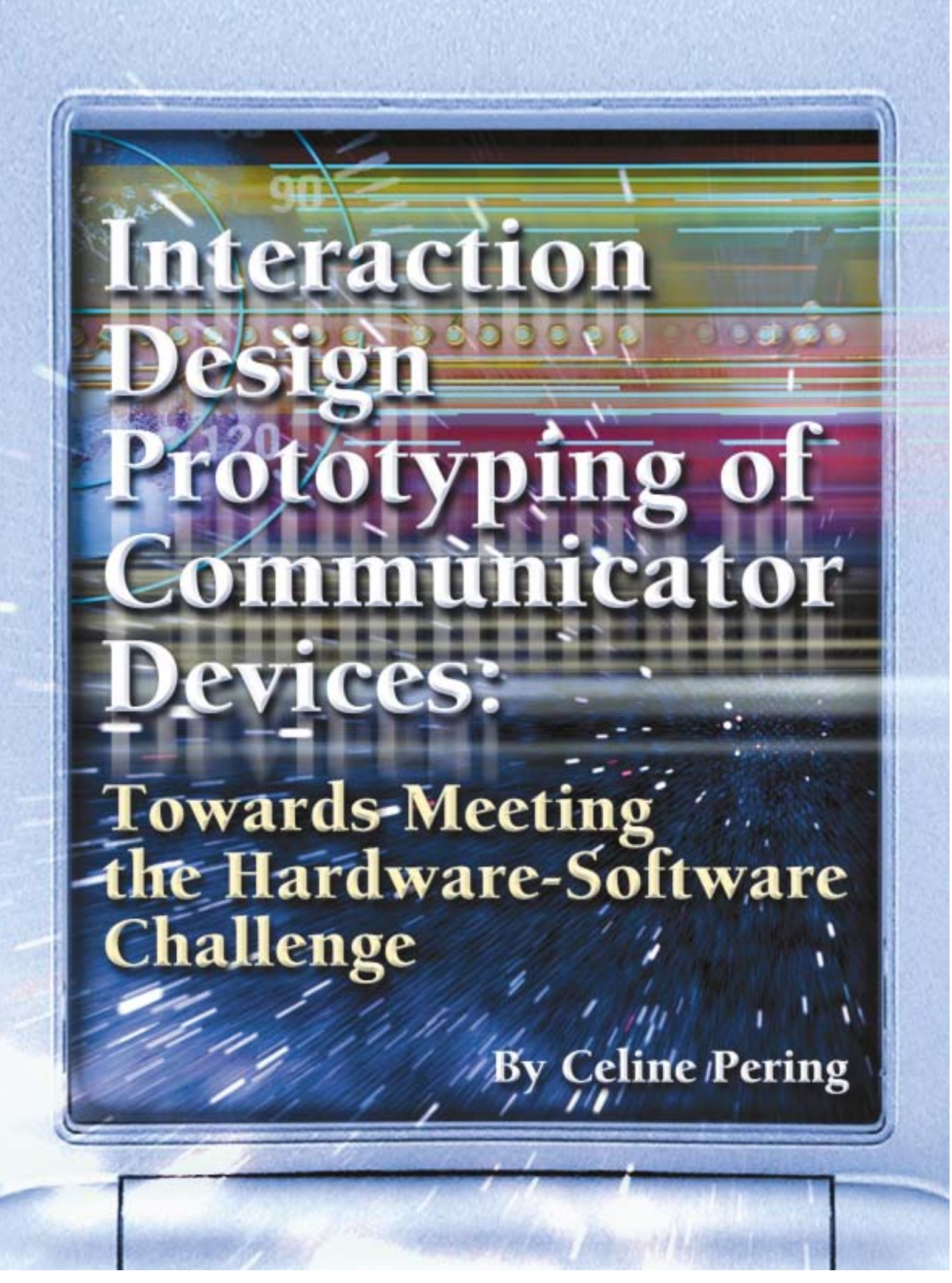
Interaction Design Prototyping of Communicator Devices:
Towards Meeting the Hardware-Software Challenge
By Celine Pering

Developing a Strategic Framework for Prototyping Hardware-Software Interaction

Celine Pering
Handspring, Inc.
cpering@stanfordalumni.org

Handspring's communicators are a series of innovative products that require considerable upfront design investment to develop because they combine the functions of both a cell phone and a personal digital assistant (PDA). Communicators are handheld computers that have both a standard keyboard and a touch screen as input and are small enough to fit in a shirt pocket. Interaction prototyping is especially interesting because it requires merging two different design paths (cell phone and PDA).

Through the design process, we have learned that hardware-software interaction prototyping is a challenge that is more complex than other types of prototyping. For example, prototyping interaction in a software application can assume a standard keyboard and mouse as controls. Similarly, prototyping hardware products, such as toasters, is relatively simple because they generally don't require any navigation and they are tangible embodiments, not abstract concepts. In contrast, hardware-software prototyping is challenging because it involves complex software functionality that needs to be integrated with customized physical controls and interaction. As wireless and embedded computing becomes more widespread within the consumer market, the need to overcome these hardware-software prototyping challenges will increase.

Prototyping the hardware-software system is further complicated because it requires bringing together two different organizations in the development process: product design and product marketing. Each of these organizations has its own established approach to both design and prototyping. By working cross-organizationally, hardware-software prototyping addresses significant issues that would have otherwise been missed by the departments working individually.

The Buck is a physical prototyping device designed to address the hardware-software interface. This term evolved within Handspring to refer to the functional physical model. This system has many advantages over paper, software-only, or hardware-only prototyping that we will explore in this article. In the following discussion, we will describe an overview of the interaction prototyping process, the product lifecycle constraints we started with, how we address those issues by selecting prototyping tools, and the user testing process. Further, we will share what lessons we learned and what opportunities exist for developing future interaction prototyping tools.

Interaction Prototyping: Introducing the Buck

At Handspring, prototyping the hardware-software interface has several main components to it: Paper, Screen, Buck, and Alpha (see Figure 1). These processes represent only a subset of the complete product lifecycle; different parts of the product have additional processes (e.g., industrial design) and are not represented in this figure.

Prototyping processes vary across companies. Large companies that need to coordinate lots of information and many different types of users are often specification-driven, whereas small entrepreneurial companies are more frequently prototype-driven. Some companies have a more formal process, and others have an informal prototyping process. Further, certain companies have a fixed number of prototypes per product cycle, and others prototype more opportunistically. Prototyping media also range according to industry; for example, the automobile industry uses tools such as sketches, computer-aided-design drawings, and clay forms to model ideas [11]. In general, prototyping has been commonly used in hardware development environments; in software development it is relatively new [1]. Lastly, some companies have a unified design group and others have separate design groups broken down by product divisions [9].

Handspring shares many aspects of these other processes and cultures. The company has both specification and prototyping cultures, which fall along department lines. The hardware product design team prototypes more often and more formally than the team that designs the software user interface. Handspring is, however, different from other companies in that it does not have a single unified design group; the two groups exist in different organizations. This is also different from larger companies that have design groups dedicated to product divisions. Handspring is a relatively small company, of around 400 people, and its emphasis is on both hardware and software. In addition, Handspring's interaction design process more closely resembles a software process in that the prototyping media are predominantly digitally created. For example, no sketches,

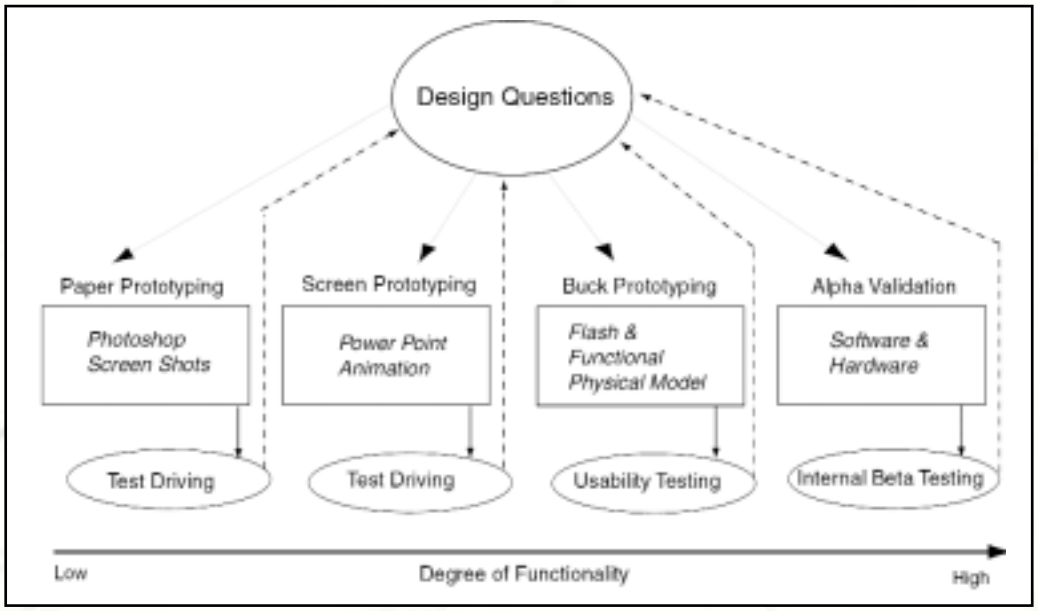

Figure 1. Interaction prototyping of hardware–software devices. Design questions are answered iteratively with the technique that best addresses them. Each approach leverages a different prototyping tool with an increasing degree of functionality (shown in italics with the square) and requires different types of user evaluation (shown within the ellipses).

foam, or clay model techniques are used. This diversity of process and culture within Handspring means that our prototypes frequently provide a *lingua franca* that allows people involved in the design process with a useful mechanism to effectively communicate [3].

Paper Prototyping

Based on a specification, quick paper prototyping is used initially to provide a visually explicit representation of a design. To illustrate the interaction, a user interface map is created that illustrates a nonlinear storyboard of user interaction possibilities (see Figures 2a & 2b). Arrows connect the screens and communicate the interaction paths that be might taken by a user who clicked a given button. These prototypes are used for test driving by the design team, as distinct from usability testing, which in contrast involves people who are not involved in the design process. This method also provided a useful technique for heuristic evaluations [8] and cognitive walkthroughs [10] through the design, and helps one catch problems with organization, terminology, semantics, and layout. These paper prototypes also have the added advantage that the screen shots can be added back into the specification as the plan of record.

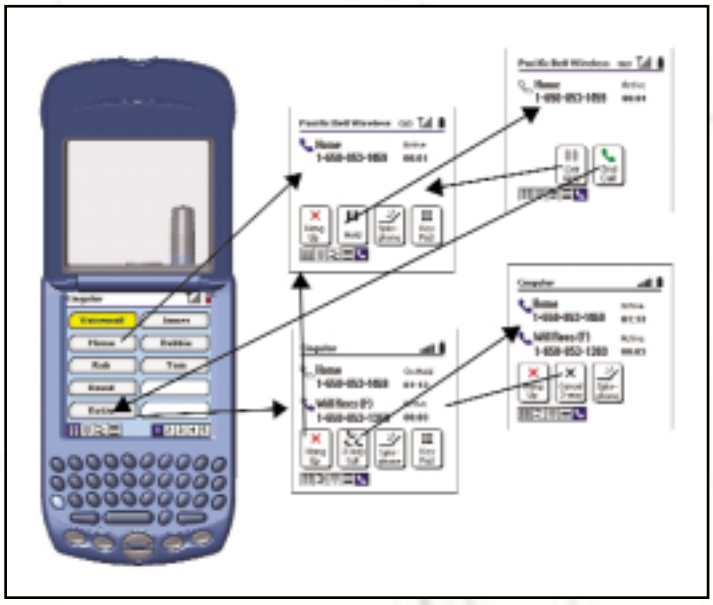

Figure 2a. Paper prototyping. In this example, arrows indicate navigation paths and highlighting is explicitly demonstrated so that designers can test-drive their proposals in design meetings.

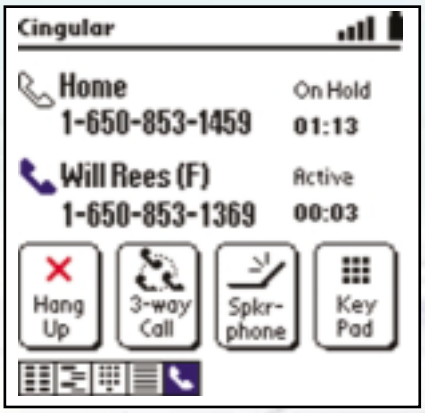

Figure 2b. Screen shot. Screen shots created in Adobe Photoshop are used in the paper prototyping process to walk through design functionality. These screen shots are used in the user interface map (see Figure 2a).

Screen Prototyping

Screen prototyping in Microsoft PowerPoint involves animating a series of screens to allow simulating the user's path to accomplish a task (see Figure 3). It is often used creatively as a design exploration tool, as well as in design meetings to facilitate a discussion. Screen-based prototyping is adopted within our design process as a quick way for simulating interaction. In our experience, branching of buttons allowed enough flexibility to mock up interaction paths, making it a method particularly suited for catching users' real-time reaction to navigation, focus, and logic. Given that the mockups reveal issues in end-user screen interaction without requiring development of hardware interaction, it is a major benefit to leverage this approach and provides considerable time and cost savings.

Buck Prototyping

Issues that are complex or that the team felt required usability testing as indicated in the specification are prototyped in Flash and tested using the Buck system (see Figure 4). This approach provides a more complete testing environment that can be evaluated

Figure 3. Screen prototyping. Screens are animated in PowerPoint using invisible buttons that allow for simple branching of navigation. The designer can use the buttons to click through different paths, allowing the test driving of specific ideas as well as sharing within the design team.

formally with external users [6]. This is the first point in our interaction design process at which external users are exposed to a design. A user's press of a hardware button communicates actions and sends a key press signal to the software. The software interprets the signal appropriately (up, down, option, shift, etc.). This provides the visual feedback associated with user behavior. Prototyping at this stage is effective in catching problems with timing, ergonomics, and task-based flow.

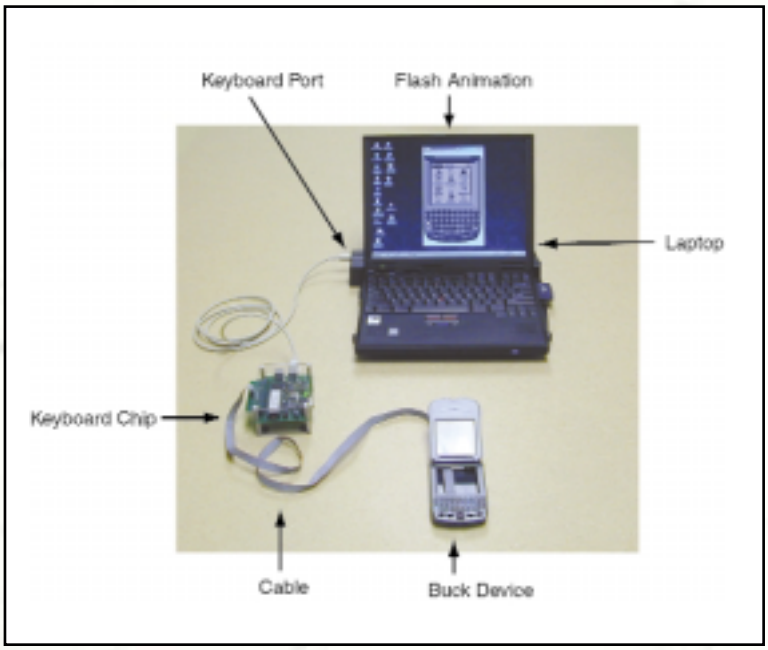

Figure 4. Buck prototyping. The Buck device is tethered to a laptop or workstation through the keyboard port. The software user interface information is read entirely from the laptop screen, whereas the hardware icons and labels are read from the Buck's hardware interface.

Alpha Validation

Alpha validation is often one of the first points at which software can be used on functional hardware (see Figure 5). Although Alpha code is not really prototyping, it is used to experiment with design issues as usage issues are identified. Internal beta testing occurs in this phase, in which participants are asked to use their own data in the device for extended periods of time. This is distinct from usability testing because it is based on real use rather than a series of tasks conducted by a facilitator. Without the Buck, most of the hardware-software interaction validation would only now take place. However, even after prototyping with the Buck, certain issues can be caught only here. In particular, issues of repeated use or problems that are most visible with a user's own data will mainly be caught in this phase of development.

Maximizing Prototyping Payoff

One of the biggest challenges of prototyping is to be efficient with the full spectrum of tools. Each step in prototyping has an increasing level of functionality that can be tested, and the more the level of functionality, the more time it takes to implement (see Figure 6). Thus it is disadvantageous to wait until real software code is developed to try out certain parts of a design, which is an expensive mistake companies often make. It is important to take advantage of each tool at the appropriate point in the design process.

The art of prototyping includes first knowing the prototype goals, the schedule for making a decision, and what questions can be answered within a given prototyping medium (e.g., Photoshop, PowerPoint, Flash, Buck). The medium that can answer questions in the least amount of time should be chosen. It is also important to manage the process of working with different tools simultaneously and be able to upgrade to the next level of media if needed. Another common problem is becoming too comfortable using only one type of prototyping tool. This familiarity can constrain prototyping to addressing only the issues that can be addressed with that tool. It

Figure 5. Alpha validation. Early versions of software code are implemented on a Treo device for internal beta testing with real data and use over time, often a period of weeks.

is important to have access to a full spectrum of tools in a wide range so that each is used at the appropriate time to ensure the maximum efficiency in prototyping.

Prototype effectiveness for hardware-software prototyping is different from more conventional prototyping environments (e.g., Web and PC applications). The time-functionality relationship has a more gradual ascent with conventional environments because there are less dimensions of interactivity (such as input controls and physical embodiments). Accordingly, with more conventional environments, paper-based and screen-based prototyping can be leveraged for a larger portion of the development cycle.

Prototyping Constraints and Tools

Prototyping is constrained by the development history. For example, when we started interaction prototyping for the Treo communicators, we had access to physical models but existing software didn't fully support the new hardware. Additionally, the basic industrial design was complete, and the software user interface design was still in the specification stage. Therefore, we could modify the software specification but not easily change the physical device. Ideally, we wanted a system to help us create a prototype of the hardware-software interaction that several designers could learn without much programming experience and that could be easily distributed to a large audience for testing.

To prototype the system, we developed the Buck, which combined the physical device prototype with software running on a laptop computer. After exploring many development and animation tools available on the market [5], we chose Macromedia Flash as the software base, mainly because of its graphical flexibility and ability to interface with keyboard events. To simulate interaction, we added electronics to the physical model and connected these circuits to the laptop's keyboard port using a simple microcontroller interface. The combination of this functional model and a responsive software interface provided a system that became the basis for prototyping. Within the dimensions of "fidelity" that often are used to classify prototyping techniques [7], this system is in the midrange of interactivity and functionality.

In our typical Buck prototyping process, designs begin on paper and are then animated in Flash using a user interaction map to understand the navigation paths (see Figure 2a and 2b). To use this system, a designer first describes a problem in text or with screen shots. Those screen shots are imported into or recreated in Flash. Key presses controlled by the Buck device drive the animation's events.

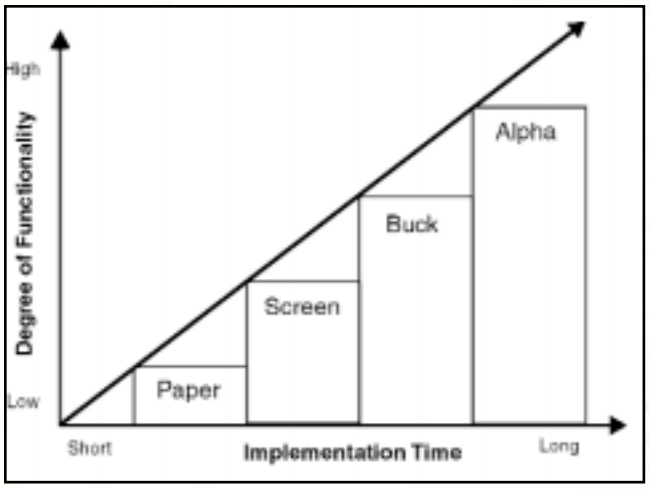

Figure 6. Prototyping effectiveness. Each prototyping step allows for a greater level of functionality and takes more time to implement.

What interaction elements do we typically prototype? On the basis of previous product designs and experience, important issues are flagged in the body of the specification for prototyping, usability testing, or both. Generally, if the issue is a problem with past

product development and is identified internally by quality assurance or externally by customers it is flagged for the next product cycle. Furthermore, if the design is experimental or complex it is likely to be flagged for validation. By focusing on these highlighted areas, we address the most "risky" features without having to completely deploy all the capabilities of the device.

Prototyping is also useful in the role of tie-breaking at design meetings. In the past, some design discussions became very long, as people had strong differences of opinion about what users would find confusing. Some discussions got to the point where it was much faster to prototype and test an idea than to discuss it. Prototyping as a solution became part of the decision-making process.

Prototyping is useful in the role of tie-breaking at design meetings.

Issues Captured with the Buck

With the Buck, we were able to prototype dozens of task-based scenarios. The Buck provides many advantages over hardware-only, software-only, or paper prototyping processes. Since it combines hardware and software into one system, the Buck is able to catch many subtle interaction problems, such as the following:

- Catching two-handed use problems. The Buck allows users to drive the software interaction using real buttons. Accordingly, we were able to find double-handed use problems. For example, to scroll sometimes the user had to hold the option key while pressing the scroll buttons. The impact of the user's experience with this design choice would not have been caught with paper or on-screen prototyping, because users are otherwise constrained with the use of a single mouse in a desktop environment.
- Graphical connections between the screen and the keys. By combining the hardware buttons with software functionality, users instinctively looked at the keyboard as well as the screen for visual clues about navigation and control. This type of connection was made clear when users had to learn a new function. The physicality of the Buck interface allows associations from other physical experiences to be explored, like similarities to a TV remote control or a landline phone.
- Timing and feedback associated with the keyboard controls. Experimenting with key-press interactions such as a long press versus a short press versus a double press could not be done with paper prototyping. This type of interaction is laden with expectations set by other products and is often hard to pinpoint without physically going through the motions and understanding the unexpected outcome of a particular sequence.
- Ergonomic issues with button placement. When participants were using the Buck to navigate menus and screens, ergonomic issues became apparent from the strain caused by repeated use. In addition, having users with different-sized

Figure 7. Buck user testing. A participant holds the functional model in his hands to control the animation on screen. The device is tethered to a laptop for usability testing in various locations.

hands revealed constraints of button placement as buttons were accidentally pressed. These issues would have been revealed only by repeated use of a device that provided visual feedback from button access.

User Testing

The Buck allows for effective user testing. Depending heavily on task-based scenarios, we use a "think-aloud" protocol that includes sitting next to the participant and guiding him through the various aspects of the system. The scenarios use a mix of canned data and real-time user input, creating a realistic experience graphically and interactively. The main caveat is that we were not using the participant's own data, which creates a somewhat artificial experience. For example, we might ask people to look up phone numbers and demonstrate dialing, but the participant did know any of the people in the phone book.

On the whole, users felt comfortable with our Buck testing setup (see Figure 7). Generally, the Buck makes the testing experience more concrete than that with paper-only or screen-only testing. Users are able to explore and simulate using the device for their own needs. Often we found users discovering functions on their own. We found this positive, and it meant that we merely had to guide them to the areas that we were interested in evaluating. The device is robust enough to allow for exploration and never broke. As a result, the testing assumed a much more observational quality than paper-only evaluation because it did not depend so heavily on a facilitated experience.

One challenge of user testing with Treo communicators was testing the two-way interaction of callers and receivers. The Buck system was not set up for multiple devices to interact, so we could not have two physical devices allowing sender and receiver participants to interact in real time. Additionally, the time to set up the devices would be too costly for our schedule. We addressed this in two ways: First, we use a *Wizard of Oz* approach [2], in which we verbally act out the part of the receiver. For example, the caller makes the phone call and we act out the part of the receiver answering the phone. Second, we digitally simulate the interaction of one of the parties. For example, when we were testing a messaging application, we simulated a two-way conversation by timing a two-second reaction to the user's input. After a user typed "did you get the spec?" plus a eturn character, that would trigger the simulation of another person's response, "yeah thanks." In this example, we took whatever input the user typed and made it fit the scripted dialog. So even if the participant typed "bla bla bla," our system would turn it rinto "did you get the spec?" We then instructed the user to go with the flow of the conversation. This worked as a technique for simulating communication sufficiently enough for us to evaluate the interaction.

Limitations and Opportunities

Although Buck prototyping allows a level of interaction validation that was not available previously, its major limitation is that the iterative part of prototyping occurs only in the software user interface. We did not modify the hardware user interface except the mappings of particular buttons. Why was this? How many times have software designers wanted to "just add one button" to do a certain task?

Constraints of modifying the hardware user interface are due to both organizational and financial factors, limitations that offer opportunities for both the extension of the process and the refinement of tools. As is typical of product development, different parts of the design operate on somewhat different schedules. Because the hardware manufacturing process is long, the hardware design is often defined earlier than the software design and is relatively fixed by the time the software user interface is prototyped. The cost of each hardware model is also much more expensive than each software iteration. As a consequence of the imbalance between hardware user interface and software user interface, the interaction iteration occurred only within software. This doesn't mean that hardware wasn't continuing to prototype button snap or fit, but that the interaction design was not being explored further. From an overall cost perspective, the Buck system required a team of five engineers to construct (mechanical, hardware, electrical, software, and industrial), because it involved developing a custom board. On the next iteration, it would be much cheaper to make, but it still would require many resources.

Prototyping occurs with more investment in resources and addresses issues of both design risk and design validation.

These issues suggest an opportunity for the human-computer interaction community. How can interaction design truly iterate over the hardware user interface and software user interface design simultaneously? What are the key ingredients for a hardware-software interaction prototyping setup?

Ideal Attributes

In our experience, we have identified several ideal attributes for an interaction prototyping setup. These features are different from those found in other prototyping toolkits proposed [4], in that they address the hardware interface design as well. Accordingly, a system with these aspects would allow designers to begin to iterate over both the hardware and the software user interface elements.

Our ideal attributes are the following:

- Graphic design. The ability to modlate the graphical design would help designers to experment with user interface elements. Designers could try novel button shapes, icons, and dialog windows, for example.
- New functional controls. The capability to rearrange the hardware user interface by adding new functional controls and buttons would facilitate more interaction variation before the hardware is relatively fixed. The ability to swap in and out controls such as buttons, jog rockers, scroll wheels, and touch pads while retaining functionality would help.
- Remapping existing controls. Being able to remap the hardware controls to

have different functions would provide flexibility. Having the ability to change an action from the "option key + scroll down" to the "jog rocker" instead would allow more fine-tuning iteration in the hardware user interface.

- Timing and feedback. Controlling button-press timing and feedback is critical for a smooth experience. Being able to change a button press from a "long press" to a "short tap" or to "two short taps" provides the ability to adjust important interactions as they indicate different actions to users.
- Ease of programming. The ability to develop the prototype relatively easily would allow multiple people to participate. The fewer technical skills required by a tool, the more designers will be able to use the tool, providing greater access within a design group.
- Low cost. Organizations will use the tools more widely if the cost is relatively inexpensive. Cost is based on the price of the product and the time needed to learn technical tools, which can make investment prohibitive for many designers.

The Buck system that we developed addressed all the above issues except the ability to rearrange the hardware, fully remap existing controls (our system only allowed partial remapping), and allow ease of programming. These hardware-related issues are the most challenging to incorporate.

Conclusion

Hardware-software interaction prototyping is a challenge. Its complexity lies in what it is trying to demonstrate (various controls, navigation paths, and timing issues), and the organizational structures upon which the prototyping process stands are diverse.

The difference between the hardware and software development cycles provides timing implications for hardware-software interaction prototyping. Because hardware tooling is relatively costly, frequent hardware prototyping occurs early because changes later are extremely expensive and can cause the schedule to slip. Software is often more flexible and can be changed later in a design process. Time and cost are important to both groups, but their flexibility is different. Understanding the design schedule and where each group is flexible is critical to planning when to prototype.

Using a prototyping approach to design, a team may build several variations on a concept and choose one.

Prototyping culture is also a major factor in organizations' approaches to design [10]. Some organizations have a prototyping-approach, and others have a more specification-based approach; each has its advantages. Using a prototyping approach to design, a team may build several variations on a concept and choose one. Prototyping occurs with more investment in resources and addresses issues of both design risk and design validation. Designs are more bottom-up and provide potentially more creativity because more people are involved early in the design process. Conversely, a specification approach to design begins by fleshing out the text in a specification using paper screenshots as examples and prototyping the major issues that are flagged in this process. Design-

ers using the specification approach often privately use prototyping as a tool to help resolve design problems and allow them to come up with the best solution. Specification-based prototypes are also more frequently used to answer risky questions rather than to validate concepts. This top-down process can be much faster with experienced designers, and the issues can be more easily communicated to the development team for implementation, reducing time-to-market. The main difference between these two cultures is the frequency, scope, and stage within the design at which the prototyping is appropriate and effective.

Despite the developmental and organizational challenges our teams faced in hardware-software prototyping, we came up with a creative solution. The organization that wanted more prototyping provided the prototyping services for the group that wanted more development speed. This cross-organizational solution worked well because both groups were able to collaborate synergistically. By working together, the organization was able to improve the interaction of products in the communicator line by improving the navigation, timing, visual feedback, ergonomic issues, and single-handed use of the product.

As a result of the increasing role of wireless and embedded computing in the consumer marketplace, the need for better hardware-software prototyping, like the system we have developed, is increasing. Consumer-focused interactive systems including products such as MP3 players, digital jewelry, digital cameras, and smart toys all share the necessity of a seamless user experience between the hardware and software interface.

Acknowledgments

The prototyping and design work at Handspring discussed in this article was based on the work of Rob Haitani and the design team. The people involved with building the Buck prototype include Peter Skillman, Dan Kim, and Miles Brown and engineering support. Also, Claudia Knight collaborated on the user testing. I would like to thank Heiko Sacher for the inspiration for this article, as well as Elizabeth Churchill, Trevor Pering, Ken Anderson, and Brian Sager for useful feedback.

References

1. Alavi, M. An assessment of the prototyping approach to information system development. *Communications of the ACM* (1984), p. 556.

2. Bernsen, N., et al. *Wizard of Oz* Prototyping: How and when? CCI Working Papers in Cognitive Science and HCI, WPCS-94-1. Center for Cognitive Science, Roskilde University, 1994.

3. Erickson, T. Lingua francas for design: Sacred places and pattern languages. In *Proceedings of Designing Interactive Systems Conference* (Aug. 17-19, New York), Association for Computing Machinery, New York, 2000.

4. Hakim, J. and Spitzer, T. Effective Prototyping for Usability. In *Proceedings of ACM Conference on Systems Documentation* (Cambridge, Mass.). IEEE Educational Activities Department, Piscataway, NJ, 2000, p. 52.

5. Hix, D. and Schulman, R.S. Human-computer interface development tools: A methodology for their evaluation. *Communications of the ACM* 34, 3 (March 1991), pp. 74-87.

6. Lin J., Newman M., Hong, J., and Landay, J.A. DENIM: Finding a tighter fit between tools and practice for Web site design. In Proceedings of the Conference on Human Factors and Computing Systems (The Hague, Netherlands). ACM Press, New York, 2000.

7. Mayhew, D. *The Usability Engineering Lifecycle*. Morgan Kaufman Publishers, 1999, p.22.

8. Nielsen, J. Heuristic evaluation. In Nielsen, J. and Mack, R.L. (eds.), *Usability Inspection Methods*, John Wiley & Sons, New York, 1994, pp. 25-64.

9. Norman, D. Design as practiced. In Winograd, T., *Bringing Design to Software*, ACM Press, New York, 1996.

10. Polson., P., Lewis, C., Reiman, J., and Wharton, C. Cognitive walkthroughs: A method for theory-based evaluation of user interfaces. *International Journal of Machine Studies 36*, 5 (1992), pp. 741-773.

11. Schrage, M. Prototyping cultures. In Winograd, T., *Bringing Design to Software*, ACM Press, New York, 1996.