AutoFritz: Autocomplete for Prototyping Virtual Breadboard Circuits

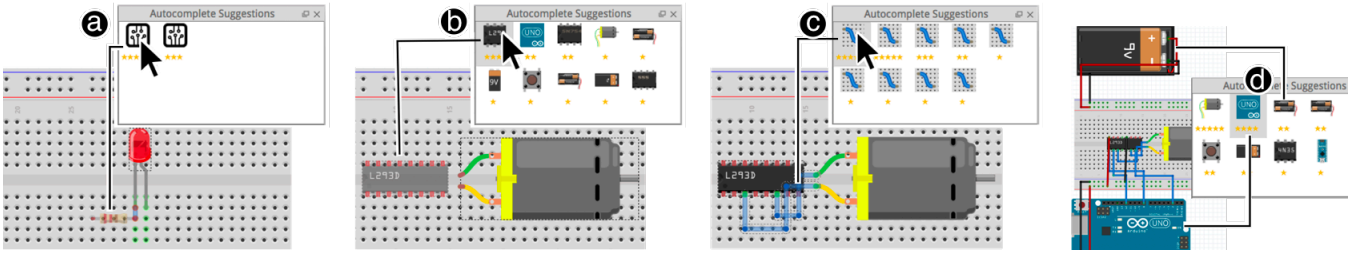

Figure 1. Using AutoFritz to create virtual breadboard circuits. (a) A user inserts a LED in a breadboard, and AutoFritz gives two suggestions that completes the LED into a function module with the top ranked one (a resistor) shown in a semi-transparent overlay. (b) In another example, after the user inserts a DC motor, AutoFritz provides a list of suggestions that are commonly used with the DC motor with stars indicating the popularity of the choice among the maker community. (c) Upon the user accepting the H-bridge, AutoFritz suggests several choices for wiring connections, and finally (d) an Arduino board and Battery to complete the circuit.

ABSTRACT

We propose autocomplete for the design and development of virtual breadboard circuits using software prototyping tools. With our system, a user inserts a component into the virtual breadboard, and it automatically provides a user with a list of suggested components. These suggestions complete or extend the electronic functionality of the inserted component to save the user's time and reduce circuit error. To demonstrate the effectiveness of autocomplete, we implemented our system on Fritzing, a popular open source breadboard circuit prototyping software, used by novice makers. Our autocomplete suggestions were implemented based upon schematics from datasheets for standard components, as well as how components are used together from over 4000 circuit projects from the Fritzing community. We report the results of a controlled study with 16 participants, evaluating the effectiveness of autocomplete in the creation of virtual breadboard circuits, and conclude by sharing insights and directions for future research.

Author Keywords

Circuit design, breadboard, autocomplete.

ACM Classification Keywords

H.5.2. [Information interfaces and presentation]: User Interfaces – Prototyping.

INTRODUCTION

Designing and documenting breadboard circuits using software prototyping tools (e.g., Fritzing [33], VBB [10], and 123D Circuits [14]) has become a common practice in the maker community and beyond, with these tools being heavily used by those without a strong electronics background. However, much of the existing software requires the circuit components to be manually added and connected by user, which can be error prone and time-consuming. Common hardware errors, such as missing components (e.g., not using a resistor with a LED) or miswiring [38], also occurs in the virtual world, and can lead to significant time and efforts from the users to debug.

In this paper, we introduce the concept of *autocomplete*, commonly seen in web search, text editing, and programming, to the construction of virtual breadboard circuits. With autocomplete, software provides the user with a list of suggested components based on the one that is newly inserted in a virtual breadboard by a user (Figure 1). The suggested components complete or extend the functionality of the inserted component to save the user's time and reduce circuit error. For example, when a user inserts an LED into the virtual breadboard, the system suggests that a resistor can be used, whose absence may cause damage to the LED or an Arduino board. The user can ignore the suggestion or accept it, in which case, a resistor will be placed and connected automatically to the LED on the virtual breadboard. With this feature, it becomes less likely for a novice user to miss the resistor. A user does not have to spend time on searching and reading the datasheet schematic of a component to learn how it should be used properly with basic components like resistors or inductors, to function properly.

In this example, the completed LED circuit is an electrically functional *module,* defined by an I/O or IC component with a supporting component (e.g., resistor). Once a module is created, the system suggests a list of other modules often used by others from the maker community for extended functionality. For example, a possible suggestion for the LED module is the button module to control the LED. Aside from the modules, the system can also suggest a battery or Arduino board to complete the circuit. If the suggestion is accepted by the user, wire connections between the two modules are also suggested by the system.

In submission to CHI 2019

To demonstrate the proposed autocomplete features, we implemented them in Fritzing, a popular open source breadboard circuit prototyping software [5], used by novice makers. Our system (called AutoFritz) supports autocompleting 58 common electronic components (I/O and IC) into functional modules based on the components' datasheet schematics. Our system can also provide module-to-module suggestions based on how commonly the modules are used together by the Fritzing community [4]. We evaluate the effectiveness of our system with 16 expert and novice users in a controlled lab evaluation, where participants were asked to use AutoFritz to create virtual breadboard circuits in two levels of complexity (low vs high). We found that with AutoFritz, participants were able to accomplish the tasks with significant less errors, mental effort, and frustration. They were also more confidence about their answers. We also discuss insights gained from the study.

The primary contributions of this paper are: 1) autocomplete features in prototyping virtual breadboard circuits; 2) a software tool that demonstrates the utility of these features; and 3) the results of a controlled experiment, measuring the effectiveness of the proposed autocomplete features in prototyping virtual breadboard circuits.

RELATED WORK

In this section, we discuss existing research on autocomplete in various application domains, as well as efforts made on circuit design, prototyping, and debugging.

Autocomplete in Varying Application Domains

Autocomplete has been widely used in web search, text editing, and programming to improve the experience of users by speeding up text entry and helping to avoid spelling mistakes [11, 16, 23, 24, 39, 46]. It can also help a user discover relevant search terms, build confidence with an unfamiliar search topic, and formulate better queries [46]. A typical technique used in autocomplete is to provide the user a list of suggested words according to their input and presented them in an order based on certain criteria (e.g., word frequency). This is similar to modern recommender systems (e.g., [27, 36]) in the sense that personalized suggestions are provided to users to improve task efficiency and user experience. Researchers have found a strong bias towards examining and using top-ranked suggestions, which effects task engagement and search effectiveness [29]. Work in this field has motivated innovations in query suggestions and ranking though techniques considering contextual information [41] and input semantic [30]. A survey in query autocomplete can be found in [20].

Aside from the text-based applications, autocomplete has also been used in gestural input [17, 18], sketch tools [31, 40, 43, 49], photo editing tools [26, 35], and information visualization tools [34]. For example, SimpleFlow [18] provide suggestions of standard gestures from a database to the user, based on the user's input trajectory. Tsang, et al.'s sketch tool [43] can suggest a geometry to the user by approximately matching the user's input strokes against a database of pre-made 3D models. Hays and Efros's photo editing tool [26] can complete a missing region specified in an input image with matching scenes from a database of similar photos. VisComplete [34] can help the user construct visualization pipelines by automatically suggesting construction modules based on the database of previously created ones. These works demonstrate that autocomplete is an effect technique in a wide range of applications.

Circuit Design Tools

Software tools have been developed for professional users to design and document electric circuits on PCBs efficiently [12, 15], but they are hard to use by novice users. Breadboard circuit prototyping software [10, 33] has been well received by the novice community. Fritzing [33], for example, offers a simple interface for novice users to create virtual breadboard circuits using common electronic components. It also provides an online environment for the community to discuss and share their projects. Virtual Breadboard [10] is a commercial product, which provides a similar environment but with a simulator that can be useful for users to debug circuits. To the best of our knowledge, none of the existing work has studied the effectiveness of autocomplete in assisting in the creation of virtual breadboard circuits.

Circuit Prototyping Tools

Creating circuit and electronic hardware artifacts is difficult for novice users, who do not have a strong electronics, engineering, or programming background. However, the recent development of open-source hardware platforms (e.g., Arduino [1], Phidgets [25], or Microsoft .NET Gadgeteer [44]) has significantly reduced the technical barrier for the novice users. With technologies like [28, 32], the users can even print their circuits at home on plastic sheets using a modified inkjet printer. CircuitStack [45] uses a computationally-generated PCB layer to ensure that the breadboard components are correctly connected. The device, however, was not designed to prevent users from making mistakes other than wiring errors. Another line of research in this space leverages the concept of generative design. Trigger-Action-Circuits [13], for example, provides the user with a list of candidate circuits based on a high-level behavioral description of the user's goal. However, even with these efforts, prototyping electronic circuits on a breadboard is still prone to software and hardware errors [19, 38]. Booth, et al. [19]'s study with 20 participants identified a number of common errors, including missing components, incorrect components, miss-wiring, and bad seating, as part of the typical experience of novice users. Similar result were found from Mellis et al.'s study [38], which also suggested that debugging hardware and software errors can be frustrating for inexperienced users. Little research has been conducted to reduce circuits errors when they are designed and created. This served as motivation to explore autocomplete features in this work.

Circuit Debugging Tools

Since errors are inevitable, tools for debugging becomes necessary. Scanalog [42] is a circuit debugging tool, which

provides a software environment for the user to view the behaviors of hardware components when manipulated by a user. Toastboard [22] is an interactive breadboard, providing hardware (e.g., LED) and software interfaces to facilitate debugging breadboard circuits using a known schematic provided by the user. Bifrost [37] integrates both hardware and software debugging tools in a unified environment. Digilent Electronics Explorer [21] is a commercially available product in a breadboard form factor, with built-in debugging tools including an oscilloscope, pattern generators, and logic analyzer. CurrentViz [47] can sense and visualize electric current flowing on a breadboard and CurrentSense [48] can sense the location and type of an component on a breadboard. Both technologies can be used to for debugging breadboard circuits. Although debugging cannot be avoided, techniques that can effectively reduce circuit errors during design and construction can largely reduce the time and efforts the users spend on debugging.

SYSTEM OVERVIEW

This section discusses component types, modules, and four autocomplete suggestions supported by our system.

Component Categories

We classify the common circuit components into five major categories: (1) basic components, (2) I/O components, (3) IC components, (4) MCUs, and (5) batteries.

Basic components. This category represents primitive electronic components, such as resistors, capacitors, inductors, diodes, and transistors, etc. The components in this category are commonly used as primitive building blocks to form functional circuits for the I/O and IC components. Novice users without an electronics background may find it hard to understand in what situations these components are to be used. Our system shows the user what and when basic components are needed for a certain I/O or IC component to function properly (as a module). This reduces the technical barrier for designing circuit projects.

I/O components. This category represents the major user interface components, including input devices (e.g., push buttons), sensors (e.g., pressure or light sensor), and output devices (e.g., LEDs and motors). Many require being used with basic components or the modules of other components in order to function properly. An example for the former is LED with a resistor. An example for the latter is a flex sensor, which needs an amplifier module (an amplifier clip with a resistor) for increased signal amplitude for high-resolution data processing. The instructions on the requirements of the supporting modules or basic components are documented in the circuit schematics in the datasheets provided by component manufactures. Our system provides autocomplete suggestions related to such requirements according to a component's datasheet schematics.

IC components. This category includes varying types of integrated circuits (ICs), such as H-bridges and amplifiers. ICs are, in general, the most difficult to use for novice users. It requires users to understand the datasheet and schematics in terms of terminal functionality, requirements for supporting modules or basic components in order for an IC to work properly. Our system provides autocomplete suggestions for the ICs based on their datasheet schematics.

MCU and *batteries*. Hardware platforms (e.g., Arduino) and batteries are provided to the I/O or IC components as autocomplete suggestions.

Component Module

In our system, a module is an electrically functional circuit, composed of one of the following two options: (1) an I/O or IC component or (2) I/O or IC component with a supporting base component(s) or module(s). A module is defined by the component's schematic from manufacture's datasheet for the component to work properly. In our case, a module should work on its own if powered by a battery or an open-source hardware platform (e.g., Arduino). For example, a photocell module is composed of a photocell (I/O component) and a resistor (base component), which can be used directly with an Arduino board for power and data. An I/O or IC component may be associated with multiple modules according to its datasheet schematic. For example, the photocell module can also be created by replacing the resistor with a capacitor. In this case, sensor data needs to be retrieved from a digital pin of the Arduino board. Our system presents both options to a user to decide. The order of their presentation is based on how frequently they are used in the existing circuit projects shared by the Fritzing community. The one with higher frequency is ranked higher. A user can also use the module with another module for extended functionalities. For example, the user can connect an H-bridge module to a DC motor module to change the rotation direction of motor dynamically (Figure 1).

A component can also be a module itself if the component can function on its own once powered. For example, a DC, stepper, or servo motor is an I/O component, but they are also modules because no other basic component or module is needed for them to spin. Examples of such components include slide/rotary potentiometer, rotary encoder, infrared proximity sensor, barometric pressure sensor, and peltier element. Note that for many components, even though their datasheets may suggest that working on their own is possible, a basic component like a resistor is often required in most cases to protect the components or circuit. An LED is an example because for most cases it will burnout without a resistor. Similar examples include push button, feed switch, toggle switch, and tilt switch. For such components, a resistor(s) is suggested by our system.

Modules defined by the datasheet schematic do not necessarily represent the minimum requirement for an I/O or IC component to work. For example, an ID-20 RFID chip has a leg dedicated for visual feedback using an LED. Although the LED is included in the datasheet schematic, the chip works without it. Presenting this information to users

through autocomplete creates an opportunity to learn the functionalities of the chip.

Autocomplete Suggestions

Our system provides four types of autocomplete suggestions: (1) Module Completion, (2) Module-to-Module Completion, (3) Wire Connection Completion, and (4) MCU or Battery Completion.

Module Completion (A1). Based on a newly inserted I/O or IC component, our system suggests ways to form a component module based on its datasheet schematic. The suggestions are ranked based on their frequency in our circuit database. Aside from saving time and reducing errors, this feature also allows users to learn alternative ways to create component modules.

Module-to-Module Completion (A2). After a module is completed, the system suggests other modules that could be used for extended functionality. For example, the system can suggest a H-bridge (module) to the user if a DC motor is inserted into the virtual breadboard. This is done based on how frequently a candidate module is used together with the current one in the past by the community. The most frequent candidates are ranked at the top of the list. We only consider modules that are immediate neighbors in the circuit database as candidates to each other. For example, in the situation, where three modules A, B, and C are connected in series (e.g., A-B-C), only B is considered to be a candidate suggestion for A, not C because the order of connection often matters. For example, with a circuit connecting (A) a piezo speaker, (B) a 555 timer for generating signal waves to the speaker, and (C) a photocell module for resetting the timer, in series, the photocell is unrelated to the function of the speaker. Modules that are connected in parallel are also considered as candidate suggestions to each other, except that they are both connected to the power and ground since this way their functions are loosely related to each other.

Wire Connection Completion (A3). If a candidate module is accepted by the user, the system suggests wiring the current module according to a source project. Our system can also provide suggestions to complete wire connections if the user connects the first wire between two modules.

MCU or battery completion (A4). Upon completing a module, our system can also suggest a battery or hardware platform such as Arduino to complete the circuit.

We considered these four features the most fundamental to demonstrate the promise and usability of AutoFritz. Advanced features like suggesting based on component model or suggesting component values were left outside the scope of this work.

SYSTEM WALKTHROUGH

This section demonstrates a running example to illustrate different capabilities of AutoFritz. We demonstrate constructing a circuit for a *distance alarm*, which uses a distance sensor to measure the distance of a nearby object. If the distance is shorter than a threshold, the device turns on an LED and plays a sound using a piezo speaker.

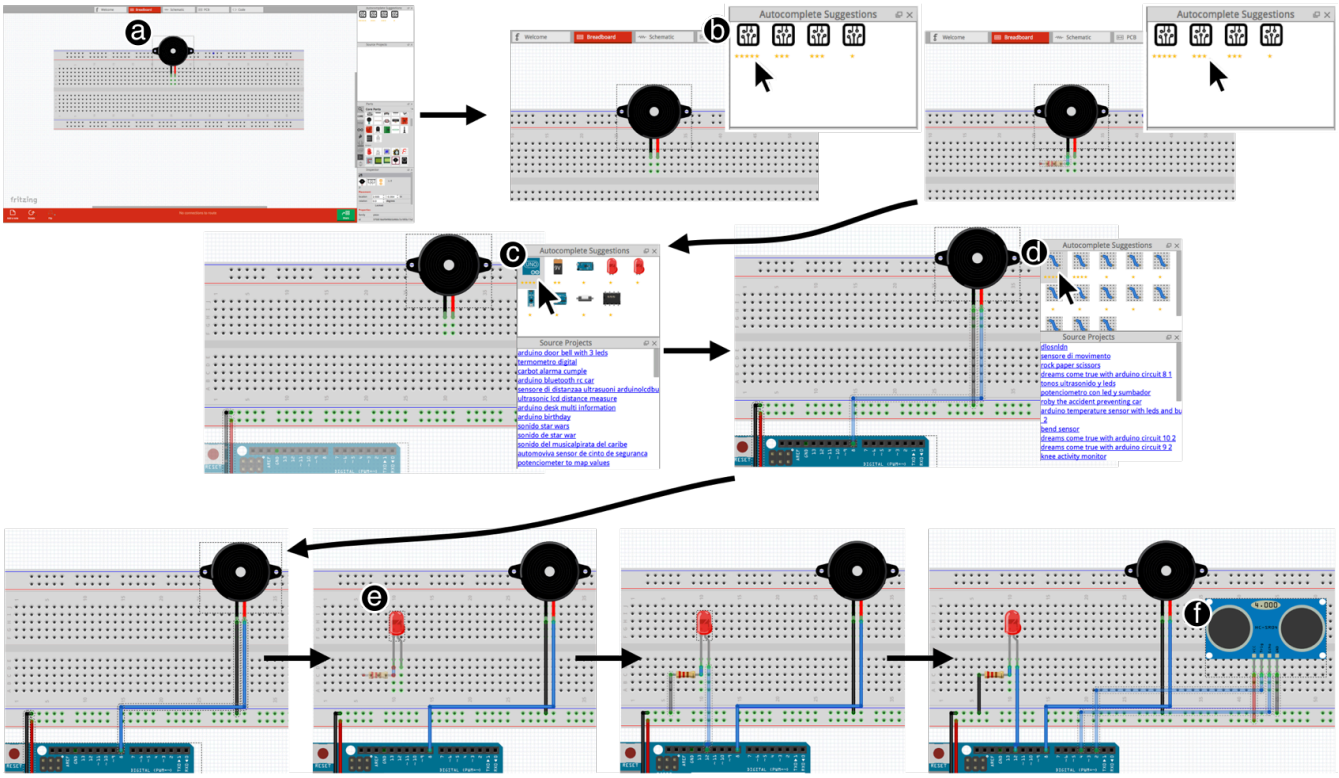

Figure 2. An example of the system walkthrough demonstrating the construction of a *Distance Alarm*.

With AutoFritz, a user drags a piezo speaker component into the virtual breadboard (Figure 2a). Once the piezo speaker is added, AutoFritz shows a grid of four suggestion icons in the Suggestion Panel, each represents a different way to complete the speaker component to a speaker module (A1) (Figure 2b). The suggestions are shown in a descending order from left to right (and up to down). Below each suggestion icon, the user finds a list of 1-5 stars, representing how frequently the suggested way of completing a module appears in the existing circuits in our database. For example, 5 stars indicates that the suggestion appeared in more than 40% of the circuits in the database. Four stars indicates the range between 30% to 40%, etcetera. The top-ranked suggestion is shown in the semi-transparent overlay on the breadboard. In this case, nothing is shown indicating that most people used the piezo speaker as a stand-alone module. If the user wants to further examine the suggestions, they can hover the cursor over a suggestion icon, which shows the suggested circuit in semi-transparent on the breadboard. The user finds that the first suggestion seems to be the one they need and click the icon to complete the module.

Next, the system updates the suggestion panel to show the modules (A2), MCUs, or batteries (A4) that are often used with the speaker module (Figure 2c). The user can browse the icon images to have a brief idea of what component is suggested. She also places different suggestions on the breadboard to examine them closely. She undoes the unwanted elections. Finally, the user places the cursor over the first suggestion (e.g., Arduino Uno), and notices that in the Source Panel, there is a list of links that provides source projects, based on the suggestion made. The user clicks the links and finds that they are all projects involving a speaker directly connected to the Arduino Uno. The user is now confident about their choice and selects the Uno for the piezo speaker. The system now updates the Suggestion Panel (Figure 2d) to show varying ways of wire connections (A3). The user selects one that seems viable for the goal of completing the speaker circuit. Following a similar procedure, the user inserts the LED (Figure 2e) and distance sensor modules (Figure 2f). She connects them to the Arduino board by following AutoFritz's suggestions.

IMPLEMENTATION

We developed AutoFritz on Fritzing [5, 33], an open-source virtual circuit prototyping tool. The software provides a drag-and-drop interface that allows users to design and construct circuit projects on a virtual breadboard. Our system has two parts: (1) a database, storing autocomplete logic; and (2) a frontend user interface integrated into the Breadboard and Parts view of Fritzing for showing suggestions.

Data Collection

Our autocomplete suggestions are made based on datasheet schematics and existing circuits shared by the Fritzing community [4]. For the circuit data, we included 67 example circuits provided by the Fritzing software and 4065 circuits downloaded from Fritzing's project website [4] (4132 in total). Circuit data was downloaded in Fritzing's XML format. We converted and stored them in an SQLite database. Our implementation included 84 circuit components commonly used by novice makers (e.g., 36 I/O, 22 IC, 11 basic, 7 batteries, and 8 MCU). The components were chosen from the Fritzing software as well as those used in more than 1% of the 4132 circuit projects we have in our database.

Note that Fritzing does not separate different models of the same component type. For example, models of the same piezo speaker are treated the same in the software. One of the main reasons is that they are not very much different from each other in terms of functionality and wiring connection. This is particularly true for the components commonly used by novices, to whom presenting the classified models may be unnecessary and likely cause confusion. We adopted the same strategy by presenting the components based on type not model. Technically, it makes little difference for most of our components as we found the datasheet schematics of common-used models of a component type are mostly the same. In cases when they are different (e.g., some models of piezo speakers can work without a resistor, but some cannot), we presented all of them as suggestions for that component. Since our goal was not to exhaust all the possible models for different types of components, we surveyed around three models per component.

Figure 3 shows all the different types of I/O components that was included in our implementation. For each of them, we found the datasheet of different models and documented the schematics in the SQLite database. Our database contains information about all the components, functions of component legs (e.g., polarity, VCC, data, etc.), component schematics (or modules), and all the collected circuits. We walked through all circuits in the database. Within each circuit, we first identified all the I/O and IC components. For each, we also found its module within the circuit based on the schematics in our database.

There are several cases where constructed component modules are different from those in our database since our collection of database schematics may not cover all the possible ways the module of an I/O or IC component can be constructed. Examples include those needing additional protection from an extra basic component, such as LEDs or toggle switches that are often used with a resistor. These methods were missing in our initial database. To resolve this type of problem, we manually examined the circuits, which contained the modules constructed in the ways that are different from our database. We created a supplementary schematic in the database if the module were constructed correctly (see Figure 3). In cases that a module was created wrong, we corrected it in the circuit. We also calculated how frequently a module appeared in our circuit database and present them in a descending order in our suggestions for Module Completion (A1). There are cases where a circuit component was unknown to our system. We excluded them from the current work.

With all the component modules identified in a circuit, we were able to figure out whether and how they were connected to each other by analyzing if the endpoints of two modules were connected to each other or not. This information was stored in the database for providing suggestions for Module-to-Module Completion (A2).

Component	Supplementary Module
INPUT	
LM35 Temperature Sensor	
Trimmer Potentiometer	
Humidity and Temperature Sensor RHT03	
Basic Force Sensing Resistor (FSR)	
RFID Reader	
Basic Flex Resistor	
Slide Potentiometer	
Rotary Potentiometer	
Infrared Proximity Sensor	
Rotary Encoder	
Hc-sr04	
Barometric Pressure Sensor	
Triple Axis Accelerometer Breakout	
Pushbutton (2 pins)	1 resistor
Pushbutton (4 pins)	1 resistor
Reed switch	1 resistor
Toggle Switch	1 resistor
Tilt Switch	
Photocell (LDR)	
Antenna	
OUTPUT	
128 x 64 Graphic LCD	
LCD screen	
7 Segment Display	
LED Dot Matrix 18mm (0.7 INCH)	
Microphone	1 resistor
Piezo Speaker	1 resistor
LED (2 pin)	1 resistor
LED (4 pin)	1 resistor
RGB LED	3 resistors
Loudspeaker	1 resistor
Solenoid	1 diode, 1 transistor, 1 resistor
Stepper Motor - Bipolar	
DC Motor	
Basic Servo	
Peltier Element	
DIP-Relay-D31A	

Figure 2. I/O components included in our implementation. Supplementary Modules are those created by us according to common practice.

Fritzing-end User Interface

We modified Fritzing's source code to show the top-ranked suggestions in its Breadboard view with the suggested components and wires, rendered semi-transparently. We also changed Fritzing's Parts view by including a Suggestion Panel and Source Panel (Figure 4). The Suggestion Panel shows all the suggestions in a grid view. The Source Panel was implemented to show links to source projects of each suggestion, so that a user can see the reason why a particular suggestion was made by our system.

Figure 4. AutoFritz's Suggestion and Source Panel.

USER STUDY

We conducted an experiment to evaluate the effectiveness and usability of autocomplete in the design of virtual breadboard circuits. In particular, we were interested in

understanding whether autocomplete could help reduce circuit errors and improve task efficiency for users with and without a strong background in electronics. We were also interested in whether and how the complexity of target circuits may affect the effectiveness of our tool.

Participants

Sixteen participants (7 female) between the ages of 21 and 26 participated in the study. Half of our participants had a bachelor's degree in electronics and the remaining did not and deemed themselves nonexpert, having some knowledge from high school and college. Before they came to the study, participants were given a tutorial about Fritzing and our autocomplete features demonstrating how virtual circuits can be created. We gave them a small test to create a simple circuit to ensure that they were all familiar with the functions and interfaces of the software.

Task and Procedure

The tasks involved creating four circuits with two levels of complexity (e.g., low and high). Participants were asked to

complete two circuits per complexity level (with and without) using the autocomplete tool. All the circuits were chosen from Arduino [2] or Arduino Project Hub [3].

The two circuits in the low complexity category involved modules of simple components, such as LEDs, piezo speaker, push buttons, or photocell. The first circuit is Morse Code Communication [9] (Figure 5 top), involveing three I/O components (e.g., push button, LED, and a piezo speaker). It takes input from two push buttons, one for dot and another one for dash. An LED flashes when a button is pressed. The input letter is pronounced using a piezo speaker. A resistor is required for all I/O components expect the piezo speaker, which worked either way in our implementation. The second circuit is Mixing Color Lamp [8] (Figure 5 bottom), which involves two I/O components (e.g., tri-color LED and photocell). It changes the color of a tri-color LED based on lighting conditions, sensed from three photocells, one per R, G, or B channel. Resistors are also needed for the components.

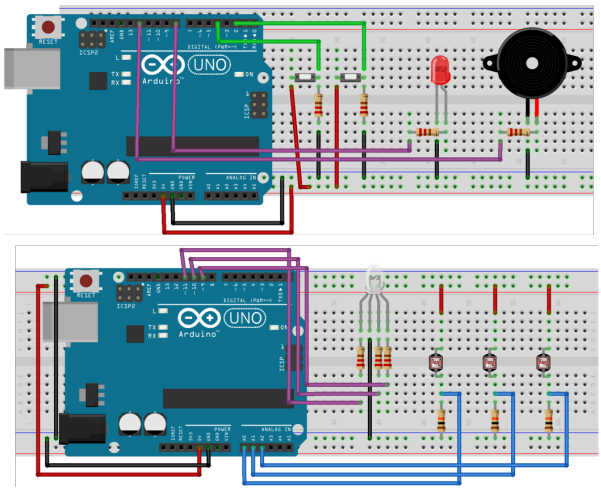

Figure 5. Low complexity circuits: Morse Code Communication (top); Mixing Color Lamp (bottom).

The circuits in the high complexity category involve modules of more complicated components, such as ICs. The first circuit in this category is a modified Guitar Speed Pick and Stomp Pedal [6] (Figure 6 top), which involves three I/O and IC components (e.g., DC motor, push button, and H-Bridge). It takes input from a push button to turn on/off a 9V DC motor. Once the motor is turned on, the speed of the motor has to be controllable through Arduino code (e.g., changing from the lowest to highest and vice versa). This means that participants had to figure out that an H-Bridge (L293D) had to be used in the solution. A battery is also needed to power the DC motor. The second circuit is LCD Thermometer [7] (Figure 6 bottom), which involves two I/O components (e.g., LCD, thermometer). It shows the temperature readings from a thermometer on a 16x2 LCD display. Participants had to figure out how to properly connect the LCD to the Arduino board. A rotary potentiometer (or resistor) is needed for the LCD to display in the right contrast. Since the components involved in this study are quite common, the suggestions associated with them were all top ranked between 1 to 3.

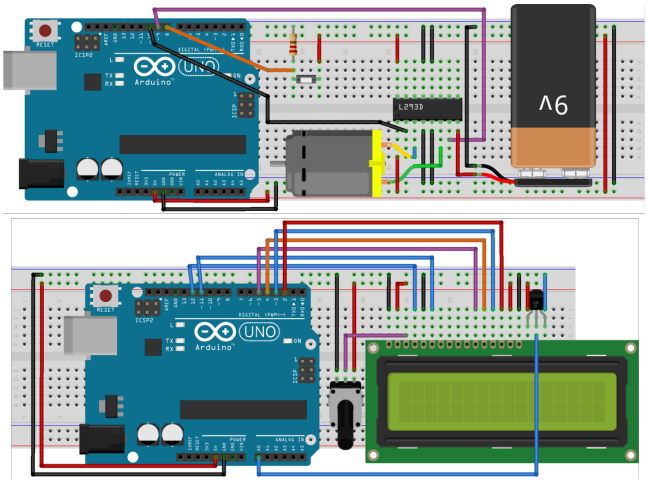

Figure 6. High complexity circuits: Guitar Speed Pick and Stomp Pedal (top); LCD Thermometer (bottom).

For evaluation, we only cared about the correctness of the circuit components and their connections. Participants did not have to specify the component values, nor did they have to program in Arduino. The aesthetics of the final circuits was also not considered in our evaluation.

Prior to the study, participants were informed that they needed to create four circuits using Fritzing. We gave them a textual description of the functionalities of the circuits that they had to develop. The text also listed the components they had to use for all four tasks including some distractor components, but without telling them how to use them to complete the task. During the study, participants had access to the task description sheet, the help content, examples built into Fritzing, and access to the internet for them to search for information. They were informed that they should not search for the exact solution (e.g., do not search for 'use a 16x2 VISHAY LCD to show temperature'). Half of the study required participants to finish the tasks with the autocomplete features. In the remaining half, they completed the tasks using the standard Fritzing software. The two conditions were counter-balanced among participants.

Similar to [13, 19], we also constrained task completion time. Since participants were not required to program in Arduino, we gave them 20 minutes per task. Participants were asked to do their best to accomplish the tasks without using a simulator to test or debug. This way we were able to measure their confidence and correctness of their initial answers. Participants were not allowed to modify their answer or view the correct solution once they submitted their answers. Upon completion of the study, participants filled out a post-experiment questionnaire where they indicated subjective ratings for *Mental Effort*, *Frustration*, and *Confidence* (1: very low, 7: very high) using a continuous numeric scale. Decimal ratings like 3.8 were permitted.

Results

The data was analyzed using Mixed ANOVA. Independent measures included Expertise (*Novice* and *Expert*), Autocomplete (*with* and *without AutoFritz*), and Task Complexity (*Low* and *High*).

Task Completion Rate

Among the 16 participants, 11 were able to complete the four tasks within the time constrain, with the uncompleted tasks being Guitar Speed Pick and Stomp Pedal. Table 1 summarizes the reasons for the failures. Circuit correctness is discussed in the next section.

ID	Expertise	AutoFritz	Reason
P1	Novice	Yes	Unable to figure out how to use the H-Bridge with the DC motor.
P4	Novice	Yes	Unable to figure out how to use the H-Bridge with the DC motor.
P2	Novice	No	Unable to figure out how to use the H-Bridge with the DC motor and button with Arduino.
P5	Expert	No	Completed the circuit by mistakenly using the push button as a toggle button, which left no time for P5 to figure out how to fix the issue.
P7	Novice	No	Spent all the time trying to power the DC motor using Arduino.

Table 1 Summary of task failures.

Both P1 and P4 failed to complete the tasks even with AutoFritz. In case of P1, the participant was able to find tutorials about DC motors, H-Bridge, and the (right) suggestion provided by AutoFritz. However, the participant was unable to understand if the H-Bridge was the right component for the task. Digging into the source project of AutoFritz's suggestions to understand the concept and reasoning was beyond this participant's capability within 20 minutes. P4 was similar, being able to connect the DC motor with the battery by following the suggestion of AutoFritz. However, the participant was stuck at choosing what motor controller to use because of the same reason. Therefore, some basic understanding of the desired components is still needed in order for AutoFritz to work. Therefore, we expect that AutoFritz should work better for people who are familiar with standard electronic components. When AutoFritz was not used, participants failed to complete the task because they did not know how to power the DC motor (P7) or use the push button (P2, P5). These issues did not occur when AutoFritz was used. For example, both P2 and P5 were able to successfully complete the Morse Code task, which also required them to use a push button.

Task Success Rate

Every answer submitted by participants was examined by an experimenter, excluding the five uncompleted ones. If the components and wire connections are correct, the circuit was considered successful. The success rate represents the number of correct circuits participants were able to create without the help of a simulator.

The success rates with AutoFritz were 86.6% (13/15) and 92.8% (13/14) for the tasks with Low and High Complexity, which are both higher than those without AutoFritz , such as 68.8% (11/16) and 69.2% (9/13) for the tasks with Low and High Complexity. This suggests that AutoFritz helped participants build more correct circuits. Being an expert does not mean they made less errors. Our results showed that experts made more errors than novice users, even with AutoFritz (Figure 7).

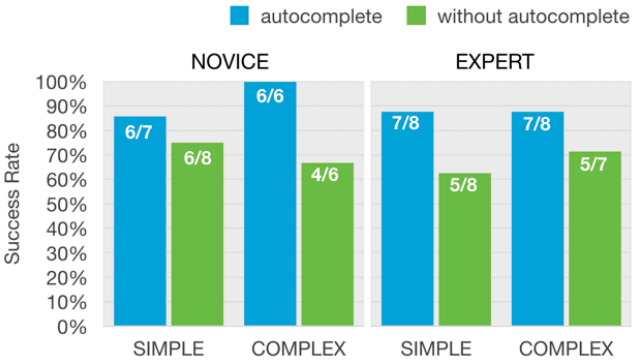

Figure 7. Task successful rate shown by Task Complexity and Expertise.

We observed three types of errors occurring without AutoFritz across the participants of different levels of expertise: (1) *missing basic components*. For example, participants forgot the resistors for LCD, LED, photocell, and push button. (2) *missing wire connections*. For example, a participant forgot to connect a potentiometer's pin to vcc. Another participant missed a wire to connect the push button to Arduino's digital pin. (3) *incorrect use of I/O or IC components*. For example, a participant did not know how to use the H-Bridge correctly. Another participant connected the photocell to Arduino's analog pin and ground. The last participant used the Arduino to power the DC motor.

All three types of errors can largely be reduced with AutoFritz, as only 3 out of 29 submissions contained errors, and the reasons for them were very similar, as participants were not sure about which suggestion to choose for their task because they were uncertain about the function of the suggested modules. For example, a novice user who did not how to use the photocell, examined the suggestions provided by AutoFritz and chose one that did not work for the task. An expert initially selected the correct suggestion for the push button but decided to ignore it and develop their own incorrect solution instead. Similarly, another expert ignored the suggestion of H-Bridge, and created a circuit to control the DC motor directly from Arduino because this participant believed that it should also work. These two experts appeared to be unfamiliar with the tested components but were still quite confident about their electronics skills. We believed that eventually they might be able to figure out the correct solutions, but time would be wasted. To address this issue, AutoFritz should provide more informative suggestions to help users to understand the functionality of the suggestions. The current way of showing the source projects of the

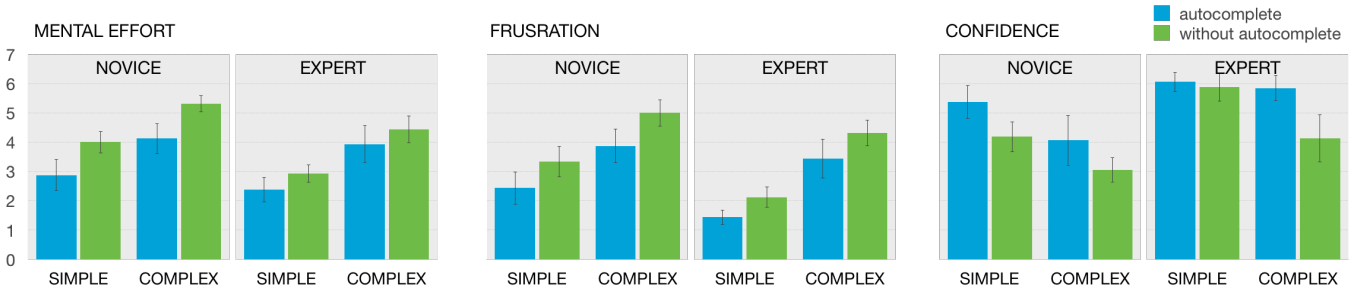

Figure 8. Subjective ratings on mental effort, frustration, and confidence.

suggestions did not work as well as we expected because many of the source projects were unrelated to the tested task. Therefore, we still saw participants relied on the web to find relevant tutorials.

An interesting benefit of AutoFritz is that it allowed users to learn new ways to construct a circuit, in ways that they might not have thought about. This is something primarily appreciated by the experts. For example, an expert told us that *"I did not know that the LCD has to be used with a potentiometer until I saw the suggestion, and I googled and found that it was a common thing to do. I am glad that I learned something new" (P6)*. Another expert told us that *"I didn't know how to drive a DC motor and was thinking about creating a driver circuit by myself. Then I saw the L293D (H-bridge) from the software and was happy to learn that it worked" (P3)*. This is similar to how software engineers often learn new comments the autocomplete suggestions in a programming IDE, and we expect that AutoFritz can also be a good learning environment for the users.

Mental Effort

The amount of mental effort spent on completing the tasks was not significantly different ($F_{1, 14} = 0.52$, $p = 0.822$) from Novice (3.80, SD = 1.31) and Expert (3.69, SD = 1.88). As expected, mental effort was higher with tasks of High (4.34, SD = 1.50) than Low Complexity (3.14, SD = 1.50) ($F_{1, 15} = 21.343$, $p < 0.01$). AutoFritz helped reduce participants' mental effort ($F_{1, 15} = 18.204$, $p < 0.05$) Participants found their mental effort higher without AutoFritz (4.44, SD = 1.44) than with AutoFritz (M = 3.05, SD = 1.48) (Figure 8).

Six out of eight novice users told us tasks were much easier with AutoFritz. A participant stated "*I felt like I was an expert with AutoFritz as I could create circuits without spending all the time googling" (P2)*. We see this as encouraging evidence highlighting the potential of AutoFritz in encouraging novice learners to be more active in participating in circuit design and prototyping. Experts appreciated AutoFritz from a different perceptive. Unlike novice users, many of them had a better idea of how the pins of the components should be connected correctly. However, connecting the wires manually can be tedious and error-prone. Some of our expert users told us that they liked the suggestions for Wire Connection Completion (A3) the most because it "*saved a great deal of effort from tedious wiring" (P10, P14),* and essentially *"allowed them to focus on the design of the function of the circuit" (P11)*. This is an important benefit of AutoFritz, which we hoped to achieve when designing the software.

Frustration

Frustration scores from Novice and Expert across all the conditions were 3.66 (SD = 1.72) and 2.83 (SD = 1.68), but the difference was insignificant ($F_{1, 14} = 4.05$, $p = 0.06$). Similar to mental effort, participants found the tasks with High Complexity (4.15, SD = 1.59) more frustrating than Low Complexity (2.33, SD = 1.38) ($F_{1, 15} = 40.98$, $p < 0.01$). Participants were less frustrated with AutoFritz (2.80, SD = 1.75) than without it (3.69, SD = 1.63) ($F_{1, 15} = 4.90$, $p < 0.05$).

Both novice and experts found AutoFritz helpful in a time-sensitive task in that *"it can help me quickly find a solution and place it automatically on the breadboard" (P2, P10, P11)*. This allowed them to *"insert an answer first and verify it afterword using online tutorials" (P8, P14, P15)*. By having an answer at hand, even uncertain about whether it is the right one to user or not, certainly reduced the level of frustration of our participants. We foresee that this can be particularly useful for improving the learning experience of novice learners or students at school, whose engagement in learning electric circuits is largely affected by frustration.

Experts may not necessarily know the best solution for a problem that they were unfamiliar with and this can be frustrating for them. AutoFritz can mitigate such frustration. For example, one of our expert participants mentioned that *"I could have completely missed the H-Bridge and gone for a way more complicated solution if autocomplete was not provided. I will definitely be more frustrated as I do not know if my initial thought works or not." (P11)*.

Being uncertain about which suggestion to use is a source of frustration, especially for novice users dealing with a difficult problem (e.g., tasks of High Complexity). For example, a participant commented that "*I was a bit frustrated when I did not understand the functions of these suggested modules and which one was the right one to use" (P3)*. Many novice users were not interested in reading the source projects to learn the components if learning was not their goal, in which case, AutoFritz could be less effective than alternatives such as generative design [13].

Confidence

Participants were asked to rate their level of confidence regarding the correctness of their answer. As expected Expert (5.48, SD = 1.66) was more confident than Novice

(4.17, SD = 1.85) ($F_{1, 14} = 5.55$, $p < 0.05$). Participants were more confident about their answers to the tasks with Low (5.37, SD = 1.50) than High Complexity (4.27, SD = 2.06) ($F_{1, 15} = 18.01$, $p < 0.05$). Participants found themselves more confident with AutoFritz (5.34, SD = 1.76) than without AutoFritz (4.31, SD = 1.86) ($F_{1, 15} = 5.95$, $p < 0.05$).

Both experts and novices were confident about their initial answers with AutoFritz even without using a simulator to test their answer. This is because they trusted the sources of the suggestions, since "*all the suggestions are from datasheets and example circuits" (P2, P9, P15)*. Component datasheets and online circuit examples and tutorials are the main sources of information, where participants usually receive help from. Another reason for them to be more confident when using AutoFritz is that *"the software mainly took care of wire connects so the chance for me to make wiring error is reduced" (P2)*. Note that it may take time for people to develop confidence on AutoFritz before they completely trust its suggestions. For example, we found many participants spent more time on their first autocomplete session than the second one. The time was spent on googling tutorials first and comparing their findings with the suggestions from AutoFritz. This is particularly true for the novice users. Our star ratings also helped participant develop confidence on their choices among the suggestions based on their frequency in the existing circuit projects.

LIMITATIONS AND FUTURE WORK

We present insights we learned from our work, discuss its limitation and suggest future research directions.

Autocomplete & generative design. Providing help to users in the design and creation of electronic hardware is a fruitful research topic which has attracted attention recently with solutions proposed from many different perspectives. One of these is generative design, which allows a novice user to specify a high-level description of the needs and lets the computer generate the solutions (e.g., breadboard schematic) for the user. This allows a novice in electronics to create hardware projects. In this context, a user may not have the desire to design or learn electronics as their goal is to complete the task. Autocomplete is different because the user is in the center of the loop, where technology assists the user in the design and creation of virtual breadboard circuits, and the desire to design and construct is essential. Circuit autocomplete (at least in our implementation) is lacking in the consideration of a user's current goal and context, which was shown in our study to be important for providing meaningful suggestions to the users. We believe circuit autocomplete can benefit from the concepts of generative design in better understanding the user's needs. Common grounds can be found for the two approaches. AI and machine learning techniques can also help improve autocomplete in understanding the user thus better serving the user's goal.

Implementation. The current implementation of AutoFritz allowed us to explore of the concept of autocomplete in the construction of virtual circuits and gain initial insights into it usability. Our implementation can be improved in many ways. For example, fine-grained classification of circuit components should be considered to provide more precise suggestions and reduce the time and efforts from the user to search among relative suggestions. Suggesting component value would also be a useful addition to the current system, which again requires understanding a user's goal. Another improvement is to consider the user's own search history.

Our database was created using an approach mixed with manual and automatic operations in parsing circuit data from the web. This approach is limited in scalability to outside the Fritzing's website, where the circuit data may not be stored in a structured form like XML. This is the main challenge we had when trying to get data from Arduino's project hub, which presents the sample circuits using RGB figures of breadboard circuits. We intend to develop image processing techniques to parse the data from the figures and test the robustness of our system over a larger dataset. With more data in the database, it is also important to be sure that querying autocomplete suggestions is up to speed.

Our ranking system is based on frequency, which can be improved in the future to incorporate user's goal, context, and the user's own search history. Existing research in autocomplete in programming, web search, or text editing can provide useful insights to improve our implementation. Finally, it is possible that an inserted component has not occurred historically, meaning there will always be opportunity for future research to assist in these situations.

Evaluation. Our evaluation focused on the usability of AutoFritz in a controlled environment. It is our goal to deploy it in the wild and test it with users who have a variety of backgrounds and goals, understand the issues with the current design and implementation, and iterate the improvements in both system and usability design.

CONCLUSION

We introduce the concept of autocomplete to the design and construction of virtual breadboard circuits. With autocomplete, the system can provide a user with a list of suggested components based on the one that is inserted by the user. The suggestions complete or extend the electronic functionality of the inserted component to reduce circuit error. To demonstrate the effectiveness and usability of the proposed idea, we implemented autocomplete on Fritzing, an open source software, based on schematics from datasheets for standard components and over 4000 circuit projects from the Fritzing community. Our implementation provides Module Completion, Module-to-Module Completion, Wire Connection Completion, and MCU or Battery Completion. Through a controlled user experiment with 16 experts and novices, we demonstrated the effectiveness of autocomplete in creating virtual breadboard circuits of different levels of complexity. We sheared our insights gained from this research and foresee that this work can inspire a fruitful line of future research in the field of rapid prototyping.

REFERENCES

1. Arduino. http://arduino.cc.
2. Arduino Blink Example. https://www.arduino.cc/en/tutorial/blink. 2018
3. Arduino Project Hub. https://create.arduino.cc/projecthub. 2018
4. Fritzing Project Hub. http://fritzing.org/projects/. 2018
5. Fritzing Software. http://fritzing.org/home/. 2018
6. Guitar Speed Pick and Stomp Pedal. https://create.arduino.cc/projecthub/marc_uberstein/guitar-speed-pick-and-stomp-pedal-35a4e3?ref=platform&ref_id=424_trending___&offset=3. 2018
7. LCD Thermometer. https://create.arduino.cc/projecthub/TheGadgetBoy/making-lcd-thermometer-with-arduino-and-lm35-36-c058f0?ref=search&ref_id=lm35&offset=1. 2018
8. Mixing color lamp. https://programminginarduino.wordpress.com/2016/03/01/project-04/. 2018
9. Morse Code Communication Using Arduino. https://create.arduino.cc/projecthub/Jalal_Mansoori/morse-code-communication-using-arduino-f339c0?ref=platform&ref_id=424_trending___&offset=1\. 2018
10. VirtualBreadboard http://www.virtualbreadboard.com/.
11. Serge Abiteboul, Yael Amsterdamer, Tova Milo and Pierre Senellart. 2012. Auto-completion learning for XML. in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, ACM, 669-672.
12. Altium Designer 17 Overview. http://www.altium.com/altium-designer/. 2017
13. 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*, ACM, 331-342.
14. Autodesk Circuits. https://circuits.io/. 2017
15. EAGLE PCB Design and Schematic Software. 2017
16. Holger Bast and Ingmar Weber. 2006. When you're lost for words: Faceted search with autocompletion. in *SIGIR*, 31-35.
17. Olivier Bau and Wendy E Mackay. 2008. OctoPocus: a dynamic guide for learning gesture-based command sets. in *Proceedings of the 21st annual ACM symposium on User interface software and technology*, ACM, 37-46.
18. Mike Bennett, Kevin McCarthy, Sile O'modhrain and Barry Smyth. 2011. Simpleflow: enhancing gestural interaction with gesture prediction, abbreviation and autocompletion. in *IFIP Conference on Human-Computer Interaction*, Springer, 591-608.
19. 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*, ACM, 3485-3497.
20. Fei Cai and Maarten De Rijke. 2016. A survey of query auto completion in information retrieval. *Foundations and Trends® in Information Retrieval*, 10 (4). 273-363.
21. Digilent Electronics Explorer https://store.digilentinc.com/electronics-explorer-all-in-one-usb-oscilloscope-multimeter-workstation/. 2017
22. 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*, ACM, 677-686.
23. Stephen R Foster, William G Griswold and Sorin Lerner. 2012. WitchDoctor: IDE support for real-time auto-completion of refactorings. in *Software Engineering (ICSE), 2012 34th International Conference on*, IEEE, 222-232.
24. Darío Garigliotti and Krisztian Balog. 2017. Generating query suggestions to support task-based search. *arXiv preprint arXiv:1708.08289*.
25. Saul Greenberg and Chester Fitchett. 2001. Phidgets: easy development of physical interfaces through physical widgets. in *Proceedings of the 14th annual ACM symposium on User interface software and technology*, ACM, 209-218.
26. James Hays and Alexei A Efros. 2007. Scene completion using millions of photographs. in *ACM Transactions on Graphics (TOG)*, ACM, 4.
27. Will Hill, Larry Stead, Mark Rosenstein and George Furnas. 1995. Recommending and evaluating choices in a virtual community of use. in *Proceedings of the SIGCHI conference on Human factors in computing systems*, ACM Press/Addison-Wesley Publishing Co., 194-201.
28. Steve Hodges, Nicolas Villar, Nicholas Chen, Tushar Chugh, Jie Qi, Diana Nowacka and Yoshihiro Kawahara. 2014. Circuit stickers: peel-and-stick construction of interactive electronic prototypes. in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM, 1743-1746.
29. Kajta Hofmann, Bhaskar Mitra, Filip Radlinski and Milad Shokouhi. 2014. An eye-tracking study of user interactions with query auto completion. in

Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management, ACM, 549-558.

30. Eero Hyvönen and Eetu Mäkelä. 2006. Semantic autocompletion. in *Asian Semantic Web Conference*, Springer, 739-751.
31. Takeo Igarashi and John F. Hughes. 2001. A suggestive interface for 3D drawing *User Interface Software and Technology*, 173-181.
32. Yoshihiro Kawahara, Steve Hodges, Benjamin S. Cook, Cheng Zhang and Gregory D. Abowd. 2013. Instant inkjet circuits: lab-based inkjet printing to support rapid prototyping of UbiComp devices *Proceedings of the 2013 ACM international joint conference on Pervasive and ubiquitous computing*, ACM, Zurich, Switzerland, 363-372.
33. André Knörig, Reto Wettach and Jonathan Cohen. 2009. Fritzing: a tool for advancing electronic prototyping for designers. in *Proceedings of the 3rd International Conference on Tangible and Embedded Interaction*, ACM, 351-358.
34. David Koop, Carlos E Scheidegger, Steven P Callahan, Juliana Freire and Cláudio T Silva. 2008. Viscomplete: Automating suggestions for visualization pipelines. *IEEE Transactions on Visualization and Computer Graphics*, 14 (6). 1691-1698.
35. Jean-François Lalonde, Derek Hoiem, Alexei A Efros, Carsten Rother, John Winn and Antonio Criminisi. 2007. Photo clip art. *ACM transactions on graphics (TOG)*, 26 (3). 3.
36. Justin Matejka, Wei Li, Tovi Grossman and George Fitzmaurice. 2009. CommunityCommands: command recommendations for software applications. in *Proceedings of the 22nd annual ACM symposium on User interface software and technology*, ACM, 193-202.
37. 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*, ACM, 299-310.
38. 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*, ACM, 1270-1281.
39. Kyle I Murray and Jeffrey P Bigham. 2011. Beyond autocomplete: Automatic function definition. in *Visual Languages and Human-Centric Computing (VL/HCC), 2011 IEEE Symposium on*, IEEE, 259-260.
40. Sam Seifert. 2016. Autocomplete Sketch Tool.
41. Christian Sengstock and Michael Gertz. 2011. CONQUER: a system for efficient context-aware query suggestions. in *Proceedings of the 20th international conference companion on World wide web*, ACM, 265-268.
42. 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*, ACM, 321-330.
43. Steve Tsang, Ravin Balakrishnan, Karan Singh and Abhishek Ranjan. 2004. A suggestive interface for image guided 3D sketching. in *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, ACM, 591-598.
44. Nicolas Villar, James Scott, Steve Hodges, Kerry Hammil and Colin Miller. 2012. . NET gadgeteer: a platform for custom devices. in *International Conference on Pervasive Computing*, Springer, 216-233.
45. Chiuan Wang, Hsuan-Ming Yeh, Bryan Wang, Te-Yen Wu, Hsin-Ruey Tsai, Rong-Hao Liang, Yi-Ping Hung and Mike Y Chen. 2016. CircuitStack: supporting rapid prototyping and evolution of electronic circuits. in *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*, ACM, 687-695.
46. David Ward, Jim Hahn and Kirsten Feist. 2012. Autocomplete as research tool: A study on providing search suggestions. *Information Technology and Libraries*, 31 (4). 6-19.
47. Te-Yen Wu, Hao-Ping Shen, Yu-Chian Wu, Yu-An Chen, Pin-Sung Ku, Ming-Wei Hsu, Jun-You Liu, Yu-Chih Lin and Mike Y Chen. 2017. CurrentViz: Sensing and Visualizing Electric Current Flows of Breadboarded Circuits. in *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*, ACM, 343-349.
48. 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*, ACM, 311-319.
49. Jun Xing, Li-Yi Wei, Takaaki Shiratori and Koji Yatani. 2015. Autocomplete hand-drawn animations. *ACM Transactions on Graphics (TOG)*, 34 (6). 169.