for a bug. Reducing the size of the system to be analysed can help to eliminate potential causes of a problem and simple tests can then then be performed to diagnose the exact cause. Although there were far more cases where participants didn't attempt to systematically isolate a bug, we did observe a few instances where this was done effectively. For example, P02 seated one of their LEDs the wrong way round in the breadboard. When it did not light up as expected, they isolated the LEDs from the rest of the circuit (and each other) to establish whether their LEDs were correctly oriented*:* *"I'm just going to test that the LEDs light up when I connect power to them, so I know they're the right way round."*. When the same LED still didn't light up but the others did, they concluded it was the wrong way round: *"But if I touch that one, it doesn't light up. So, therefore these are all the other way round."* They fixed the problem by turning the LED around and then it lit up correctly.

We found that the two participants who successfully completed the prototype were generally confident that they knew where the error was located *before* they attempted to fix it, in contrast to the unsuccessful ones, whose uncertainty was evidenced by the comments they made when making changes to their circuits, such as "Perhaps…", "Let's see…", "I'll try…" and "Maybe". Unsuccessful participants often made changes to their circuit in the hope that doing so would help them locate the source of the problem, a trial and error approach that often led to additional problems. Overall, the successful participants were far more likely to make changes *after* localizing the problem rather than make trial and error changes; this is in contrast to the unsuccessful candidates who were more likely to adopt a tinkering approach (Fig. 3).

Ten of the twelve participants tried to fix circuit bugs by adopting a tinkering approach at some point in their troubleshooting, including nine of the participants who failed the task due to circuit bugs. Few of these attempts were successful: only 16% (13/82) of the instances that we observed resulted in a bug fix, while in 52% (43/82) new bugs were introduced into the circuit. In contrast, more systematic troubleshooting was far more effective, that is, where participants showed evidence of having localized the cause of their problem before attempting to

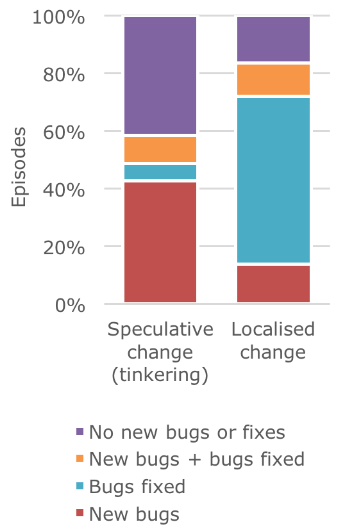

Figure 4. A comparison of the outcomes of the two approaches to making changes in an attempt to fix bugs. Speculative changes (tinkering) introduced far more bugs and resolved far fewer bugs than changes that were made after a bug had been localized.

fix it. While there were fewer instances of this approach, they were much more successful: 70% (30/43) resulted in bugs being fixed and only 26% (11/43) led to new bugs (Fig.4 summarizes these results).

It is worth noting that using a tinkering approach to troubleshooting was not just observed in participants who were less experienced in physical computing, it was also employed by the most experienced makers in our study. While tinkering was the primary troubleshooting approach for many participants, in some cases, when faced with a problem, participants started with a systematic approach to troubleshooting, but when their efforts failed to fix the problem, or even provide them with information that might help to focus in on the cause, they resorted to tinkering.

Summary

Tinkering is an empowering and creative approach to making but it is a poor strategy for fixing bugs. There is an opportunity for HCI to develop tools to support makers' troubleshooting, a role it currently plays in the programming domain. There has been extensive HCI research aimed at helping end-user programmers diagnose problems, generate and test hypotheses, and localize bugs that have led to tools such as the Idea Garden [4]. Makers would benefit from similar support tools, and this is the focus of our current research.

Tracey Booth is a PhD student whose research focuses on investigating how to develop support tools for end-user developers doing physical computing and spends lots of her time closely observing people using Arduino. **Jon Bird** is a lecturer in pervasive computing and uses Arduino in his research to create public interactive displays and wearables. He has been running Arduino workshops for over a decade in different UK universities. **Simone Stumpf** is a senior lecturer in HCI and her research focuses on supporting end-users in programming and physical computing.

References

1. American Society for Engineering Education. *Envisioning the future of the maker movement*: Summit Report. Washington, DC, 2016.
2. Massimo Banzi, *Getting started with Arduino*, 1 edition. Sebastopol, CA, USA: Make: Books, O'Reilly Media, Inc., 2009.
3. Tracey Booth, Simone Stumpf, Jon Bird, and Sara Jones. 2016. *Crossed wires: investigating the problems of end-user developers in a physical computing task*. In Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems (CHI '16). ACM, New York, NY, USA, 3485-3497. DOI: https://doi.org/10.1145/2858036.2858533
4. Jill Cao, Scott D. Fleming, Margaret Burnett, and Christopher Scaffidi, *Idea Garden: situated support for problem solving by end-user programmers*, Interact. Comput., vol. 27, no. 6, pp. 640–660, Jan. 2015.
5. Andrew J. Ko and Brad A. Myers, *Development and evaluation of a model of programming errors*, in 2003 IEEE Symposium on Human-Centric Computing Languages and Environments, 2003. Proceedings, 2003, pp. 7 – 14.
6. Nancy M. Morris and William B. Rouse, *Review and evaluation of empirical research in troubleshooting*, Hum. Factors: J. Hum. Factors Ergon. Soc., vol. 27, no. 5, pp. 503–530, Jan. 1985.