Sketching in Hardware and Building Interaction Design: tools, toolkits and an attitude for Interaction Designers

Camille Moussette, Umeå Institute of Design, Umeå University, Sweden, camille.moussette@dh.umu.se

Fabricio Dore, IDEO, Munich, Germany, fdore@ideo.com

Abstract

In this paper, we present a *Sketching in Hardware* perspective to Interaction Design (IxD) education and practice. We start our discussion by highlighting the differences between *Prototypes* and *Sketches*, and explaining why we believe the term *Sketching in Hardware* is suitable and appropriate to the IxD practice. We introduce a short history of the term and its origins before relating it to Experience Prototyping activities and other related design processes/methodologies.

Our main discourse consists of observations and a critical analysis of academic activities and professional work suggesting that *Sketching in Hardware* remains quite challenging despite the recent progress in the development of new tools and toolkits. The low barrier to entry and the explosion of tools and toolkits are very welcome, but this democratization can also be misleading. The learning curve is still steep in many ways. The current sketching tools seem to have leapfrogged our design skills and our ability to deal with that avalanche of technical capabilities. Designers regularly loose a critical perspective on their sketching and prototyping activities. We noted that students and designers alike spend a lot of time mastering intricate tools and debugging technical issues when they should be developing, evolving and fine-tuning interesting experiences or sketches informing their design process.

We close our discussion with a review of various toolkits and building blocks currently available to interaction designers for designing new technology and future concepts. We ultimately suggest five guiding principles to be taken into account in the design of new toolkits or upgrading of existing ones. These same principles and qualities not only can, but should also radiate in the experiential qualities, well beyond the built material artifacts. *Sketching in Hardware* is not just playing with electronics; it has serious implications and repercussions in the way we design stuff.

Keywords

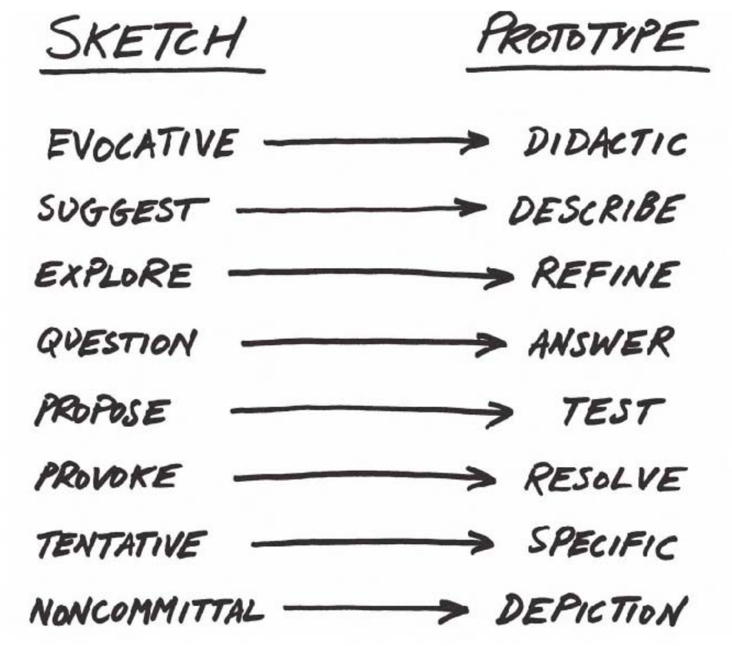
Interaction Design, Prototypes, Prototyping, Sketches, Sketching in Hardware, Toolkits, Experience, Design Tools.

Prototyping is a core aspect of design activities. As a loose definition, prototypes are manifestations of a design made before the final artefacts exist. Many authors have written about the various forms and variations of prototyping activities (Buchenau & Suri, 2000; Buxton, 2007; Houde & Hill, 1997; Lim, Stolterman & Tenenberg, 2008; Shön, 1982). They often diverge in the specifics but globally, they describe the same design activities used to filter a design-space and build corresponding representations. In the field of Interaction Design (IxD), such design activities are often encompassed in terms like Experience Prototyping, Hardware

Prototyping and Physical Computing. Our main objective with this paper is to present a new perspective on sketching in Interaction Design and describe our view of how students and designers go about it.

Sketches are not Prototypes

A few authors like McCullough (1996) and Buxton (2007) point out the importance of the mediums and that prototypes are different from sketches. Sketches, in any mediums, have different qualities and purposes. Without simplifying too much, sketches are generally used and valued earlier in the design process, while prototypes are more beneficial towards the later stages of the process. These two representations have their own specific role, intent, qualities and purposes in a designer's toolbox. Buxton (2007) proposes this comparison or continuum (Fig. 1). In our view, while this polarization can be useful to discuss and understand the differences, most design manifestations sit somewhere in the middle, depending on the context and situation they originated from.



SKETCH	PROTOTYPE
EVOCATIVE	DIDACTIC
SUGGEST	DESCRIBE
EXPLORE	REFINE
QUESTION	ANSWER
PROPOSE	TEST
PROVOKE	RESOLVE
TENTATIVE	SPECIFIC
NONCOMMITTAL	DEPICTION

Figure 1. The Sketch to Prototype Continuum (Buxton 2007)

Prototyping is a term widely used in many domains outside design. In software development, a prototype is generally understood as a working proof of concept, a partial instantiation of the final application where some aspects are usable for testing or internal reviews with client, users and contractors (Floyd, 1984; Wikipedia, 2010). The focus is clearly towards showcasing solutions, evaluating options and agreeing on specifications, all of which are situated in the later stages of development. Although newer approaches and methods are breaking this tradition (agile method), prototyping in software development involves very little ideation and explorative and noncommittal activities.

In Industrial Design and Product Engineering, a prototype corresponds often to the latest instantiation before going to final production (Feirer, 2002). All problems are solved at this stage and the prototype is (ideally) an one-off copy of the upcoming production item. What comes before the prototype adopts a variety of names depending on the use and purpose: exploration models, volumetric or form study models, visual models, functional models, proof-of-principle models, engineering models, throwaway prototypes, concept prototypes, etc. Sketches are also well understood and highly valued in Industrial Design. They are fast, explorative and tentative representations to exteriorize and communicate ideas. Additionally, the self-reflective qualities of sketching are often more important than the resulting sketches themselves (Hanks & Belliston, 1990). Numerous authors have highlighted the crucial underlying processes of sketching that support creative thinking (Schön & Wiggins, 1992).

From our perspective, we believe '*sketching*' is an important term that designers and design students should be using more when referring to design activities that relate to evolving and building interaction ideas creatively. Although not perfect, the term '*Sketching in Hardware*' reflects well the action of sketching in any mediums (with or without computing capabilities) well beyond pen and paper. Compared to '*prototyping*', '*sketching*' feels definitely more appropriate for evolving ideas and developing design concepts. To our knowledge of educational curriculums and today's practice in design consultancies, very few designers work in the later phases of the design process: close to mass manufacturing and production, where 'real' prototypes are commonly found. These activities tend to be very technical and are naturally best handled by competent engineers and professional builders.

History of Sketching in Hardware

The term *Sketching in Hardware* is not exactly new. Researchers like Holmquist (2006) have introduced the term some years ago, and it is somewhat present in today's IxD teaching curriculum and professional practice.

It is important to note that although the term is relatively new, the activities and motivations have been around for decades. Companies, design consultancies and research groups have been developing interactive physical products or projects with a strong focus on early electronic sketches or half-faked interactive demos (Fitzmaurice, 1993). Almost a decade ago, Greenberg & Fitchett (2001) introduced Phidgets as a toolkit to support, build and test physical interfaces quickly. During the following years, the ideas and tools evolved rapidly, left the research labs and gained widespread adoption in design school and consultancies. The term '*Experience Prototyping*' is well recognized today in the design community and has strong roots with the user-centered design movement (Buchenau & Suri, 2000). We agree that both *Experience Prototyping* and *Sketching in Hardware* have a lot in common and are almost interchangeable. Our intention with this paper is not to argue over exact definitions, but more trying to describe, understand and enhance the design activities taking place. Let us just finish by mentioning that both terms complement each other very well, *Sketching in Hardware,* due to it's strong sketchy, almost messy and more designerly connotations, has the benefit of emphasizing the material and experimental qualities, whereas Experience Prototyping highlights that the goal is to design and support the experience and not the prototype (or the built artefact).

Sketching and building Interaction Design differently

Technical know-how can be a blessing or a curse

From our teaching and tutoring experience (as well as being students ourselves), we observed that students are often mesmerized by the new technological and technical possibilities. Discovering new technology and being able to appropriate it to build something that works (partially at least) is usually very satisfying and gratifying. It is somewhat natural to be in that 'wow phase'.

But designers have to move on and make good and meaningful use of those tools. As good craftsmen, designers should avoid pursuing a project (or prototyping) relentlessly to a point where it becomes self-conscious demonstration and point-less perfectionism (Sennett, 2008). Great and seductive demos and prototypes surely have their value in the design process, "but they are only one part of a means to an end, and certainly not the end in itself" (Buxton, 2007, p. 412)

However, the skills and knowledge required to build hardware sketches are difficult to outline, mostly because the nature of the work spans many disciplines and domains. Technical know-how in electronics, mechanical systems, programming, sensors and the like are absolutely helpful, but so is knowledge in improvisation, performance arts, storytelling, communication, movie making and psychology. In our opinion, it is not necessarily the sketch you make that counts, it is how you use it to inform your design activity.

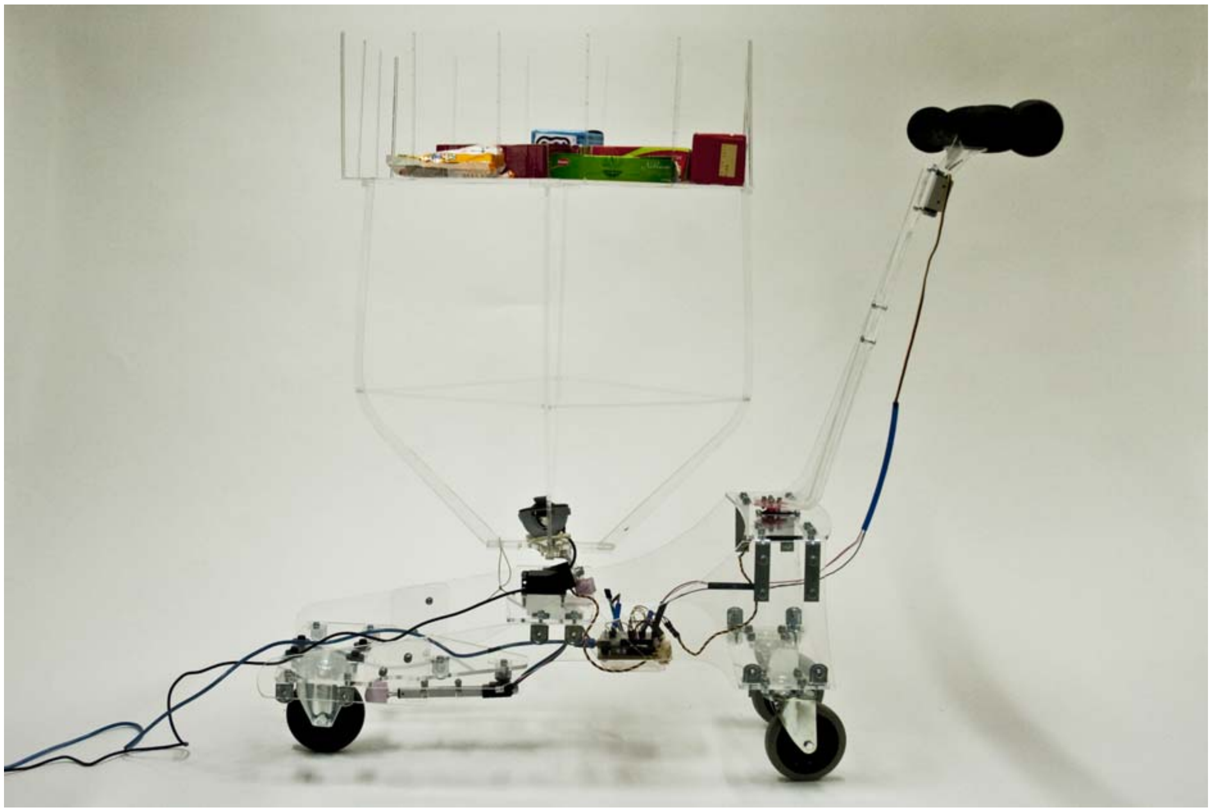


Figure 2 Shopping Cart prototype, an example of over-prototyping (image credit: George Paravantes and Amid Moradganjeh)

Throughout the courses we teach in Experience Prototyping and Sketching in Hardware, we repeatedly have technically-inclined students who have difficulties balancing technical prowess and the actual design outcomes they get from building hardware sketches. They build technically impressive stuff "because they can". Despite clear learning benefits, we feel these projects are missing core characteristics of *Sketching in Hardware*, namely the noncommittal, rapid and disposal aspects. Whatever comes out of these sketches has usually very little to do with the actual design project they are associated with. Impressive: sometimes; informative to the design project: often no. Figure 2 shows an example of an ‘over-prototyped’ project realized by two interaction students at the Umeå Institute of Design. The realization of the hardware sketch of a shopping cart represents a substantial amount of work. It never worked properly and was never presented to users or other participants. The remaining of their project was terribly well executed and coherent. In short, it didn’t help or support their design project significantly. We can see in it a perfect example of a prototype realized just for the sake of prototyping.

At the other end of the spectrum, designers and students with no programming or electronics knowledge have also their share of challenges. Although quite approachable, most tools or toolkits still require some understanding of programming, code and electronics. Without these skills, it is very difficult to evolve beyond ready-made examples and pre-caned configurations. Students often feel frustrated when they cannot get anything interactive working on their own. We understand their frustration but encourage them to try different routes or attack smaller and more manageable problems. What these students are generally very good at is finding alternatives and detours around problems, to eventually reach their intended goal (or a different one).

We see this ‘problem-solving with detours’ as a highly valuable skill in Sketching in Hardware because it is humanely impossible to know everything. Everyone has his/her limits. How one manages to keep moving forward despite these hurdles and constraints is in our opinion admirable. We could call this cleverness or pragmatic sketching.

In an ideal world, students and designers should have the perfect mix of technical skills and creative approaches to evolved complex but useful and informative hardware sketches. There are no magic recipes or prescriptive notions that dictate how one should go about building hardware sketches and using them in a particular project. It is a life-long (or career-long) learning experience.

Bringing back the Experience

Prioritizing design outcomes over technical know-how

The current prototyping toolkits use technology about 10 to 20 years old, if not more. Compared to what can be found in the latest electronic devices today, they are dinosaurs. From an engineering perspective they are almost laughable. Therefore, having something that works using those toolkits might be personally gratifying, but from a technical standpoint it is not exactly groundbreaking. The real value for designers and other collaborators in toolkits comes from leveraging their simplicity, mixing and remixing technology with creative processes, and most importantly aiming on the experiential outcomes and not so much on the process of ‘making it work’.

We have seen that in other disciplines like Industrial Design (ID), doodles or volumetric models fulfill different functions during the design process. The vocabulary in ID is not extremely sharp, but decades of activities have laid considerable foundations to describe processes and methodologies that are commonly agreed upon. The Interaction Design discipline is in its infancy. Our current shared vocabulary, mostly adopted from other disciplines, often falls short to properly describe the dynamic nature of our sketches and interactive prototypes. We usually try to minimize this gap the best we can by using a variety of mediums and interactive depictions (like hardware sketches). Without a clear vocabulary and understanding that intuitively connects intent and outcomes, designers can sometimes find themselves focusing on building something that works and not explicitly identifying why they are sketching for.

In our view it is important to consider that hardware sketches can also work as tangible and intellectual springboards towards other disciplines. They help the designer play in other disciplines' sandboxes, and learn how to better communicate their design ideas in different technical jargons. Finally, hardware sketches and experience prototypes are key components in establishing shared reference points with colleagues, users and other collaborators. They naturally invite others to relate and build on something (half) real, tangible and concrete.

Toolkits: technology building blocks for Interaction Designers

A toolkit has to be useful and helpful to its users: that is easier said than done. What is useful to reach these *sketching in hardware* objectives? The big answer is: it depends! As we highlighted above, each project is unique and the designer's skills and understanding of hardware sketches are constantly changing. Toolkits need to balance flexibility and simplicity. What is useful for a first-time user can be very different for an advanced user. An advanced user might enjoy very basic features if pressed by time or under certain condition/mindset. So it is never black or white. It's all grey!

In our quest towards a better education curriculum and a more enjoyable IxD practice, we decided to reflect on what would be, in our view, the ideal toolkit for interaction designers. We quickly came to the conclusion that it is futile to think that such unique magic-bullet toolkit exists. If we look back at history, we notice that a multiplicity of tools and toolkits will always prevail, some very highly specialized ones, others more multipurpose and aimed for general use.

Despite this failed intellectual attempt, we believe that current toolkits are not always well suited for Sketching in Hardware activities. In regards to the current status of the design practice, we think there is definitely room for improvement in many areas. Students and designers are often struggling on the same issues over and over again. We think new or improved tools and toolkits could be developed to better match today's needs in sketching in hardware activities. These beliefs come from our experience in teaching and using such tools ourselves, but they are also inline with similar discourses from numerous researchers and teachers in the field of Interaction Design (Buxton, 2007; Moussette 2007; Saffer, 2009)

We identified five high level characteristics or qualities that should be supported or at least considered when developing new toolkits for interaction designers. They are presented in no particular order of importance. We try to support each point with concrete propositions, examples or critique found in today's toolset.

Openness and level of visibility/accessibility

Just like Nature, complex and elaborate systems are made of simple underlying parts or processes. Accessing and peeking at how things work or are composed can be very beneficial to develop a good understanding.

The goal is not to expose and decompose all processes, but allow for different levels of visibility/accessibility. Complex processes can be encapsulated in simple representations/actions to ease introduction and support high-level perspective/work. The inner working processes should be accessible if one is interested to dig in. Some complexity can be exposed to invite or tease the user to explore more. The level of details cannot be infinite, but documentation should be as good as possible. Knowing that one can further explore references, sources, libraries and other internal details is very reassuring, even though very few are actually doing it. Open-source software and hardware like Processing and Arduino exist and are successful because of the all-embracing commitment to this openness principle.

On a more graphical level, applications like LabVIEW, VVVV, MaxMSP and Pure Data offer simplified graphical dataflow programming environments where encapsulation is obvious. Advanced and curious users can access the functional code behind each individual graphical node. Expert users can often build new functionalities into the tool if the underlying processes are exposed properly.

Each design problem can be said to be unique. Manufacturers and toolmakers cannot realistically envision every possible use or situation for their tools. With open tools, designers and users can fully embrace the system, and effectively help the producers to refine or enhance their offerings. Note that simplicity or reduction in complexity doesn’t translate to simple or trivial work. Some level of abstraction is necessary for humans to cope with complex task or endeavors.

Hackability

By hackability, we mean that parts, modules, or the whole toolkit should be open for unintended and unplanned uses. While manufacturers cannot obviously support these activities, the design or functions should not thwart those who feel adventurous. For example, parts from one toolkit should be usable with other toolkits without major difficulties.

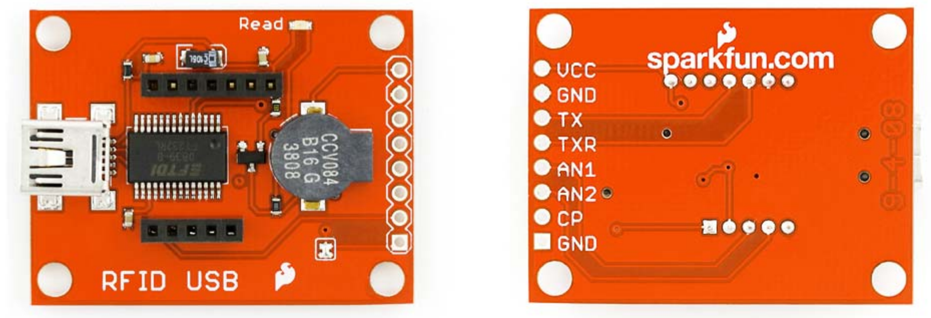


Figure 3 Sparkfun's RFID module, with numerous exposed connectors. Ready for repurposing or hacking (image credit: Sparkfun.com)

Sparfun’s model (Fig 3) provides breakout pins for wiring extra parts or using the module with other than the default USB port. By exposing the already-available functions, one can repurpose it more easily. The module from Sparkfun uses a very ubiquitous technology (RS232 serial communication) to transmit its data. The design is intentionally simple. This simplicity has a cost. The module is quite dumb and only send data values. Once attached to a computer, it cannot announce its functionalities and share other secondary status or auto-configuration data.

Figure 4 Phidgets RFID module. Friendly but a closed platform, on the hardware side at least. (image credit: Phidgets.com)

The Phidgets module (Fig. 4) works with a special set of hardware and software combination and offers advanced software functionalities (notification when the RFID tag is removed). This tighter integration can be a double edge sword: it is simple and works very well as long as you work within the limits offered by the manufacturer. Outside this area, it is very difficult if not impossible to reuse the module and combine it in your own ways. For example, it is impossible to use with a dedicated microcontroller due to the USB connectivity. By design, it has to be plug into a computer to work, despite the low computation power required to send and receive RFID tag values.

Added value when time is limited

The ability to get going quickly and get results fast, within minutes if often crucial in the design practice. In educational settings the time constraints are not so present and crucial. Nevertheless, the focus or the work should be elaborating on variations of the experiences, not on debugging technical issues.

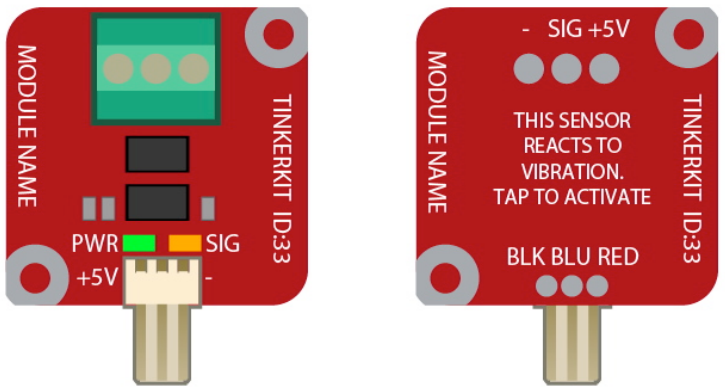


Figure 5 A proposal for new TinkerKit modules, with emphasis on labels and at glance information.

Labels, diagrams, numbering and details on the device are crucial for novice users, or users under stress or pressure. These details usually incur no additional hardware costs, but they do require extra care and attention by the people who develop the tools. What can be evident to an electronic engineer (designer/builder of such modules) might not be obvious to an interaction designer. Having to download and look up a PDF document just to find how much voltage is needed to power a device is frustrating and not efficient.

At IDEO and presumably in many other design consultancies, the most challenging sketching/prototyping issues lay beyond the technical skills of its designers. The difficulties reside in how to balance *build-to-learn* – from the medium or experimentation – and *build-to-gather* rich user feedback within a constrained timeframe. Projects last on average 12 to 16 weeks. The team needs to define the problem, come up with innovative concepts and provide deliverables that have the potential to outlive inside the clients' organizations and ultimately impact their product portfolio and business.

Spending anything more than a few days building something is a luxury and need to be well justified even within a company with a strong culture of prototyping like IDEO (Buchenau & Suri, 2000). Prioritization is key and there isn't much time available for experimentation. Low-fidelity prototypes are still useful to gather initial feedback but once we move towards designing the user experience, the fidelity needs to be tuned up immediately for two reasons. First, because the user experience usually means 'behaviors, transitions or gestures' that might have never been done before; second, because users are increasingly technology and user-experience savvy. Obtaining a 'suspension of disbelief' when presenting pen and paper concepts for electronics devices is becoming harder and harder.

From our experience in practice, only a few (2-3) serious prototypes/hardware sketches are realized for a particular project. Their aim is to have the most impact in the overall design, or present extreme concepts that occupy opposite places on a scale (e.g. knobs and levers vs touch screen and voice commands) and that can be used to align the designers' intents with real user expectations.

No matter how diverse the project challenge is, three points or needs consistently show up when developing hardware sketches under stringent time constraints. For us, the ideal tools or toolkits would:

- allow the designers to experiment different versions of the 'experience',
- build something convincing to be presented in user feedback sessions and
- allow for improvements and modifications at a later point in time.

Versatility or 5 ways of doing the same thing

Design is all about exploring options and possibilities. The toolkits shouldn't force the users into a monolithic approach and impose an unique way to solve a particular problem.

We find value in approaching a problem from many angles and perspectives. Building or sketching interaction can take so many forms. The continuum from a highly specialized tool to a totally generic device is vast.

If we take the case of Phidgets, designers have a large choice of programming languages and platforms to develop and build their ideas. An expert user can use C++ to configure and interact with the devices, while a novice designer might use Actionscript or Java instead. The manufacturer of the system supports both equally.

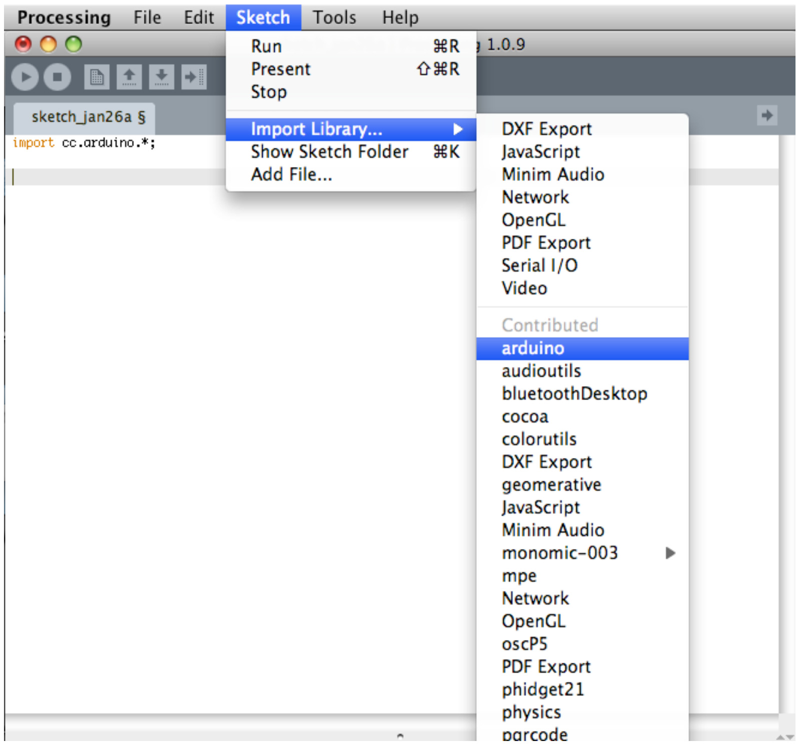
On the hardware side, electronics modules can "speak" many protocols and adapt to various host systems. For example, a particular distance sensor can work in analog mode for simplicity and out-of-the-box use without reading much documentation. The same sensor can also be connected using a serial protocol in order to reconfigure it or send more complex commands/queries. This versatility comes with some drawbacks (the device in inherently more complex), but does offer flexibility and some form adaptability.

Generally, exploring options and possibilities will involve more than one system or toolkit. Mixing and matching different components in new and creative ways help designers the space of possibilities. We appreciate tools and systems where designers can come in and apply knowledge they already acquired previously. Each tool has its limitation and requirements, but often a knowledgeable user will be able to quickly get up to speed if the tool follows best practices and general conventions. Having to learn new ways of working can be frustrating and time consuming.

Human friendly

Technology should speaks for itself, ‘transpires’ its status or ways of working. Power lights, communication status, functions broadcast, non-symmetrical connections are small details that make the life of designers so much more enjoyable (or less painful).

One recent example that our students have been struggling with is the new ActionScript 3.0 language from Adobe. When writing code in the Flash application, one has to include and specify various libraries to add functionalities to the interactive application. The application requires you to “include” and type the exact name of the libraries you want and need to use. For an experienced programmer, this might be standard or obvious procedure, but for a designer this is quite tricky. How one should know which libraries to include and what is the exact name for it, even if it was never encountered before? No default choice, pre-made lists or options are offered to the user. You have to look deep down in the documentation to find that information, and manually type the information in your document. It is not very straightforward and satisfying indeed. You inevitably learn the most common ones quickly.



```
import cc.arduino.*;
```

Figure 6 Processing application and the available libraries selection.

If we compare this experience to developing software sketches in the Processing application, the task is almost the same, but the application in this case provides various cues and selections to choose from. First the application has default libraries covering the core functionalities, so you do not have to write anything for the standard libraries. Second, if you need to use a specialized library or set of functions, a menu entry lists all the installed and available additional libraries. In one or two clicks, one can choose, include and readily use new functionalities in your sketch (Fig 6). We find this procedure definitely more humane, at least for novice and casual programmers like us.

On the hardware side, we strongly encourage the widespread use of status lights to communicate power status and activity right at the hardware layer. It is much easier to track down problems with blinking LED (or absence of it) instead of writing custom debugging code in software. A quick glance at a power status light will let you know if the device is properly powered. Physical affordance can definitely help too. Non-symmetrical connectors enforce the right polarity when connecting devices and cables. They often incur no or ridiculously low additional costs (fraction of a cent) in parts. Small details like this make huge difference in the long run. All humans make errors, even the highly trained ones.

Discussion

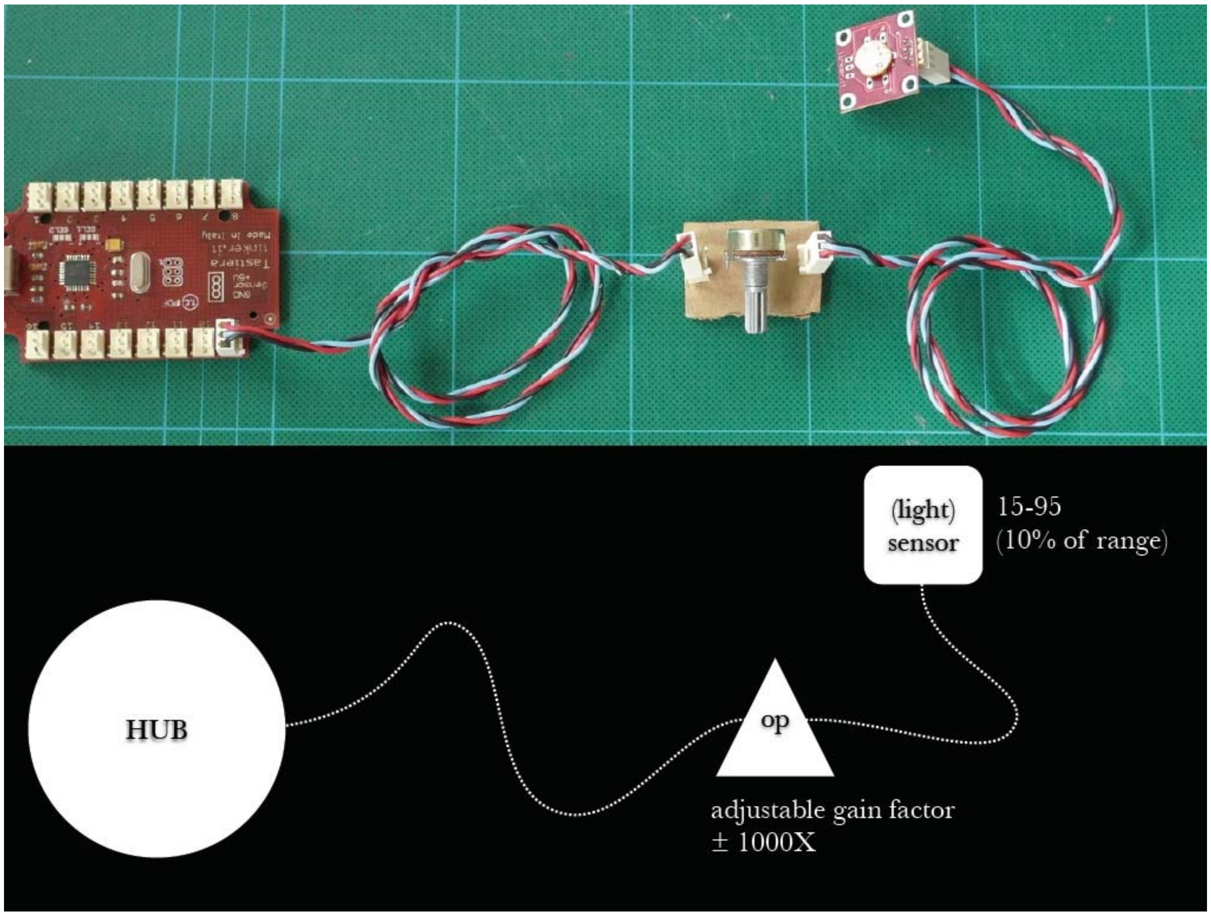


Figure 7 A non-working depiction of inline operator module.

Our work is currently ongoing but we are actively developing instances of tools where these values or qualities are put forward. We are building on previous tools and toolkits, leveraging previous successes and de facto best practices and standards. Figure 7 shows an early sketch of inline operators that provide a tangible control over sensors and various modules. The operator allows one to explore and adapt to a much wider spectrum of values, without changing code or sensor module. It supports expanded design explorations in a simple tangible way.

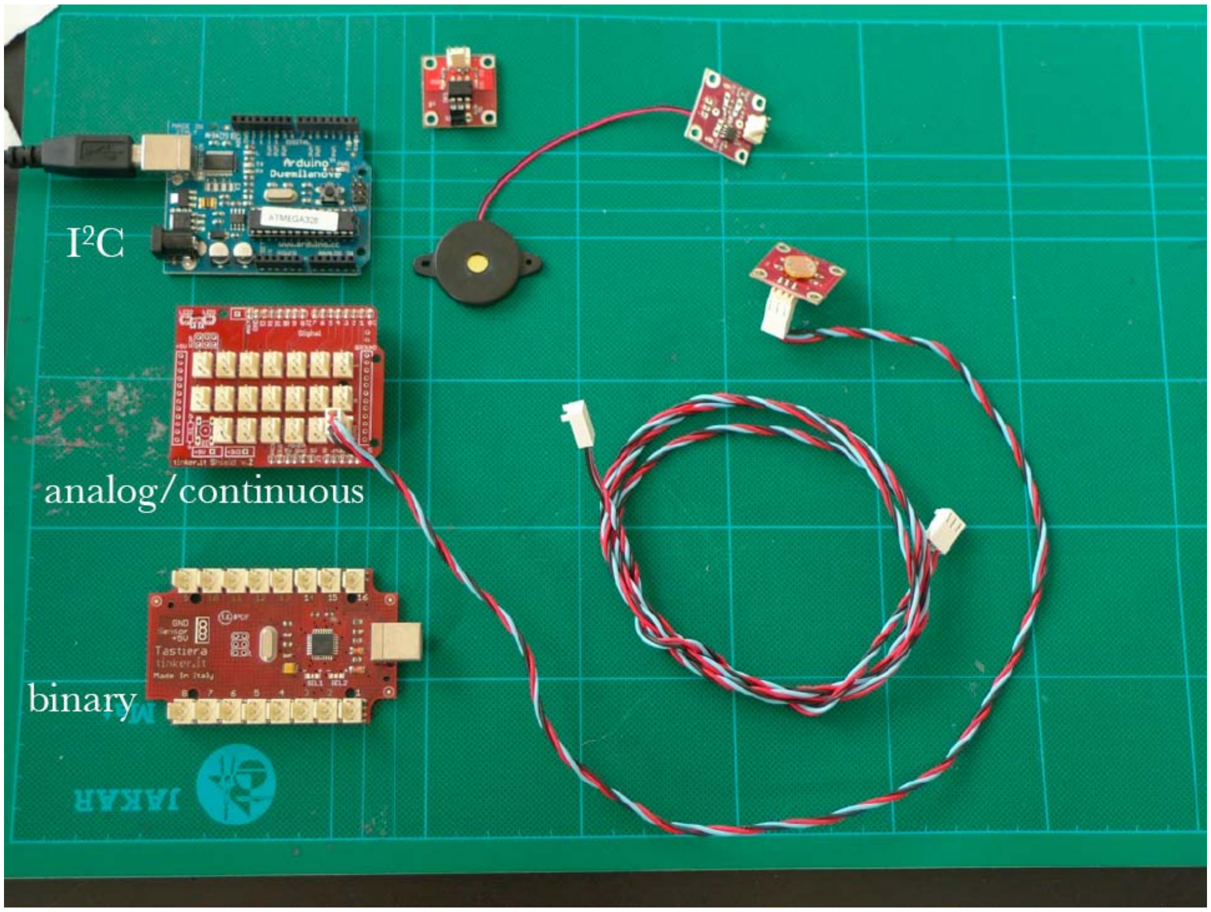


Figure 8 Smart sensors/modules can be used with different host systems

The availability of faster and more powerful microcontrollers means we can build “smart” toolkits and sensors more easily. When connected, modules broadcast their functionalities and requirements, and then establish the most desirable protocol of connection or communication. Simple analog parts can be mixed and combined with fancier or more complex ones. Older parts and items can be used without worrying about compatibility and interoperability. Designers can focus more on the outcomes of their sketching, and less on the technical side of things. Figure 8 shows how a particular smart light sensor would work depending on the capability of the host system. It would transmit configuration and status data only if the host module can receive and understand it. Otherwise, the smart light sensor would revert to a simpler analog mode, or even to a binary mode (on/off) if necessary.

This is work in very early stages. We hope to refine our ideas and sketches further, and evaluate them with design students and practicing professionals in design consultancies.

Conclusion

With this paper, our main objective was to present the *Sketching in Hardware* perspective in the field of Interaction Design, and explain our view and experience around those somehow new ways of building or sketching ideas. We observed and noted numerous challenges, but also very rich and stimulating design activities. Despite our criticisms, we strongly believe in this new perspective and its potential for establishing a strong understanding of interaction concepts, solidifying prototyping knowledge and taking sketching activities to a new level. Designers have to know their limits and when it is advisable to ask other professionals. *Sketching in Hardware* is not just playing with electronics; it has serious implications and repercussions.

Ultimately there is an open discussion that pervades the professional practice of (interaction) design about whether designers should focus on being excellent craftsmen of human-to-human and human-to-device interactions, or should designers excel as good design generalists ready to take on challenges a few levels up in abstraction (the big picture). It is our belief that one cannot live without the other. Some designers will go deep into the technical side and into the art of crafting intricate interactions, while others will focus on high-level themes and global agendas. We hope those sketchers and visionary designers can together build a better and more enjoyable world.

References

Buchenau, M. and Suri, J. F. (2000) Experience prototyping. *In Proceedings of the Conference on Designing interactive Systems: Processes, Practices, Methods, and Techniques* (New York City, New York, United States, August 17 - 19, 2000). D. Boyarski and W. A. Kellogg, Eds. DIS ‘00. ACM Press, New York, NY, 424-433.

Buxton, Bill, (2007). *Sketching User Experiences: Getting the Design Right and the Right Design,* Morgan Kaufmann.

Dore, F. (2009). Sketching in Hardware is Chaning Your Life, Core77.com, retrieved Jan 5 2010, from http://www.core77.com/blog/featured_items/sketching_in_hardware_is_changing_your_life_by_fabricio_dore__14769.asp

Fitzmaurice, G. W. (1993). Situated information spaces and spatially aware palmtop computers. Communications of the ACM (CACM), 36 (7), 39-49

Floyd, C. 1984. *A systematic look at prototyping*. In Approaches to Prototyping, Budde, R., Kuhlenkamp, K., Mathiassen, L., and Zullighoven, H. Eds. Springer-Verlag, Berlin, Germany, 1–18.

Feirer, M. (2002) Prototypes in Industrial Design: Once a Design Goes Digital, All Sorts of Things Are Possible, *The Technology Teacher*, Vol. 61 (2002), pp. 24-29.

Greenberg, S. & Fitchett, C. (2001). Phidgets: easy development of physical interfaces through physical widgets, *In UIST ’01: Proceedings of the 14th annual ACM symposium on User interface software and technology*, ACM Press, 209-218.

Hanks, K. & Belliston, L. (1990). *Rapid Viz: A new method for the rapid visualization of ideas*. Menlo Park, California. Crisp Publications.

Holmquist, L. E. (2006) Sketching in hardware. *Interactions 13*, 1 (Jan. 2006), 47-60.

Houde, S., & Hill, C. (1997). What do prototypes prototype?, *in Handbook of Human-Computer Interaction (2nd Ed.)*, Elsevier Science B.V: Amsterda.

Lim, Y.-K., Stolterman, E., and Tenenberg, J. (2008). The anatomy of prototypes: Prototypes as filters, prototypes as manifestations of design ideas. In *ACM Transactions on Computer-Human Interaction (TOCHI)*. 15,2, Article 7 (July 2008), 27 pages.

McCullough, M. (1996). *Abstracting craft: The practiced digital hand*. Cambridge, MA: MIT Press.

Moussette, C. (2007). *Tangible interaction toolkits for designers*. Scandinavian Student Interaction Design Research Conference 2007, Sweden.

Norman, D. (2009). *Technology First, Needs Last*, retrieved Jan 25, 2010, from http://www.jnd.org/dn.mss/technology_first_needs_last.html

Saffer, D. (2008). *Designing Gestural Interfaces*. O'Reilly Media, Inc, USA

Schön, D. A. (1982). *The Reflective Practitioner: How Professionals Think in Action.* Harper Collins, New York, NY.

Schön, D. A. & Wiggins, G. (1992). Kind of seeing and their function in designing. *Design Studies*, 13(2), 135-156.

Sennett, R. (2008). *The Crafstman*.Yale University Press, London.

Sketching in Hardware (2006), *Sketching in Hardware 1: a summit on the design of/with physical computing toolkits*, retrieved Jan 7 2010, from http://www.sketching06.com

Software Prototyping (Wikipedia), retrieved Jan 25, 2010, from http://en.wikipedia.org/wiki/Software_prototyping