Fritzing – A tool for advancing electronic prototyping for designers

André Knörig
andre.knoerig@fh-potsdam.de

Reto Wettach
wettach@fh-potsdam.de

Jonathan Cohen
jonathan.cohen@fh-potsdam.de

Potsdam University of Applied Sciences
Pappelallee 8-9
14469 Potsdam, Germany

ABSTRACT

Today a growing community of DIY-practitioners, artists and designers are using microcontroller-based toolkits to express their concepts for digital artifacts by building them. However, as these prototypes are generally constructed using solder-free technologies, they are often fragile and unreliable. This means a huge burden of care and upkeep for these inventions when they are either exhibited or sold.

We present a software application called Fritzing which allows artists, designers and DIY-tinkerers to prepare their hardware inventions for production. Through an interface metaphor based on the typical workflow of the target group, Fritzing has proven its ability to provide useful support in the steps following the invention of an interactive artifact.

Fritzing serves also as a tool for documenting these interactive artifacts. As sharing of knowledge has been a driving force within this new DIY-movement, there is a need for a consistent and readable form of documentation which Fritzing can provide.

Fritzing has also proven to be a useful tool in teaching electronics to people without an engineering background.

Keywords

Physical Interaction Design, Design Tools, Prototyping

INTRODUCTION

In design and art schools around the world people are using computer technology—both hardware and software—in creative and innovative ways[1]. This trend has also been embraced by a growing community of DIY-practitioners who are developing and sharing new ways of using technology[2].

The programming environment Processing [1] and the microcontroller Arduino [2] have been two important enabling technologies in this area, and in fact have become defacto standards. Processing is a Java-based programming language including an easy-to-use IDE with a focus on visual programming. Arduino is a system consisting of a microcontroller with USB or Bluetooth connection, and a programming environment targeted towards designers and artists who want to build interactive artifacts (Figure 1).

Both Arduino and Processing have a number of features in common, and these commonalities—principles—have been a strong influence in the development of Fritzing.

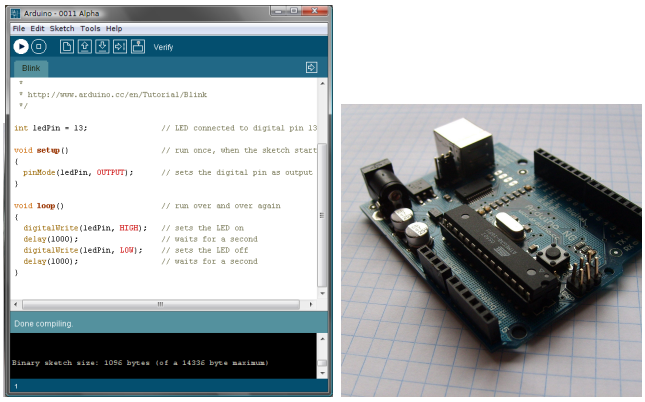

Figure 1: Simple electronics for designers: Arduino IDE and microcontroller

First, they both run on all three major operating systems. Prior to the development of Arduino there was no easy microcontroller-programming environment for Apple computers. What might seem as a minor contribution has been one of the reasons for the huge success of Arduino, because of the prevalence of Apple computers within the creative community.

---

[1] E.g., the „Physical Computing“ class by Tom Igoe at ITP/NYU, „Computer Related Design“ at the Royal College of Arts London, introduced by Gillian Crampton-Smith, „Design of Physical Interfaces“ class by Reto Wettach at the FH Potsdam

[2] For example, the MAKE:community, with its flagship print magazine, books, blog and regular events. http://www.makezine.com

Second, both systems heavily rely on the open source idea, not only in the way they have been and still are being developed, but also within their community of users. Processing, for example, has a feature which allows a publication of a project in form of a website, including an applet and the source code. Because of this feature, many projects built with Processing are being published with the source code, which allows other people, especially beginners, to learn from existing projects.

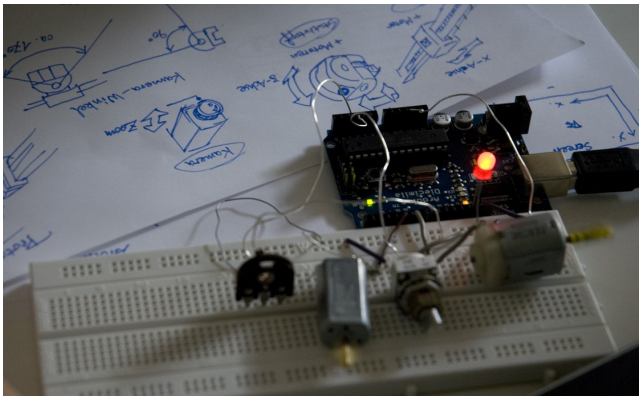

Figure 2: Typical manual prototyping with Arduino and breadboard

Third, both Arduino and Processing share an orientation towards designers and artists as users. The tools' functionalities, workflow, and language are focused on the activities of sketching: Without any initial setup, users can start experimenting immediately, and directly experience the results. A large set of examples can be accessed from the main menu as a starting point a user's own explorations.

CURRENT DESIGN WORKFLOW

Whereas electrical engineers usually work with CAD software and from there transfer directly to PCBs (printed circuit boards), designers/artists take a more hands-on approach. In conjunction with Arduino and other electronic prototyping kits, they use a breadboard and wire up their circuits manually (Figure 2). This proves to be compatible with a self-taught trial-and-error copy-and-paste approach [3]. It also enables the seeing-drawing-seeing cycle, where the design is explored in a tight iteration of modification followed by testing the experience [4].

The breadboard is therefore not a crutch but an essential aspect of the design process, allowing for quick changes and a step-by-step approach. However, the reliability and endurance of breadboard-based prototypes has been a big problem. They are too fragile to be presented outside of labs or studios. Furthermore there are limitations in miniaturization and replication of such prototypes.

CONCEPT

Fritzing supports designers and artists of electronic artifacts through this prototyping barrier. While the breadboard approach works well for sketching out an artifact, it does not facilitate the production of multiple stable copies. In order to advance the workflow to this stage, we have implemented a new approach to screen-based design of electronics. Instead of starting with schematics, as most Electronic Design Automation tools (EDAs) do, we decided to allow the user to document his working breadboard-based prototype with a visual metaphor that mimics the user's real world situation. Once this is accomplished, the software supports the user through the process of turning her circuit into a professional PCB (Figure 3).

Multiple Views

Fritzing offers three alternative views on the circuit: A real-world-like breadboard view, a classical schematic diagram view, and a PCB design view (Figure 4):

The default 'breadboard view' provides an abstracted illustration of the common electronic parts, primarily as seen from a top view. Parts that are more recognizable

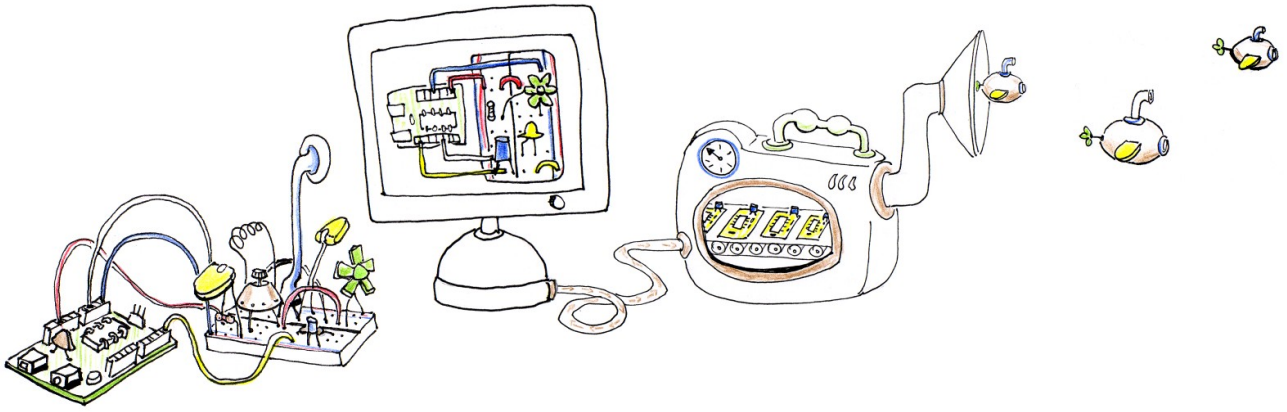

Figure 3: Concept of Fritzing: The recreation of their breadboard prototype in the Fritzing software enables non-engineers to produce professional PCBs and also to share their designs.

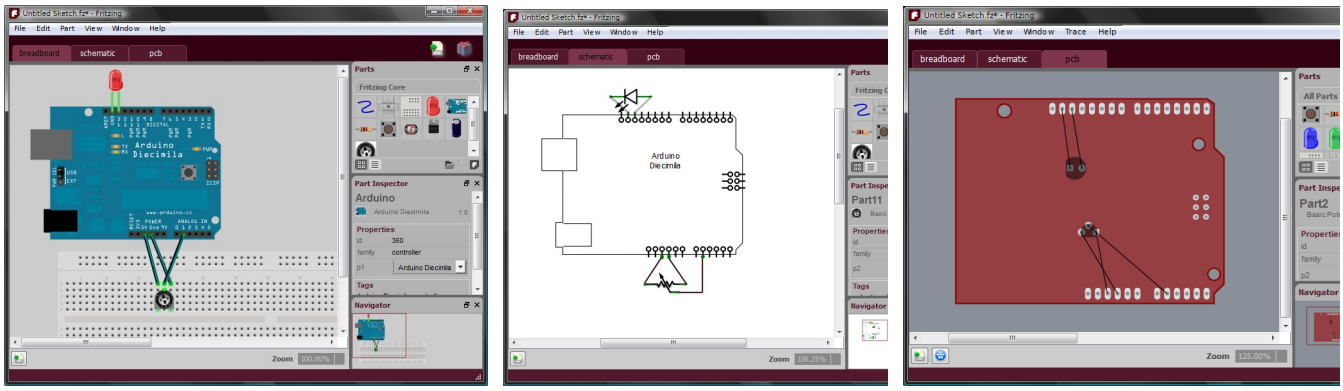

Figure 4: Fritzing's synchronized views: breadboard, schematic, and PCB

from the side (e.g., a capacitor) are drawn in a 30° perspective. The feel of the interaction is geared towards tangibility. For example, parts can be dragged from the palette onto the sketch area and wires can be drawn directly from the parts' leads or from breadboard holes. (By contrast, a typical EDA presents a tool palette, and selecting such a tool puts the user interface into a modal state.)

Fritzing's schematic view is valuable for teaching and also for dealing with more complicated circuits. It could also serve as an entry point for users coming from a more traditional electronics background.

The PCB view allows the designer to turn the sketch into a professional circuit board. This does not require much work, as Fritzing takes care of most of the tasks. In this view, the user can make adjustments to the positioning of the parts and control the routing process.

Challenges of the Multiple Views paradigm

This core feature turned out to be intricate for two reasons: Transition and Synchronization. Transition refers to the problem that a user who is only familiar with a breadboard view somehow has to learn about, and become comfortable with either schematic or pcb views – what they are about and how they are used. Fritzing's job partly is to educate users about these advanced options.

Synchronization refers to the problem that all views need to be always maintain the same state – a semantic change in any one view must be directly reflected in the others. This ensures a simple mental model and is another feature oriented to the non-engineer's unplanned style of working.

Transition

This problem was not clear to us from the beginning but proved to be a stumbling block for many early adopters with no technical background. Our approach is still evolving, but we have chosen a number of techniques. For example, when a part is selected in any view, its icon from the other views is also displayed in the parts inspector. Also, the navigator widget gives a small preview of the other views next to the currently selected one. Whenever a user performs an action in the main view, she can see that a simultaneous change occurs in the other views. A third approach is straightforwardly tutorial: When a user brings up any view for the first time, a guiding help text is the background of each window. We will be testing these approaches, and others, as Fritzing moves forward.

Synchronization

Our intention is that to a naïve, non-technical user the synchronization should be transparent and self-evident – even though technically it is not. The physical situations in the different views are certainly similar, but not equal, which brings up technical and user interface problems.

Technically, keeping the views in synchrony is a challenge. Originally, the plan was to have a single model, and each view would be a "view" of that model. However, as we found the form and behavior of each view becoming more divergent, we have migrated to a system of three models which communicate via messaging and command objects.

In general, parts are easy to synchronize, though sometimes parts visible in one view, are invisible in another—for example the breadboard. This enables us, for example, to easily preserve connections across views, even if the user doesn't directly see those connections. But the hardest part of the problem is when behavior in one view doesn't clearly map to another view. For example, connecting two parts by dropping them into the breadboard, results in the two parts being connected by wires in both breadboard and schematic view. This isn't too difficult. But what is trickier is when the user deletes one of those connecting wires in schematic view. The corresponding action in breadboard view is to lift one of the parts out of the breadboard, and keep the remaining connections by drawing a wire. Which leads to yet another ambiguity: Which part should get popped out of the breadboard? Or for a more complex ambiguity, what should happen in the breadboard when a user connects two nets in schematic view? Our solution is to draw wires and

create connections on a breadboard—and to add a breadboard, if necessary. Such cases simply require decisions. After a few of these have been solved, we are confident that we can continue with this strategy and add rules whenever new situations arise.

A related user interface issue, which seems simple at first is, if you move or rotate a part in one view, should that change propagate to the other views? Most of the time, the answer would be "no", because the positioning in each view follows different rules and considerations. But for a new user, who is struggling to understand the relationship between these views, having a part change across the three views might help the user keep the correspondence in mind. Currently, we only synchronize structural/semantic changes, but not changes that apply to the layout.

Parts

Another barrier presented by classic EDAs is part selection —typically the user must choose from an endless list of technical acronyms. Instead, Fritzing offers a visual parts bin containing a set of 'typical' parts. One part in a given family can represent any of the others, so that the user simply picks 'the' resistor part if she needs one. Later, the selected part can be made more specific by changing its properties. While this looks like a simple interaction to the user, underneath this is a database-backed operation for swapping the part with the one that has the desired properties. We intend to grow this mechanism to seamlessly search a web-based parts library in the future.

Figure 5: Parts editor for ad-hoc editing and creation of parts

Because the users in target group frequently make spontaneous use of special parts, we also make it easy to create parts on the fly,. A simple parts editor (Figure 5) lets the user define the necessary images (preferably vector graphics), the connectors, and metadata. Even unusual parts, such as those found in hacking a toy can be added using the parts editor.

Open interfaces

Fritzing makes consistent use of open interfaces to ensure flexibility throughout the workflow. All native files use an open XML schema, and it is possible to import footprints from and export circuits to the popular EDA tool Eagle [5].

The use of advanced graphics formats like SVG and PNG makes it appealing for designers. They can take special care of parts aesthetics and even style the PCBs, both shape and print, with their favorite graphics software. From a technical standpoint, basing our system primarily around SVGs gave us very fast infinite zoom capibilities; offered a general speedup over our previous pixmap-based versions; and enabled us to easily manipulate the graphics programmatically, for example, in export and in the parts editor.

Finally, the software is completely open source and built on top of the open Qt cross-platform application framework [6].

Website

The accompanying website, www.fritzing.org, provides documentation targeted at the learning style of the user group: It contains project examples and tutorials, pragmatic information about electronic parts, and links to similar resources (Figure 6). Soon, the site will be extended to become a hub for sharing projects, custom parts, and related knowledge, similar to the online community around Scratch [7]. We regard such a community website as an essential element in fostering creativity across the field, as recent research [8] and the examples of Processing and Arduino have shown.

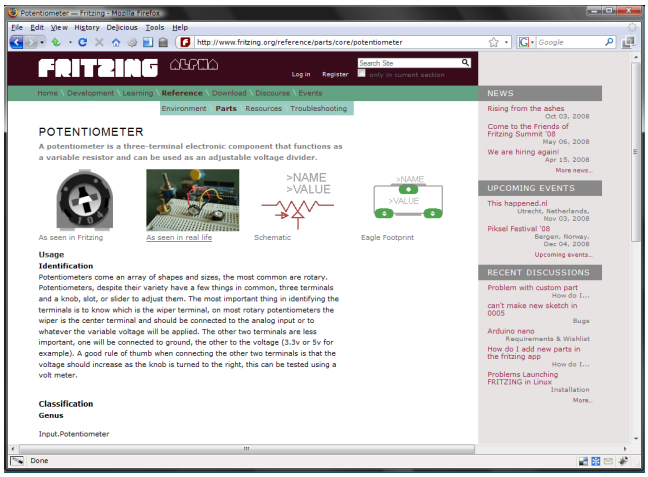

Figure 6: Community website for supporting open-source hardware and knowledge exchange

USE CASES

When developing Fritzing we based our design decisions on three possible use cases:

Documentation

As discussed earlier, an important factor for a thriving learning community is an easy way to document existing projects and share them. Currently, in the DIY community, electronic circuits are documented simply photographing them. However, these images are hard to

read due to parts occluding other parts, and the difficulty of following wires that cross and entangle.

Documenting with Fritzing

Fritzing provides a means for Arduino users to properly document their projects. By simply recreating the physical circuit in the software they create an archivable and shareable file. The file includes breadboard, schematics, and PCB view and can contain additional information such as notes or part ordering numbers.

The file can then be shared with colleagues or teachers, or published on the Fritzing website. A single-click share button in the main toolbar brings up the website for uploading the project. The shared file enables other users to interactively inspect the circuit by moving elements around and switching between views. They are invited to re-use and appropriate the shared circuits for their own projects – open-source hardware for the rest of us.

Teaching

Teaching "practical" electronics to a group of non-engineering students is not an easy task. The first difficulty is how to visually present circuits. The authors of this paper tried using camera and projection, which had the same occlusion and wire-following issues as previously discussed. Second, in consulting with students it proved to be difficult to "read" their breadboard-based prototypes, as they were not built with clarity-of-presentation in mind. Finally, it is a cumbersome task to find bugs on a breadboard, as there are so many reasons for errors: from wrong wiring of components, loose connections, broken parts, and problems with power supply, to errors in the software. If any of these multiple sources of errors can be eliminated, debugging becomes easier.

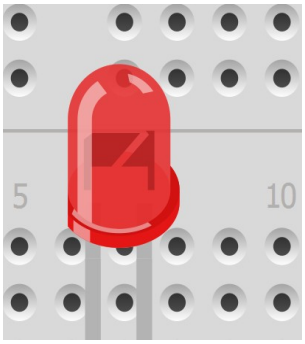

Figure 7: High-resolution graphics enable recognition of detailed physical hints and affordances

Teaching with Fritzing

In the classroom, the Fritzing software helped to teach electronics to a larger group of designers/artists by displaying it on a large screen. The high-resolution graphics allow us to point out detailed aspects, such as the bulge in the LED referring to the positive polarity (Figure 7). The schematics view was also introduced specifically with teachers in mind: By switching to this view students can gradually be exposed to the professional notation system and enabled to access the many resources that are based on it.

Consultation with students is also enhanced by the ability to exchange Fritzing files. A student who is in need of help simply sends the current state of his design to the teacher who can review it and return it along with suggestions for improvement.

Production

When it comes to exhibiting or deploying electronic artifacts, the designer needs to move beyond the breadboard. As these are so delicate and fragile, it is nearly impossible to transport them or to make them work for mobile use, which is a huge constraint to the creativity of the target group. Furthermore, the size of the prototype is dictated by the size of the breadboard. The authors have seen prototypes by students who sawed their breadboard in half make it smaller.

Another problem with breadboard-based prototypes is that they are one-offs: reproduction is expensive and complicated. This might be the reason why in the past only a few artists/designers have developed hardware concepts which consist of multiples of the same interactive object.

One approach to make prototypes more robust is the use of strip boards, an intermediary between breadboard and PCB. However, we have observed that artists/designers are hesitant to take this step as it still offers many pitfalls and requires even more technical skills that they are not willing to learn. Their wish is to have the working breadboard prototype more-or-less automatically turned into a real product, and it is not obvious to them why this should not be possible. Basically, the artist/designer is trying to focus on the aesthetic and emotional qualities of the prototype, and may not be so interested in some of the tedious nuts-and-bolts requirements.

Producing with Fritzing

Fritzing for the first time empowers non-engineers to produce a prototype as a professional without requiring them to know a great deal more than what they have already learned in breadboard prototyping. The software makes the transition to a PCB layout almost automatic. For example, if the user creates an Arduino-based project, the shape of the PCB defaults to a so-called Arduino shield that can be plugged into the Arduino.

The circuit that the user assembled in the breadboard view is reflected in the PCB view with matching footprints, and the user only needs to arrange the parts on the PCB and start the auto-routing process. Further manual corrections, such as the use of jump wires, are possible (the software may even suggest using jump wires or rearranging certain parts on the board). The auto-routing capability is limited at the moment but will improve with future versions.. More advanced users may make use of a function to export to the professional Eagle EDA software.

Once the PCB layout is finished, it can be exported for industrial- or self-manufacturing. Both processes are

documented on the Fritzing website. Self-manufacturing is relatively cheap in equipment and material. Interestingly, we have observed that design students, who are used to working with their hands, do not have problems with this process and take a lot of joy in creating their personal PCBs (Figure 8).

In effect, the designer/artist creates a robust and lasting version of his prototype, close to a product level. Its size can be greatly reduced compared to the original, and it can easily be produced in multiples.

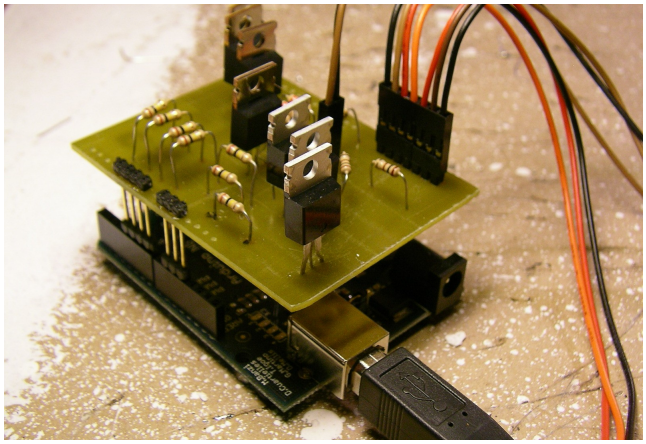

Figure 8: Arduino shield designed with Fritzing by 1st semester design student, and self-produced

RELATED WORK

As already mentioned, Fritzing attempts to follow the approach of Processing and Arduino in a different arena. They all strive to open technology to creative, non-technical people. They do not provide radically new functionality – it is rather the way that long-existing technology is made accessible, respecting the way a group of people work, and embedding it in a lively community. These tools generally follow the design principles for creative tools summarized in [9] and are discussed in relation to Fritzing in [10].

Through this orientation, Fritzing distinguishes itself from classical EDA tools made for engineers, such as Eagle [5]. While these tools provide much more functionality, they are only usable by people who are trained professionals. The commercial tool Circuit Wizard [11] at first sight takes a similar approach to Fritzing: It also has a tangible real-world feel, provides realistic simulation, and is quite advanced in the complexity it can handle. However, it fundamentally differs in the approach: First, it is oriented towards learning about basic electronics and logic circuits. Fritzing on the other hand is supporting non-engineers in their pragmatic use of electronics. Our audience is mostly working with micro-controller kits and is programming the logic rather than building it from basic logic parts. The parts they need are rather sensors and actuators, often unusual ones. Circuit Wizard does not support the creation of custom parts because the part creator would have to take care of the difficult simulation properties. For the same reason, Circuit Wizard does not offer micro-controllers or advanced sensors. A further differentiating aspect is Fritzing's strong orientation towards open community exchange. This is reflected across the whole project: it is open source, uses open standards and file formats, is cross-platform, provides a website for open sharing – and it's free.

The d.tools suite [12] is another toolkit for designers working with electronics. It uses a powerful visual editor with customized plug-and-play hardware to let users quickly sketch physical interactions. As such, we see it as complementary to Fritzing, which is more concerned with a later stage of prototyping and refinement. However, we would be interested to see how a more low-level approach such as in Arduino and Fritzing compares to that of d.tools in the context of a design process. While the latter is certainly easier to use, the former provides more flexibility and freedom and might instill a stronger appreciation of electronics as a creative material.

DEVELOPMENT

Evaluation & Feedback

Even though Fritzing is still under development, we have had several important opportunities to evaluate our software: through an international workshop with teachers for Physical Computing[3]; and through teaching undergraduates in Potsdam and Weimar.

The first version that was used throughout these evaluations was built on-top of a complex, experimental, Java-based framework which was specialized for creating diagram editors, for example flowcharts[4]. It allowed us a quick start, but it proved to be too restrictive and cumbersome in the long run to tune it to our needs. Instead of the schematic and PCB views, we built an assisted export to the Eagle software. This allowed for a quick, testable publication of our concept and was helpful to elicit early feedback.

The overall feedback from students using Fritzing was positive. We could observe that they learned quickly and were excited about the opportunities of such a tool. The students had no difficulties in copying their real life prototype into the 'Breadboard View' and found it natural to include such software into their workflow. However, for the subsequent steps they needed assistance, especially for finalizing the PCB design in Eagle. Later, in the PCB-lab they were able to work quite autonomously again.

[3] See http://fritzing.org/events/ffs08

[4] Eclipse GMF (Graphical Modeling Framework), see http://www.eclipse.org/modeling/gmf

The teachers' workshop took place in June 2008 in Potsdam. While generally highly welcoming the new tool, attendees were critical of the lower level UI design in Fritzing and the snappiness of the interaction. They made the point that no one would use Fritzing if the clunkiness of the low-level interaction interfered with the focus required to actually work through the process of designing a board. . Some of the problems were obvious, e.g., you couldn't simply drag-and-drop parts, but had to select a part like a tool, and then click to place a part; once a part was in the sketch, dragging it was painfully slow, and since auto-connection didn't work, connecting parts required careful tedious mousing; zoom was too slow.

At a higher level, the jump to Eagle was something of a shock to our users—suddenly plunging them into a different and complex UI, even with some scripted assistance, proved quite daunting.

Other requests centered around parts: Making a big library of parts available; the ability to quickly make new custom parts; quick swapping between parts; not restricting the PCB output to Arduino shields; generating a Bill of Materials; modules: the ability to treat a set of parts as a single unit - all without frightening away novice users.

A final set of requests focused on "low-level intelligence". Although a full-scale simulator was deemed unnecessary, automated help such as: warnings about short-circuits; nets not being fully routed; and help with part placement for routing were all deemed valuable additions.

Re-focus

We decided to re-write the software in order to gain more flexibility in realizing our ideas, and in order to improve the general performance of the tool. For one thing, we realized that Fritzing didn't really fit cleanly into a diagramming model—for example, unlike in a flowchart, wires have to be first-class objects that can stand on their own, without necessarily being owned by the parts they are connected to.

The new version is built on Qt, a very solid, full-featured, C++ GUI framework, which includes fast graphics, and which supports development across all three major platforms..

We also used the opportunity to consider the collected feedback, crafted a more fluent interaction, and included the alternative views. Taking care of the PCB design ourselves means enormous additional work, but it proved necessary, based on the observation that the "cognitive slap-in-the-face" that occurred in the switch to Eagle was not accepted by the users.

CONCLUSION & OUTLOOK

The first alpha version of Fritzing was downloaded over 10,000 times, which has encouraged the team to continue development of the software. The upcoming release of the new version will show if we understood the feedback.

For the further improvement of the tool, we have identified three main issues that would enhance its benefit:

- The concept of modules needs to be introduced: once a user has designed a working module, it would be helpful if others could import the entire module in one step.
- Due to the limited number of parts in Fritzing's parts library it would be possible to populate a placement machine with a fixed set of parts, and therefore reduce cost in the production of populated PCBs. However, we have not yet built a proof of concept, and we have yet to set up the entire production process.
- The overall quality of the PCB-design needs to be improved: the auto-router needs some serious optimization; there is no auto-placement of parts so far; and a smart way of exporting/importing to professional EDAs is also missing.

ACKNOWLEDGMENTS

We would like to thank the growing team of people working on Fritzing. The core team included Zach Eveland, Dirk van Oosterbosch, Brendan Howell, Mariano Crowe, Jenny Chowdhury, Jannis Leidel, and the authors, with support from Omer Yosha, Travis Robertson, Marcus Paeschke, Stefan Hermann, Myriel Milicevic, and Johannes Landstorfer.

Invaluable feedback was provided by classes at FH Potsdam and Bauhaus-University Weimar, the participants in the Fritzing summits, as well as the many users who have downloaded and tried early releases.

The work on this project was made possible through the greatly appreciated financial support of the Brandenburg Ministry of Science, Research, and Culture (MWFK).

REFERENCES

1. Reas, C, and Fry, B. *Processing: A Programming Handbook for Visual Designers and Artists*. MIT Press, 2007. Software available at http://www.processing.org
2. Mellis, D., Banzi, M., Cuartielles, D., and Igoe, T. *Arduino: An Open Electronic Prototyping Platform.* Presented at alt.CHI 2007. Toolkit available at http://www.arduino.cc
3. Hartmann, B, Doorley, S., and Klemmer, S.R. Hacking, Mashing, Gluing: Understanding Opportunistic Design. In *IEEE Pervasive Computing* 7(3), 2008.
4. Schön, D. A. *The Reflective Practitioner: How Professionals Think in Action*. New York: Basic Books, 1983.
5. EAGLE software by CadSoft. http://www.cadsoft.de

6. Qt application framework by Trolltech. http://www.trolltech.com

7. Monroy-Hernández, A. ScratchR: sharing user-generated programmable media. In *Proceedings of the 6th international Conference on interaction Design and Children*. IDC '07. ACM, New York, NY, 167-168.

8. Fischer, G. Creativity and Distributed Intelligence. In *Report of Workshop on Creativity Support Tools,* 2005, 71-73.

9. Resnick, M., et al. Design Principles for Tools to Support Creative Thinking. In *Report of Workshop on Creativity Support Tools*, 2005, 25-36.

10. Knörig, A. *Design Tools Design: How to design tools for designers, and a proposal of two new tools for the design of physical interactions*. Master Thesis, Interface Design, Univ. of Applied Sciences Potsdam, 2008.

11. Circuit Wizard software by New Wave Concepts. http://www.new-wave-concepts.com

12. Hartmann, B., Klemmer, S. R., Bernstein, M., Abdulla, L., Burr, B., Robinson-Mosher, A., and Gee, J. Reflective physical prototyping through integrated design, test, and analysis. In *Proceedings of User interface Software and Technology*. UIST '06. ACM, New York, NY, 299-308.