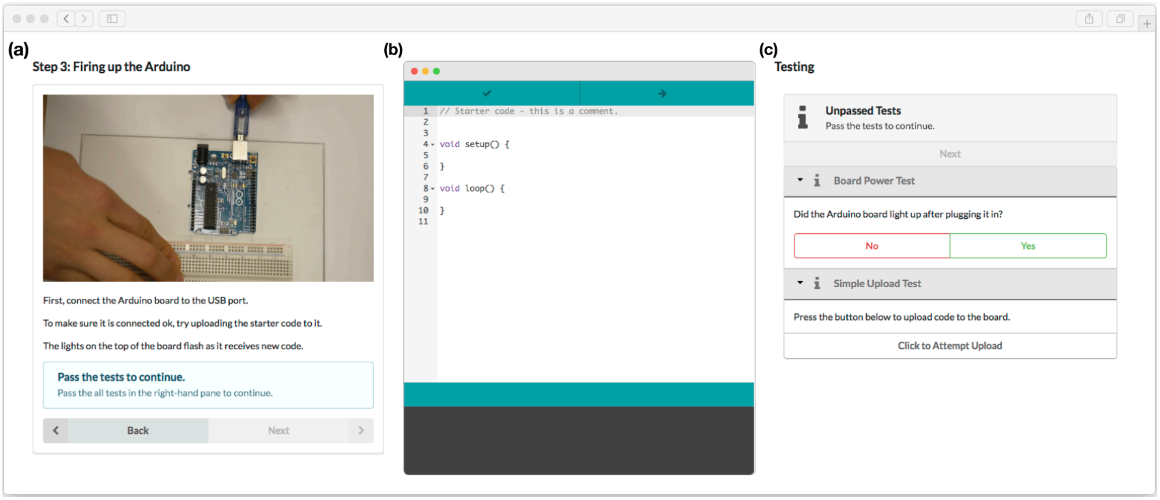
ElectroTutor: Test-Driven Physical Computing Tutorials

Jeremy Warner[*†], **Ben Lafreniere**[*], **George Fitzmaurice**[*], **Tovi Grossman**[*‡]

[*]Autodesk Research, Toronto, ON, Canada [†]UC Berkeley, CA, USA [‡]University of Toronto, ON, Canada
{*firstname.lastname*}@autodesk.com jeremy.warner@berkeley.edu tovi@dgp.toronto.edu



Step 3: Firing up the Arduino

First, connect the Arduino board to the USB port.

To make sure it is connected ok, try uploading the starter code to it.

The lights on the top of the board flash as it receives new code.

Pass the tests to continue.
Pass the all tests in the right-hand pane to continue.

Back Next

```
// Starter code - this is a comment.


void setup() {

}

void loop() {

}

```

Testing

Unpassed Tests
Pass the tests to continue.

Next

Board Power Test

Did the Arduino board light up after plugging it in?

No Yes

Simple Upload Test

Press the button below to upload code to the board.

Click to Attempt Upload

Figure 1: The ElectroTutor interface. The system provides step-by-step tutorials (a) for physical computing projects, a code editing panel (b) and integrated tests used to verify that each step was completed successfully (c).

ABSTRACT

A wide variety of tools for creating physical computing systems have been developed, but getting started in this domain remains challenging for novices. In this paper, we introduce *test-driven physical computing tutorials*, a novel application of interactive tutorial systems to better support users in building and programming physical computing systems. These tutorials inject interactive tests into the tutorial process to help users verify and understand individual steps before proceeding. We begin by presenting a taxonomy of the types of tests that can be incorporated into physical computing tutorials. We then present ElectroTutor, a tutorial system that implements a range of tests for both the software and physical aspects of a physical computing system. A user study suggests that ElectroTutor can improve users' success and confidence when completing a tutorial, and save them time by reducing the need to backtrack and troubleshoot errors made on previous tutorial steps.

Author Keywords
Physical computing; software learning; reactive tutorials.

ACM Classification Keywords
H.5.m. Information interfaces and presentation: Misc.

UIST '18, October 14–17, 2018, Berlin, Germany
© 2018 Copyright is held by the authors. Publication rights licensed to ACM.
ACM ISBN 978-1-4503-5948-1/18/10…$15.00
https://doi.org/10.1145/3242587.3242591

INTRODUCTION

A variety of toolkits for building physical computing systems have been developed, but starting out in this domain can be daunting for novices. The process of going from schematics and code posted on the web to a functional system is fraught with pitfalls, and step-by-step tutorials can lead users to perform actions without fully understanding their intention. Research in this area has shown that starting out in this domain remains difficult, with many students and hobbyists struggling to realize control over their designs [4, 27]. At the extreme, Booth et al. showed that less than one third of participants could complete a simple Arduino assembly task [4]. To address these challenges, recent work has focused on supporting development [1, 16, 17] and debugging [9, 26, 34, 38] of embedded systems, but there has been little work improving upon the format of tutorials for physical computing projects.

In the software learning literature, there have been many advancements in interactive tutorials designed to make them more engaging and increase users' success. Of particular promise, reactive tutorial systems [10, 30] can respond to a user's actions, automatically progress through steps upon completion of instructions, and provide error correction and guidance in response to detecting problems. However, these techniques are difficult to extend to physical computing projects, which integrate both software and physical subtasks, thus complicating user activity tracking.

In this paper, we introduce *test-driven physical computing tutorials*, a new approach that is inspired by test-driven development practices in software engineering [2]. These tutorials include interactive tests on each step, requiring the user to verify wiring, electronics, code, or their own knowledge before they can proceed to subsequent steps. Test-driven tutorials can include validations that are both automatically carried out by the system, or manually by the user. Involving the user in the validation process has the additional potential benefit of increasing the user's engagement and comprehension of individual tutorial steps.

The main contribution of this work is in applying the test-driven approach to physical computing tutorials. Specifically, we contribute a taxonomy of test types for this domain across three dimensions: the domain being tested (physical or software), how the test is set up, and how the test is verified. We then explore the breadth of this taxonomy in ElectroTutor, our prototype system. ElectroTutor's interface is divided into three panels – one for the tutorial instructions (Figure 1a), one for the coding environment (Figure 1b), and one for the interactive tests (Figure 1c). Users go through the tutorial steps as in any other tutorial, but must successfully complete the interactive tests on each step before proceeding.

Finally, we contribute an initial user study comparing ElectroTutor to an equivalent baseline tutorial without interactive tests. Our results show that the addition of tests enables users to complete tutorials with fewer instances of backtracking to troubleshoot and fix problems introduced on previous steps. Our results also suggest that this approach may be able to decrease task completion time and increase task success rate.

We conclude the paper by discussing limitations and areas for future work, including authoring environments for test-driven tutorials, and applications to other domains.

RELATED WORK

Test-driven physical computing tutorials are related to two main areas of prior research in the HCI literature – interactive tutorials, and tools for electronics design and debugging.

Interactive Tutorials

Early HCI research recognized the challenges of developing effective learning materials for software systems, and suggested the benefits of minimalist and task-centered learning materials [5, 6, 35]. Subsequent work explored the benefits of animated and multi-media assistance [8, 15, 29, 30] and demonstrated that these techniques can help illustrate dynamic operations, but can also enforce a passive learning process and force users to work at the pace of the video demonstration [14]. The Pause-and-Play system addresses this limitation by automatically pacing video playback in response to a user's progress through the tutorial [30].

More recently, a range of tutorial systems have been proposed that integrate directly with the software being learned [3, 8, 10, 18, 22, 24, 25, 28, 30, 31], to tightly link learning materials with the content being taught, and to provide reactive features that respond to the tutorial user's actions. For example, Stencils-based tutorials [21] present instructions within an application, and ensured users perform the correct behavior by overlaying a stencil onto the user interface. Fernquist et al. [10] later defined a design space for interactive tutorial systems based on their scope (teaching individual *features*, lower-level *tasks*, or higher-level *content-centric* workflows) and their interactivity (*passive* presentation of learning materials, *active* mechanisms that allow users to try out concepts, and *reactive* tutorials that are aware of a user's interactions and can respond to them). ElectroTutor represents a *content-centric reactive* tutorial system, where the tasks involve physical computing projects, consisting of both a software programming component and a physical electronics component.

A number of interactive tutorial systems have been developed for tasks that include working in the physical world [21, 23, 32, 36]. ElectroTutor builds on this body of work, with a focus on how physical computing tutorials can be made reactive by adding tests that validate that the user is on the right track, or prompt them to debug the project as they build it.

The idea of test-driven learning has also been embraced by the online education community, with studies showing the benefits of having students complete quizzes [11] or prompts to explain what they are learning [7]. For instance, Codecademy includes evaluation and knowledge reinforcement directly into the process of working through lessons [39], and Khan Academy incrementally adjusts the tests they provide based on a user's progress and success on quizzes [40]. In the HCI domain, recent work has investigated the potential of *in-video prompting* – presenting questions to learners during video playback to prompt reflection on the content, and to elicit more specific feedback on course materials for instructors [33]. The tests in our system share the goal of prompting a deeper understanding of the material being taught. However, our tests also serve other functions, such as validating that steps are completed correctly, and providing specific corrective feedback to the user in response to common mistakes.

Electronics Design and Debugging Tools

Prior work has shown that novice users face substantial difficulty in designing and building physical computing systems. Studies by Booth et al. and Mellis et al. asked novices to construct simple electronics projects and documented the challenges they faced [4, 27]. These studies revealed that novice users can run into trouble choosing the correct components, wiring components together, programming logic and variables, and debugging of hardware and software components.

Several research systems have been developed to address these challenges. Toastboard [9] is an intelligent breadboard that assists novices with debugging through LED indicators on the board itself, and a software interface that provides troubleshooting tips. Bifröst [26] instruments both the hardware and software components of embedded computing projects to help users trace the state of the system and assists in debugging. Trigger-Action Circuits [1] enables users to specify desired functionality at a behavioral level, and generates designs and corresponding instructions for assembling them. Finally, a number of systems have been developed that aid in sensing the state of the electronics components in embedded systems [9, 34, 38], data which could aid in debugging and troubleshooting.

Whereas many of the above systems are focused on developing novel hardware and sensing techniques, our work investigates how such sensing can be used to enable reactive tutorials for physical computing projects. We also adopt a "discount sensing" approach, using unused pins from the project's Arduino board, or an additional off-the-shelf Arduino board for targeted probing. In the short term, this could extend the reach of reactive tutorials into the physical computing domain, while some of the improved sensing technologies described above continue to develop.

TEST-DRIVEN PHYSICAL COMPUTING TUTORIALS

Test-driven development (TDD) is a software development process that involves the repetition of short development cycles in which functional requirements are turned into specific test cases, and the software is then improved to pass these tests [2]. The idea is to keep individual development cycles simple and keep the developer focused on meeting specific functional requirements.

Test-driven physical computing tutorials adapt the general philosophy of TDD to tutorial systems. Specifically, the idea is to add tests to individual steps of a tutorial to focus the user's effort on meeting a set of requirements for that step. These could be functional requirements (i.e., validating that the step has been performed correctly), or learning requirements (i.e., validating that user has learned certain information, or prompting reflection on the actions performed in a step). One goal is to prevent a user from reaching the end of a tutorial and finding that the device they have built does not work. Diagnosing the source of this kind of failure can be frustrating and difficult [4]. A secondary goal is to avoid the scenario where the user reaches the end of the tutorial and has a working system, without a full understanding of what they did to make it work.

As a starting point for developing a test-driven tutorial approach, we considered the design space of how tests could be integrated into tutorial steps, and the roles they could play.

Design Considerations

Below are a set of important considerations related to how tests could be integrated into tutorials.

Purpose. Tests can serve a range of purposes, including evaluating the user's knowledge, emphasizing certain information to the user, or validating that a step is completed correctly. In ElectroTutor, we focus on the purposes listed above, but we also see potential for tests to be used to provide feedback or data to the tutorial author, or to an instructor, an idea which has seen some exploration in recent work [33].

Authoring. Another consideration is how tests will be authored. In ElectroTutor we adopt a model where the tutorial author creates the tests for a step, based on their understanding of the functional and learning requirements for that step. However, there may be advantages to enabling tutorial users, or their peers, to create tests as well, with the intent of prompting a deeper reflection and understanding of the tutorial content.

Guidance. In TDD, tests typically pass or fail, but given the broader set of intents for tests in test-driven tutorials, tests could provide a range of different types of guidance to users (e.g., visualizing the system state to promote deeper understanding, or providing specific corrective feedback or debugging tips in response to certain conditions). We include a number of these approaches in ElectroTutor.

Adaptivity. Finally, tests can be *adaptive*, being included on a step depending on a model of the user's knowledge, or results and measurements from previous tests. Tests could also be included for a randomly-selected subset of users, to sample the knowledge of a larger group. In ElectroTutor, we adopt a fixed set of tests that all users must pass, but we view adaptive testing as an interesting area for future work.

Taxonomy of Tests for Physical Computing Tutorials

The above set of dimensions provide a general design space for test-driven tutorials. Next, we consider a more specific taxonomy of test types for physical computing tutorials. The dimensions of this taxonomy include how the test is initialized (*Test Setup*), how it is verified (*Test Verification*), and the domain of the test (*Test Domain*). Figure 2 illustrates this taxonomy, resulting in 10 unique test classes.

Test Setup	Test Verification	Test Domain	
		Software	*Physical*
Manual	*Manual*	$S_{M,M}$	$P_{M,M}$
	Automatic	$S_{M,A}$	$P_{M,A}$
Automatic	*Manual*	$S_{A,M}$	$P_{A,M}$
	Automatic	$S_{A,A}$	$P_{A,A}$
None	*Manual*	$S_{N,M}$	$P_{N,M}$
	Automatic	**Reactive tutorial systems, e.g.:**	
		[10, 30]	[21, 32]

Figure 2. A taxonomy of tests that can be used within test-driven tutorials for physical computing. The ten resulting test classes are numbered for later reference.

Test Setup

The setup up or initialization for a test can be *manual*, *automatic*, or *none*.

Manual: In the case of *manual* setup, the user is required to perform additional actions (beyond those required by the tutorial step) before the test is performed. For example, the user may be asked to place probes on a specific hardware component or pin, select a block of code, or manually engage a sensor that will be used by the test. This has the advantage of engaging the user in the testing process, but also has a potential downside in that these setup actions could introduce additional errors (e.g., if the user places probes incorrectly).

Automatic: With *automatic* setup, the system automatically performs any required initialization steps for the test, potentially behind-the-scene without the user's knowledge. For example, test code could be injected onto a board to test if an LED is functioning or inserted properly. Automatic setup has the advantage that it may be less error prone than having the user manually perform these actions.

None: In some cases, no setup is required for a test beyond completing the current step of the tutorial. This is the case with many reactive tutorial systems, where the tutorial system simply confirms that the user has performed a required instruction (e.g., testing if they selected a specified tool). This class of test also includes those that don't measure anything from the hardware or software (e.g., a test that asks the user to respond to a question to test their knowledge, or a test that asks the user to confirm that a certain outcome resulted from the current step of the tutorial).

Test Verification
In addition to the setup of a test, the verification of the test can also vary, being either *manual* or *automatic.* Verification cannot be *none* as we require tests to pass before allowing a user to proceed to a subsequent step.

Manual: In some cases, it can be advantageous to have the user evaluate the success of a test. For example, the user could answer a question "Does the light turn on when you press the button". We adopt this approach in ElectroTutor to address cases in physical computing tutorials that are difficult to sense automatically, and cases where manual verification may assist with knowledge retention. This helps avoid the need for specialized hardware or instrumentation.

Automatic: In TDD, tests are typically fully automatic, with the system evaluating whether certain conditions are met. For tests in tutorials, the system could similarity evaluate whether certain conditions hold and a test has passed. This type of test may require the system to watch for specific condition as the hardware runs, or verify that a variable is set to a specified value at runtime.

Test Domain
An interesting aspect of physical computing tutorials is that they contain steps related to both software and physical electronics. As such, the domain of tests can be either *software* or *physical*.

Software: Software tests are concerned with the software component of the physical computing project being built, or aspects of the development environment being used to write code for the project. For example, testing that required code for a certain step has been entered properly, that a variable takes on an expected value during runtime, or that the user understands what a specific line of code is used for.

Physical: Physical tests are concerned with the physical and electronic components of the system being built, such as whether hardware has been inserted and wired properly, or physical components are functioning as expected.

ELECTROTUTOR
ElectroTutor is a tutorial system that merges traditional step-by-step tutorial content with interactive verification tests. The interface for ElectroTutor, consists of static instructional content (Figure 1a), an Arduino-like integrated development environment (Figure 1b), and an interactive testing panel (Figure 1c). We discuss each of these components in turn.

Instruction Panel
The instruction panel displays the current step's instructions to the user in text with associated images or short videos. 'Next' and 'Back' buttons enable the user to navigate through the steps of the tutorial. However, for steps with associated tests, the user can only proceed once the tests for that step have been passed. Until all tests have been passed, the next button is disabled and a message is displayed to complete the tests before progressing (Figure 3).

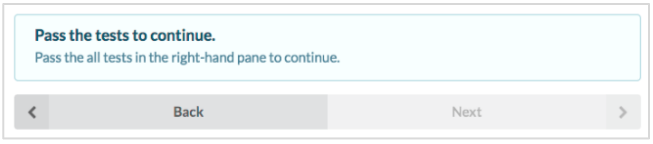

Pass the tests to continue.
Pass the all tests in the right-hand pane to continue.

Figure 3. Users are prompted to complete the tests associated with the current step before proceeding.

Development Panel
The development panel provides an integrated Arduino-like IDE, with compile and upload buttons, a code editing area, and an output panel. At the start of the tutorial, the IDE is initialized with the standard empty `setup()` and `loop()` functions. At any time, the user can edit code and compile or upload it to the connected Arduino board for the project being assembled.

Testing Panel
The testing panel displays any associated tests for the current step. Individual tests are expanded when the user starts a step, and are automatically collapsed once completed. Each test provides instructions to the user, and may include interactive widgets for the user to interact with as part of performing the test. A check mark icon or 'X' icon indicate if each individual test has been passed or failed (Figure 4).

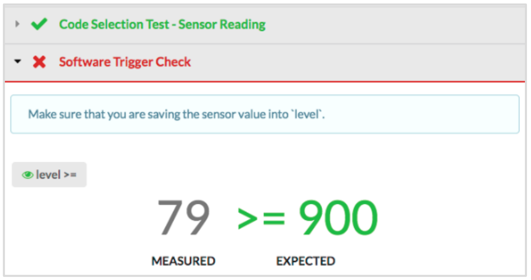

✓ Code Selection Test - Sensor Reading
✗ Software Trigger Check
Make sure that you are saving the sensor value into 'level'.

Figure 4. A step with two tests. The first test is collapsed, with a green check mark indicating it has passed. An 'X' icon indicates the second test has failed, and a custom error message is displayed.

Tests can display custom-authored error or feedback messages when the test fails (e.g., to provide corrective feedback), and the user can repeat a test as many times as they wish until it has passed. When all tests for a step have been successfully passed, the user can proceed to the next step (Figure 5).

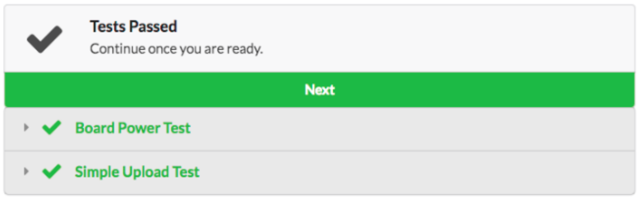

Tests Passed
Continue once you are ready.

Figure 5. The user can proceed once all tests have passed.

Test Types

ElectroTutor implements a set of tests that exemplify the various test types defined in our taxonomy (Figure 2). For some test types, the system uses an auxiliary circuit probe device, consisting of an Arduino Uno in a 3D printed case, to record physical measurements (Figure 6). Alternately, unused input pins on the project's Arduino board could be used for this purpose. In this section, we describe the range of tests implemented in the system, referring to example tests used in a "Light-Sensitive Alarm" project described later in the paper.

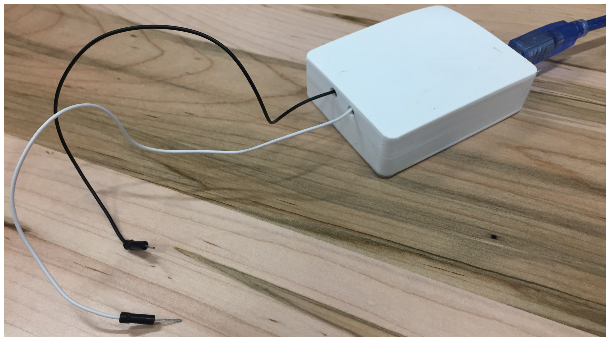
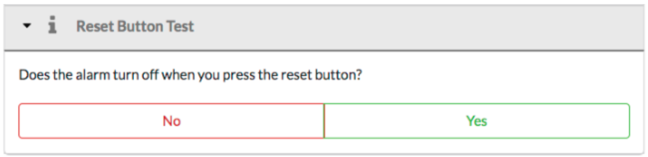


Figure 6. A secondary Arduino Uno was used as a circuit probe device (e.g., for physical tests with manual setup).

$S_{M,M}$, $P_{M,M}$: Confirmation Tests: For a *Confirmation Test,* users manually execute a specified action, and then report the outcome using a multiple choice or yes/no prompt (Figure 7). The action the user is instructed to perform could be in the software or hardware components of the project.

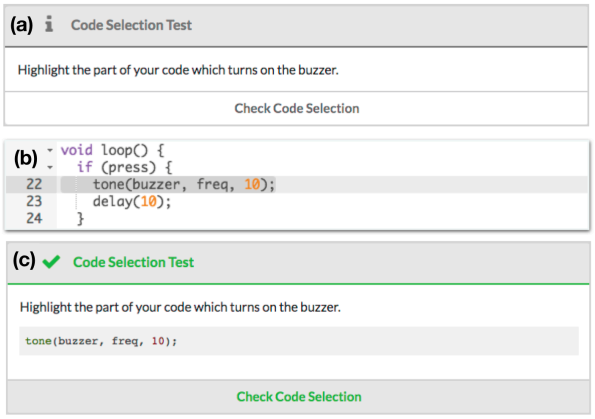


Does the alarm turn off when you press the reset button?

Figure 7. A confirmation test asks the users to manually report the result of a specified action.

$S_{M,A}$: Compile, Upload, and Code Selection Tests. The *Compile* and *Upload* tests are used to test the user's code. When run, the test compiles or uploads the user's code to the project's Arduino board. The system automatically detects if the operation is successful. If these tests fail, the user is shown a sanitized error message from the compiler, along with any custom messages that the author has added to the test (e.g., to suggest troubleshooting steps).

In the *Code Selection Test,* the user is prompted to highlight a portion of their code. For example, one such test in our tutorial asks the user to "*Highlight the part of your code which turns on the buzzer",* which would only pass when the user highlights code containing the string `tone(buzzer, freq, 10)` (Figure 8).



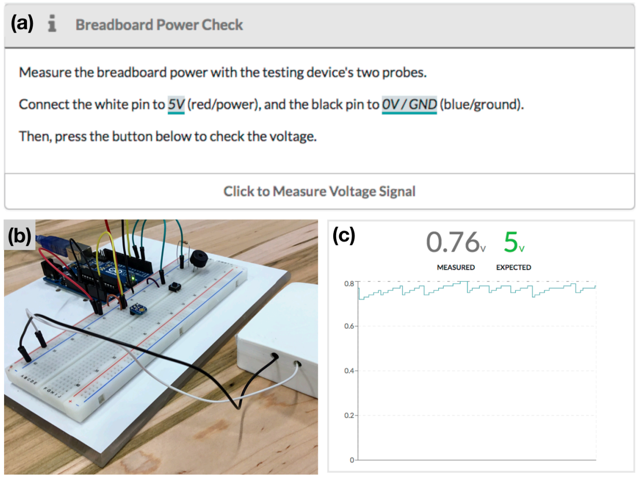
Highlight the part of your code which turns on the buzzer.

```
void loop() {
  if (press) {
    tone(buzzer, freq, 10);
    delay(10);
  }
```

Highlight the part of your code which turns on the buzzer.
```
tone(buzzer, freq, 10);
```

Figure 8. A Code Selection Test. (a) The user is prompted to highlight a section of code; (b) The user highlights the code in the editor; (c) The code is displayed in the test pane and the user clicks to check their response.

$P_{M,A}$: Voltage, Frequency, & Continuity Tests. For these tests, the user takes voltage, frequency, or continuity measurements using the circuit probe device. The system tests the readings from the probe device against a reference value (or a value range) that has been pre-authored by the tutorial author. When the test is started, code is sent to the circuit probe, configuring it to sense the relevant attribute. The measurement is shown graphically in real-time, and the test passes when the expected value is achieved (Figure 9).



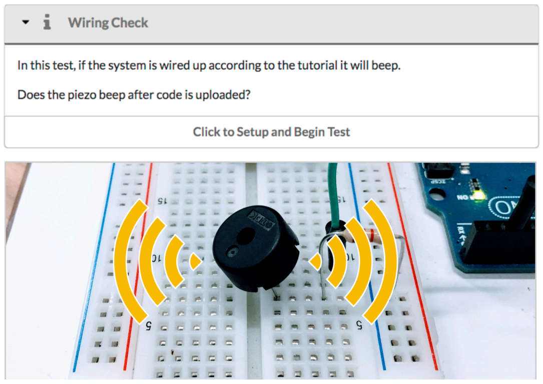
Measure the breadboard power with the testing device's two probes.
Connect the white pin to 5V (red/power), and the black pin to 0V / GND (blue/ground).
Then, press the button below to check the voltage.

Figure 9. For a voltage test, (a) the user is prompted to attach the circuit probe device to the circuit (b), and run the test. (c) Measurements are then displayed, and the test passes when the expected value is read.

$S_{A,M}$: Manual Variable Test. When the user initiates this test, their code is automatically instrumented to monitor select variable data at runtime, and the instrumented code is

uploaded to the project's Arduino board. The user is shown a visualization of the data trace and asked to confirm that it matches the expected pattern, which is pre-specified by the tutorial author.

$P_{A,M}$: *Auto-Upload Test.* This test is used to assess whether a hardware or circuit component of the system has been properly built, before the user writes associated code for that part of the project. When the user initiates this test, the system uploads pre-authored code to the project board. The user then manually verifies that the component performs as expected. For example, the user might verify that a Piezo buzzer makes the expected sound when the test is executed (Figure 10). This allows the user to confirm that a physical component is working as expected, before writing code or integrating additional parts.


Wiring Check

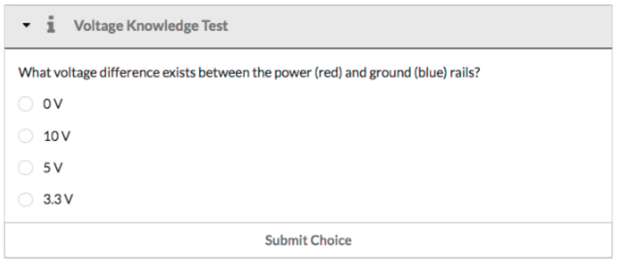
In this test, if the system is wired up according to the tutorial it will beep.

Does the piezo beep after code is uploaded?

Figure 10. The Auto-Upload test uses pre-authored code to validate hardware components of the project.

$S_{A,A}$: *Auto-Variable Test.* As in the manual variable test, this test instruments the user's code to track variable data at runtime. The tutorial system then monitors the instrumented variables to confirm that they pass pre-authored test cases. These tests can track multiple variables, record which line number variables are changed on, and can use both strict equality and range inclusion validation techniques.

$P_{A,A}$: *Auto-Sense Test.* In this test, the system uploads code to read measurements from the project board's pins. For example, this could be used to test that a sensor has been properly connected to the project board. The system performs configuration, measurement, and verification automatically. Based on any deviations from expected values, the system can provide feedback on why the test failed, or troubleshooting tips from the tutorial author.


Voltage Knowledge Test

What voltage difference exists between the power (red) and ground (blue) rails?

- 0 V
- 10 V
- 5 V
- 3.3 V

Figure 11. Knowledge tests are used to test the user's comprehension of concepts within a step.

$S_{N,M}$, $P_{N,M}$: *Knowledge Tests:* In the *Knowledge Test,* the user answers knowledge-based questions related to the tutorial content (Figure 11). If a question is answered incorrectly, users are presented with corrective guidance pre-authored by the tutorial author.

Tutorial and Test Authoring

In the current version of ElectroTutor, tutorial content and tests are manually authored in human-readable YAML-formatted configuration files. Each test is represented by an entry specifying the type of test (e.g., voltage threshold test), instructional text to show to the user (to specify the intent of the test and any necessary setup instructions), and a set of test-specific parameters (e.g., the threshold voltage). This corresponds to the simple design of the tests in the interface, which include a title, a short instruction to the user, and the interface for running the test. Finally, an optional "on error" message can be included for tests, which is shown to the user if the test fails (e.g., to provide suggestions for debugging). Though we did not develop a graphical interface for creating tests, it would be straightforward to do so, because each type of test follows a simple structured format with a small number of parameters.

Tutorial content is also represented in structured configuration files, with each step specified in the Markdown format, which is rendered into HTML for the interface.

IMPLEMENTATION

The overall system architecture for ElectroTutor is shown in Figure 12. The client-side user interface was built using JavaScript and React.js, server-side data management was done with Ruby on Rails, and serial port communication with the attached Arduino devices are processed with the SerialPortJSONServer tool[1].

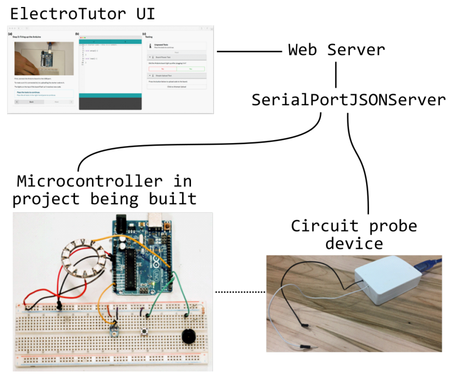


Figure 12. ElectroTutor system architecture.

To support tests that observe the runtime values of variables on the project's Arduino board, we developed a custom instrumentation tool that parses and modifies the user's code before it is uploaded to the board. The parser detects variable definitions and assignment statements for variables of interest for a test, and inserts *Serial.print()* statements that include the variable being assigned, the value being assigned, and the line number on which the assignment took place. These logging statements are transmitted over the serial connection and read by the tutorial system during runtime.

[1] https://github.com/johnlauer/serial-port-json-server

The instrumentation approach above poses a runtime performance overhead. To reduce this, a variable's identifier is mapped to a numeric index before being sent over the serial port, which is then mapped back to the variable's name by the tutorial system, reducing the amount of data that needs to be sent over the serial connection.

For tests that make physical measurements of the circuit being built, we use a secondary Arduino Uno board enclosed in a simple 3D printed case, with two probes exposed (Figure 6). Voltage measurements are performed using the *analogRead()* function from the Arduino internal library. This circuit probe device was inspired by the physical interface of a multimeter, which can perform a wide range of measurements with two simple probes. The circuit probe is flashed with different code based on the test the user runs, allowing it to serve as a voltmeter, frequency analyzer, and electrical continuity checker.

EVALUATION

To evaluate the effectiveness of the interactive tests implemented in ElectroTutor, and to gain initial insights into the test-driven tutorial approach more generally, we conducted a user study comparing our prototype system to a comparable tutorial without tests.

Study Design

The study followed a between-subjects design, with half of participants in an experimental condition, and the other half in a control condition. All participants performed a tutorial for the same physical computing project, but the test-driven features were only available in the experimental condition. Specifically, in the control condition, tests were not included for the tutorial steps, and the participant was free to progress through the tutorial steps as they wished. In the experimental condition, participants were presented with tests for many of the steps, and were restricted from progressing to the next step until all the tests for that step had been passed.

We designed the physical computing project and tutorial to ensure that participants in both conditions were exposed to the same instructional content. While tests may force experimental participants to consider one aspect of the tutorial, they did not introduce any additional information beyond that available to the control participants.

Study Procedure

The study began with an overview of the ElectroTutor interface. Participants in the experimental condition were also introduced to the circuit probe device, and given a simple demo of an interactive test. The participant was then given a maximum of 45 minutes to complete the tutorial.

Tutorial Project. The tutorial project was to build a light-sensitive alarm clock system with a reset trigger (Figure 13). The project included both electronics and programming components. The tutorial instructions and associated tests for each step are included in our supplementary materials.

Post-study questionnaire. Following the tutorial portion of the study, participants answered a post-study questionnaire which included Likert-scale ratings of their confidence working through the electronics and programming parts of the tutorial, what they learned from the tutorial, and how they liked using the system. Participants in the experimental condition also answered additional questions on the interactive tests. To evaluate whether there was a difference in how much participants learned in the two conditions, we included knowledge-based questions about the electronics and programming aspects of the tutorial, from a selection of concepts that were covered in the tutorial content.

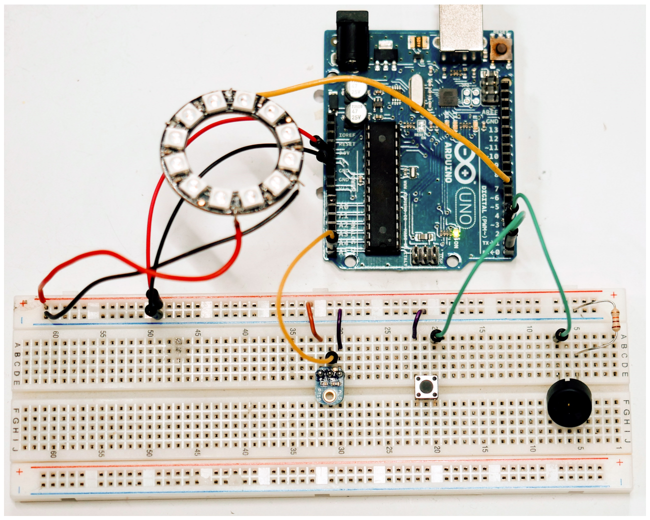


Figure 13. The completed Light Sensitive Alarm, consisting of an LED ring, light sensor, buzzer, and a reset button.

Participants

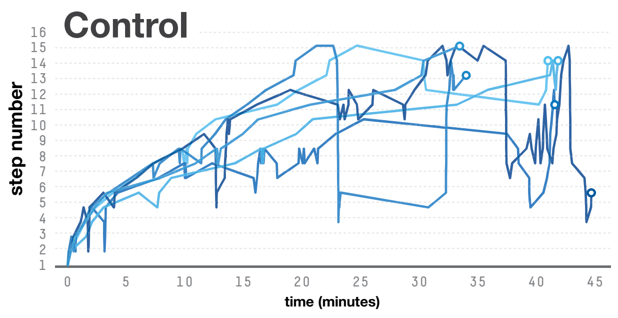
We recruited 12 participants (10 male, 2 female, ages 20-54, mean 34, SD 11) through an email to employees at a large software company. Participants were screened to ensure that they had minimal experience with physical computing, and were given a $25 gift card as thanks for participating.

RESULTS

Tutorial completion and task times

Overall, 5/6 participants completed the tutorial in the experimental condition, versus 3/6 in the control condition. In terms of timing, participants in the experimental condition had faster tutorial completion times on average (Experimental: mean 39.7 minutes (SD 6.8), Control: mean 41.5 (SD 4.3)). A t-test did not show this difference to be significant ($p=0.59$). However, it is worth noting that these times include the time taken to run tests for participants in the experimental condition, which may suggest that the tests enabled participants to spend less time working through the tutorial instructions.

In terms of how participants progressed through the tutorial, Figure 14 shows a timeline of participants' navigation through the tutorial steps. We can see that participants in the control condition exhibited much more backtracking to previous steps, as compared to the experimental condition where participants proceeded linearly through the steps. Analyzing the number of instances where participants backtracked to a previous step, we found an average of 18.8 (SD 19.3) instances in the control condition, versus an average of 0.2 (SD 0.4) for the experimental condition. A two-sample t-test found this difference to be significant (t(10)=-2.37, p<.05). Note that the participants in the experimental condition were not restricted from backtracking to previous steps, and there was no penalty for doing so – they simply chose not to.



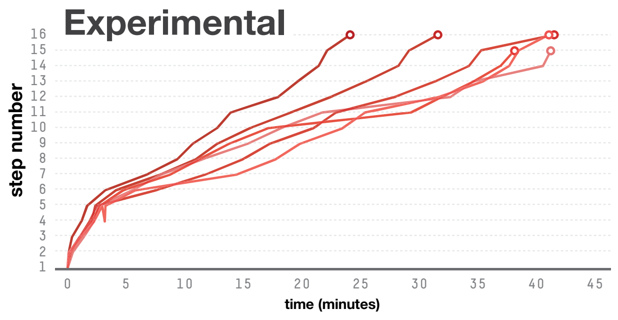


Figure 14. Timeline of participants' step navigation over the study session (blue=control, red=experimental).

From observing participants in the control condition, and discussions with them at the end of the study, backtracking was often used in response to discovering that part of the project from a previous step was not working. The participant would then backtrack to the instructions for the broken part, and engage in troubleshooting to try and get it working. This provides validation for the inclusion of tests as a means of verifying that each step is completed correctly before allowing the user to move on.

Knowledge transfer and learning

In the post-study questionnaire, we included a set of four knowledge tests based on the tutorial content. On average, participants in the experimental condition answered 2.8/4 (SD 0.8) of these questions correctly, versus 2.0/4 (SD 1.0) for the control condition. A t-test did not show this difference to be significant (p=0.16).

User confidence and subjective assessments

To understand how the addition of tests affected participants' confidence while completing the tutorial, we analyzed their responses to questions in the post-study questionnaire on their confidence with the electronics and programming portions of the tutorial (Figure 15). Overall, confidence appears to be slightly higher for participants in the experimental condition.

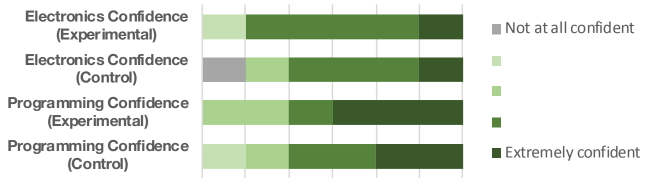


Figure 15. Participants' responses to the confidence questions in the post-study questionnaire.

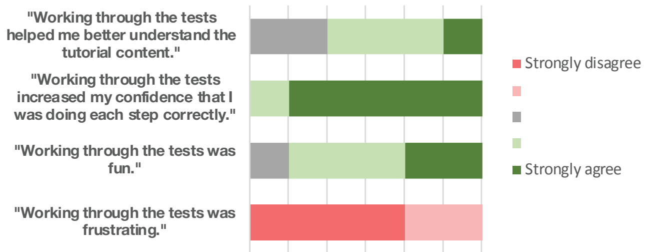


Figure 16. Subjective ratings provided by participants in the experimental condition.

We asked participants in the experimental condition to answer several additional questions on the effect of the interactive tests on the tutorial experience (Figure 16). Overall, the results are encouraging, with 5/6 participants indicating strong agreement that the tests increased their confidence that they were doing each step correctly. Participants also provided favorable ratings for the tests being fun, and helping them to understand the tutorial content. Finally, all participants disagreed with the statement "working through the tests was frustrating".

The above ratings were consistent with participants' feedback on the tutorial system. In particular, one participant in the experimental condition cited the inclusion of tests throughout the tutorial as helping his confidence:

The tests were definitely helpful. I especially liked the fact that tests were done in a granular manner at each step along the tutorial, so that I felt confident throughout the tutorial.

Conversely, a participant in the control condition expressed that while following the instructions she was uncertain about whether she was doing things correctly:

I struggled with the hardware part, I was not that familiar with it. It's kind of like, I am following the instructions but I am just not sure if I am doing it the right way.

Overall, these findings suggest that the inclusion of tests can increase participants' confidence while working on a tutorial.

DISCUSSION AND FUTURE WORK

The results from our study are promising – users of ElectroTutor appreciated the added tests, and indicated that the system increased their confidence that they were completing the tutorial steps correctly. Our analysis also indicates that the test-driven approach enabled users to diagnose problems immediately, preventing situations where they may have to backtrack to try and understand why a problem is occurring and troubleshoot a solution.

In this section, we discuss areas for further developing the test-driven physical computing tutorials approach, and limitations of the current work.

Tools for Tutorial Authors

This paper has looked at test-driven tutorials from the perspective of users, with the tutorial task and its associated tests hard-coded into the system. An important area for future work is to consider how to support authors in creating test-driven tutorials.

As mentioned earlier in the paper, authoring of tests could be achieved through a simple interface for selecting the class of test to be added (e.g. voltage test), specifying parameter values (e.g., threshold of 5V), and adding supplementary photos (e.g., to indicate where to place testing probes).

In addition to manual authoring, it is worth considering how the authoring of tests and tutorials could be automated. For example, text-recognition algorithms could analyze tutorial content and code to suggest tests that might be appropriate for a given step. For physical computing tutorials, one could also imagine a fully by-demonstration approach (similar to what has been developed for photo manipulation tutorials [8, 14]), in which the author builds and programs a circuit in an instrumented Arduino development environment and circuit simulator, and a corresponding test-driven tutorial is automatically created. For the code component of tutorials, techniques for working with multi-stage code examples (e.g. [12]) could be adapted, and linked to the generated tests.

Tests could also be generated through crowdsourcing [20] or learner sourcing [13, 19, 37] techniques. For example, each user who completes a tutorial could be asked to suggest their own tests, and the collection of these user-elicited tests could be aggregated and refined over time.

In addition to supporting authors in creating tutorials, tools could be developed to support an author's awareness of how a tutorial is being used, and the challenges that are being encountered by its users. For example, the results of tests from multiple users could be aggregated and reported in a tutorial analytics dashboard. The author could see which tests frequently fail, which may indicate unclear instructions, or a need for additional tests. Likewise, tests which never fail may be unnecessary, and could potentially be removed without diminishing the tutorial experience.

Extending the "Discount Sensing" Approach

Adding more sophisticated hardware sensing is another interesting direction for future work. While still falling well below the cost of logic analyzers and oscilloscopes, a higher resolution testing device could be constructed with an Arduino Mega board. Additional techniques could leverage computer vision to visually verify hardware configurations. For example, a webcam could be used to test expected behaviors, such as a light blinking or a servo motor moving.

There is also an opportunity to use new forms of sensor-enabled tools [21, 32, 36] to perform measurement tests in a semi-automated way. Past work has investigated the use of recording and replaying of electronic traces to facilitate the testing and development of electronics designs [17, 34]. With hardware platforms that have onboard DACs (such as the Arduino Due or Zero), the system could replay recorded traces from the tutorial author as inputs, and then verify the response of the system being built to that trace signal, highlighting and offering guidance to the user based on the differences between the expected and actual readings.

Hardware Simulations

Even with more sophisticated sensing, it will be difficult for a tutorial system to get a complete picture of what is going on in a physical circuit. An interesting extension of this work would be to integrate test-driven tutorials into a simulation-based circuit design platform, where the tutorial and tests can fully measure the state and behavior of the circuit being built. Hardware simulations could also be used to build a database of symptoms associated with common wiring errors, to help with error diagnosis and providing recovery instructions.

Generalizing to Other Domains, More Complex Projects

Test-driven tutorials are particularly relevant for physical computing projects, given the challenge of instrumenting a circuit in the process of being built. However, we believe that there are additional benefits to the test-driven tutorial approach, such as user engagement and knowledge retention, that would make the approach appropriate for other tutorial domains as well. For example, test-driven tutorials could be created for complex software applications, or software development environments. In an image editor, the user could manually take a screenshot of the layer palette after a step, to confirm that they have set up the image layers appropriately. In an IDE, a user could run their code in debug mode, and answer test questions on the value of a watched variable. Ideally, engagement with the tests could help users to learn the skills of debugging unexpected behavior in these domains.

We see potential for extending our approach to more complex tutorials as well. For large and complex physical computing tutorials, it may be valuable to add higher-level visualizations of the entire project, to give the user a view of the set of all tests, how they relate to the project and to one another, and to visualize the user's progress as they work through the tutorial content.

Limitations

There are several limitations to our system and user study that are important to acknowledge, and that point to key areas for extending this research.

First, in this work, the resolution, performance, and timing of hardware sensing were not key areas of emphasis. These capabilities of our system are limited by the capabilities of the Arduino Uno board we used as the circuit probe device. The resolution of the analog-digital convert on the Arduino Uno is 10 bits, which supports a resolution of 4.882 mV with a 5V power source, which may not be sufficient for testing precise electronic systems. This implementation decision also precludes taking accurate measurements outside of the

0-5V reference range that the Uno supports, and affords only the standard voltage protection offered by the board when taking measurements. Integrating more sophisticated hardware sensing capabilities into this type of tutorial system is an interesting area for future work.

Second, the approach we used to instrument the software running on the project board imposes a slowdown, due to the overhead of sending additional data over the serial connection. This is unlikely to be a problem for many physical computing projects that involve human interaction, where human input speed is the main bottleneck, but could limit the approach's usefulness for timing-critical systems.

Third, the collection of test types that we implemented in ElectroTutor was guided by our interest in covering all ten of the areas of our test taxonomy that are not covered by past work on reactive tutorial systems, and thus cannot be considered a comprehensive set of tests for the physical computing domain. An interesting area for future work would be to investigate more sophisticated types of tests for test-driven physical computing tutorial (e.g., tests that compare measurements against a pre-recorded signal trace, or check multiple conditions simultaneously). A key challenge in implementing new and more complex test types is keeping them easy to use and understand by the users of the tutorial.

Finally, the sample size for our study was small. We elected to conduct a small-scale study to gain initial insights on the approach, but more comprehensive studies are needed to better understand the impact of this kind of tutorial on learners, and its longer-term effects on learning and skill development. It would also be valuable to test this type of tutorial system with a more diverse group of users, including participants who have more experience on the hardware side of physical computing, but less software and programming experience.

CONCLUSION

We have presented test-driven physical computing tutorials, a new type of tutorial system that integrates interactive tests into the tutorial experience. In addition to implementing this idea in ElectroTutor, we have described design considerations for this kind of system, and presented a taxonomy of different types of tests that could guide the development of future test-driven tutorial systems. Our study results indicate the promise and potential of test-driven tutorials, demonstrating that they can help users avoid backtracking to previous tutorial steps, and demonstrating that the interactive tests were appreciated by users. While our work focuses on physical computing tutorials, we have discussed how the test-driven tutorial approach could be applied in other domains, and outlined how our work can be extended, such as by developing authoring tools or integrating more sophisticated sensing techniques.

REFERENCES

1. Fraser Anderson, Tovi Grossman, and George Fitzmaurice. 2017. Trigger-Action-Circuits: Leveraging Generative Design to Enable Novices to Design and Build Circuitry. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology* (UIST '17), 331–342. https://doi.org/10.1145/3126594.3126637
2. Kent Beck. 2002. *Test Driven Development: By Example*. Addison-Wesley Professional, Boston.
3. Lawrence Bergman, Vittorio Castelli, Tessa Lau, and Daniel Oblinger. 2005. DocWizards: a system for authoring follow-me documentation wizards. In *Proceedings of the 18th annual ACM symposium on User interface software and technology* (UIST '05), 191–200. https://doi.org/10.1145/1095034.1095067
4. 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), 3485–3497. https://doi.org/10.1145/2858036.2858533
5. John M. Carroll. 1990. *The Nurnberg funnel: designing minimalist instruction for practical computer skill*. MIT Press.
6. John M. Carroll and Mary Beth Rosson. 1987. Paradox of the active user. In *Interfacing Thought: Cognitive Aspects of Human-Computer Interaction*. MIT Press, 80–111. Retrieved March 5, 2012 from http://dl.acm.org/citation.cfm?id=28446.28451
7. Michelene T. H. Chi, Nicholas De Leeuw, Mei-Hung Chiu, and Christian Lavancher. 1994. Eliciting self-explanations improves understanding. *Cognitive Science* 18, 3: 439–477. https://doi.org/10.1016/0364-0213(94)90016-7
8. Pei-Yu Chi, Sally Ahn, Amanda Ren, Mira Dontcheva, Wilmot Li, and Björn Hartmann. 2012. MixT: Automatic generation of step-by-step mixed media tutorials. In *Proceedings of the 25th annual ACM symposium on User interface software and technology* (UIST '12), 93–102.
9. Daniel Drew, Julie L. Newcomb, William McGrath, Filip Maksimovic, David Mellis, and Björn Hartmann. 2016. The Toastboard: Ubiquitous Instrumentation and Automated Checking of Breadboarded Circuits. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology* (UIST '16), 677–686. https://doi.org/10.1145/2984511.2984566
10. Jennifer Fernquist, Tovi Grossman, and George Fitzmaurice. 2011. Sketch-sketch revolution: an engaging tutorial system for guided sketching and application learning. In *Proceedings of the 24th annual ACM symposium on User interface software and technology* (UIST '11), 373–382. https://doi.org/10.1145/2047196.2047245

11. Vicki S. Gier and David S. Kreiner. 2009. Incorporating Active Learning with PowerPoint-Based Lectures Using Content-Based Questions. *Teaching of Psychology* 36, 2: 134–139. https://doi.org/10.1080/00986280902739792

12. Shiry Ginosar, Luis Fernando De Pombo, Maneesh Agrawala, and Bjorn Hartmann. 2013. Authoring Multi-stage Code Examples with Editable Code Histories. In *Proceedings of the 26th Annual ACM Symposium on User Interface Software and Technology* (UIST '13), 485–494. https://doi.org/10.1145/2501988.2502053

13. Elena L. Glassman, Aaron Lin, Carrie J. Cai, and Robert C. Miller. 2016. Learnersourcing Personalized Hints. In *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing* (CSCW '16), 1626–1636. https://doi.org/10.1145/2818048.2820011

14. Floraine Grabler, Maneesh Agrawala, Wilmot Li, Mira Dontcheva, and Takeo Igarashi. 2009. Generating photo manipulation tutorials by demonstration. In *ACM SIGGRAPH 2009 papers* (SIGGRAPH '09), 66:1–66:9. https://doi.org/10.1145/1576246.1531372

15. Tovi Grossman and George Fitzmaurice. 2010. ToolClips: An investigation of contextual video assistance for functionality understanding. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '10), 1515–1524. https://doi.org/10.1145/1753326.1753552

16. Björn Hartmann, Leith Abdulla, Manas Mittal, and Scott R. Klemmer. 2007. Authoring Sensor-based Interactions by Demonstration with Direct Manipulation and Pattern Recognition. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '07), 145–154. https://doi.org/10.1145/1240624.1240646

17. Björn Hartmann, Scott R. Klemmer, Michael Bernstein, Leith Abdulla, Brandon Burr, Avi Robinson-Mosher, and Jennifer Gee. 2006. Reflective Physical Prototyping Through Integrated Design, Test, and Analysis. In *Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology* (UIST '06), 299–308. https://doi.org/10.1145/1166253.1166300

18. Caitlin Kelleher and Randy Pausch. 2005. Stencils-based tutorials: Design and evaluation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '05), 541–550. https://doi.org/10.1145/1054972.1055047

19. Juho Kim. 2015. Learnersourcing: Improving Learning with Collective Learner Activity. PhD thesis, Massachusetts Institute of Technology.

20. Juho Kim, Phu Tran Nguyen, Sarah Weir, Philip J. Guo, Robert C. Miller, and Krzysztof Z. Gajos. 2014. Crowdsourcing Step-by-step Information Extraction to Enhance Existing How-to Videos. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '14), 4017–4026. https://doi.org/10.1145/2556288.2556986

21. Jarrod Knibbe, Tovi Grossman, and George Fitzmaurice. 2015. Smart Makerspace: An Immersive Instructional Space for Physical Tasks. In *Proceedings of the 2015 International Conference on Interactive Tabletops & Surfaces* (ITS '15), 83–92. https://doi.org/10.1145/2817721.2817741

22. Benjamin Lafreniere, Andrea Bunt, and Michael Terry. 2014. Task-centric interfaces for feature-rich software. In *Proceedings of the 26th Australian Computer-Human Interaction Conference* (OZCHI '14), 49–58.

23. Benjamin Lafreniere, Tovi Grossman, Fraser Anderson, Justin Matejka, Heather Kerrick, Danil Nagy, Lauren Vasey, Evan Atherton, Nicholas Beirne, Marcelo H. Coelho, Nicholas Cote, Steven Li, Andy Nogueira, Long Nguyen, Tobias Schwinn, James Stoddart, David Thomasson, Ray Wang, Thomas White, David Benjamin, Maurice Conti, Achim Menges, and George Fitzmaurice. 2016. Crowdsourced Fabrication. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology* (UIST '16), 15–28. https://doi.org/10.1145/2984511.2984553

24. Benjamin Lafreniere, Tovi Grossman, and George Fitzmaurice. 2013. Community enhanced tutorials: Improving tutorials with multiple demonstrations. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '13), 1779–1788. https://doi.org/10.1145/2470654.2466235

25. Wei Li, Tovi Grossman, and George Fitzmaurice. 2012. GamiCAD: A Gamified Tutorial System for First Time Autocad Users. In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology* (UIST '12), 103–112. https://doi.org/10.1145/2380116.2380131

26. Will McGrath, Daniel Drew, Jeremy Warner, Majeed Kazemitabaar, Mitchell Karchemsky, David Mellis, and Björn Hartmann. 2017. Bifröst: Visualizing and Checking Behavior of Embedded Systems Across Hardware and Software. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology* (UIST '17), 299–310. https://doi.org/10.1145/3126594.3126658

27. David A. Mellis, Leah Buechley, Mitchel Resnick, and Björn Hartmann. 2016. Engaging Amateurs in the Design, Fabrication, and Assembly of Electronic Devices. In *Proceedings of the 2016 ACM Conference on Designing Interactive Systems* (DIS '16), 1270–1281. https://doi.org/10.1145/2901790.2901833

28. Alok Mysore and Philip J. Guo. 2017. Torta: Generating Mixed-Media GUI and Command-Line

App Tutorials Using Operating-System-Wide Activity Tracing. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology* (UIST '17), 703–714. https://doi.org/10.1145/3126594.3126628

29. Susan Palmiter, Jay Elkerton, and Patricia Baggett. 1991. Animated demonstrations vs written instructions for learning procedural tasks: a preliminary investigation. *Int. J. Man-Mach. Stud.* 34, 5: 687–701. https://doi.org/10.1016/0020-7373(91)90019-4
30. Suporn Pongnumkul, Mira Dontcheva, Wilmot Li, Jue Wang, Lubomir Bourdev, Shai Avidan, and Michael F. Cohen. 2011. Pause-and-play: automatically linking screencast video tutorials with applications. In *Proceedings of the 24th annual ACM symposium on User interface software and technology* (UIST '11), 135–144. https://doi.org/10.1145/2047196.2047213
31. Vidya Ramesh, Charlie Hsu, Maneesh Agrawala, and Björn Hartmann. 2011. ShowMeHow: translating user interface instructions between applications. In *Proceedings of the 24th annual ACM symposium on User interface software and technology* (UIST '11), 127–134. https://doi.org/10.1145/2047196.2047212
32. Eldon Schoop, Michelle Nguyen, Daniel Lim, Valkyrie Savage, Sean Follmer, and Björn Hartmann. 2016. Drill Sergeant: Supporting Physical Construction Projects Through an Ecosystem of Augmented Tools. In *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems* (CHI EA '16), 1607–1614. https://doi.org/10.1145/2851581.2892429
33. Hyungyu Shin, Eun-Young Ko, Joseph Jay Williams, and Juho Kim. 2018. Understanding the Effect of In-Video Prompting on Learners and Instructors. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '18), 10 pages.
34. Evan Strasnick, Maneesh Agrawala, and Sean Follmer. 2017. Scanalog: Interactive Design and Debugging of Analog Circuits with Programmable Hardware. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology* (UIST '17), 321–330. https://doi.org/10.1145/3126594.3126618
35. Robin Tuck and Dan R. Olsen. 1990. Help by guided tasks: utilizing UIMS knowledge. In *Proceedings of the SIGCHI conference on Human factors in computing systems: Empowering people* (CHI '90), 71–78. https://doi.org/10.1145/97243.97254
36. Christian Weichel, Jason Alexander, Abhijit Karnik, and Hans Gellersen. 2015. SPATA: Spatio-Tangible Tools for Fabrication-Aware Design. In *Proceedings of the Ninth International Conference on Tangible, Embedded, and Embodied Interaction* (TEI '15), 189–196. https://doi.org/10.1145/2677199.2680576
37. Joseph Jay Williams, Juho Kim, Anna Rafferty, Samuel Maldonado, Krzysztof Z. Gajos, Walter S. Lasecki, and Neil Heffernan. 2016. AXIS: Generating Explanations at Scale with Learnersourcing and Machine Learning. In *Proceedings of the Third (2016) ACM Conference on Learning @ Scale* (L@S '16), 379–388. https://doi.org/10.1145/2876034.2876042
38. Te-Yen Wu, Bryan Wang, Jiun-Yu Lee, Hao-Ping Shen, Yu-Chian Wu, Yu-An Chen, Pin-Sung Ku, Ming-Wei Hsu, Yu-Chih Lin, and Mike Y. Chen. 2017. CircuitSense: Automatic Sensing of Physical Circuits and Generation of Virtual Circuits to Support Software Tools. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology* (UIST '17), 311–319. https://doi.org/10.1145/3126594.3126634
39. Codecademy - learn to code, interactively, for free. *Codecademy*. Retrieved April 5, 2018 from https://www.codecademy.com/
40. Khan Academy. *Khan Academy*. Retrieved April 5, 2018 from http://www.khanacademy.org