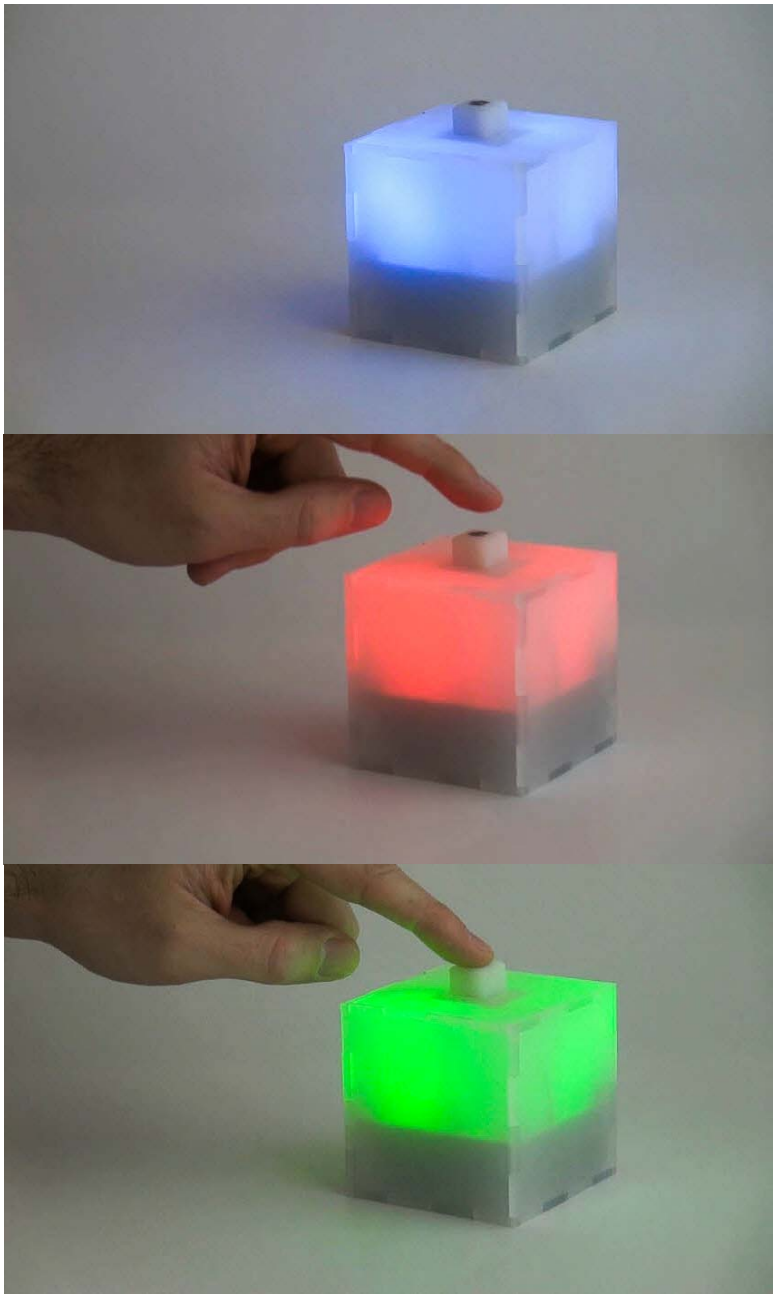
Stacked User Inputs

Combining digital and physical layers of interaction for "same-space- multiplexed" input



Master of Interaction Design Thesis – Malmö University, 2010

Author: **Rob Nero** me@robnero.com
Supervisor: Jörn Messeter jorn.messeter@mah.se

ABSTRACT

Multi-touch tables, iPhones, and iPads are just a few of the many devices to have embraced the mystical power of touch sensitivity. Somehow, without any physical push of a button, these devices can magically “feel” when my finger is touching them! Touch-sensitive technology is perceived to be such a recent addition in devices, that it still holds people in amazement and makes them believe they are living in a science-fiction fantasy. Is this the future for all devices though? The iPhone has proven to be such a success that it seems as though all mobile phone manufacturers are abandoning physical buttons in favor of touch-sensitive panels.

This thesis aims to point out that the physicality of interfaces should not be abandoned, but *combined* with touch-sensitivity. The haptic[1] feedback that I receive while pushing the keys down on my keyboard is an advantage that is quickly lost with the touch-sensitive screen of an iPhone. However, the touch-sensitive screen of an iPhone offers the ability of using natural gestures to provide input to the device, which a physical keyboard is unable to do at all. I propose that a physical interface can be combined with a touch-sensitive interface to create “Stacked User Inputs” that would combine the advantages of both interfaces, into one seamless interaction.

[1] Haptic: of or relating to the sense of touch; tactile. http://en.wiktionary.org/wiki/haptic

TABLE OF CONTENTS

INTRODUCTION

A paradox has taken shape in human-computer interfaces: computing devices are getting smaller and packing more capabilities into smaller form factors, while the invention of multi-touch surface interactions has made us attempt to incorporate these rich interactions into our portable computers. People want a computer to match their increased mobility and are unwilling to compromise on how they interact with it.

Many attempts have been made to solve this paradox. Netbooks have offered one solution for portable computing by shrinking the overall shape, but the decreased size can make interacting with it (for example, typing) more difficult. “The problem is, netbooks aren’t better at anything.” said Steve Jobs, when trying to explain what advantages netbooks hold over smartphones or laptops [6]. The post-smartphone era of iPhone clones with their digital touchscreens have been adding rich, multi-touch interactions to their pocket-sized computers, but their size still makes common computing operations (such as web surfing or writing emails) time-consuming and unnatural. Tangible computing also joins this paradox by attempting to add physical control over virtual elements, and introducing the natural motion of using our two hands as the primary method of providing input to a device. This only adds frustration to the mix by increasing our expectations of interacting with a device, only to be disappointed when it fails to respond to our natural hand gestures.

“Stacked user inputs” is a concept that attempts to offer relief, and a possible solution, to this paradox. Where current devices provide a physical *or* a digital input interface, stacked user inputs combine them to provide a cohesive physical *and* digital input interface. This combination of previously separate interfaces, could offer richer interaction while not requiring additional physical space on a device.

Figure 1: iPod Classic

The iPod Classic design (Figure 1) is a perfect example of stacked user inputs, and an easy way to understand the *physical* and *digital* interfaces. If you are not familiar with using this specific iPod model, please watch the video in Figure 2 before continuing. The entire interface to controlling the iPod is contained inside a “jog wheel” and center button. Every operation of the device is controlled within this single interface. Additional study of this interface, though, reveals the stacking of two independent interface layers.

Figure 2: Using the iPod Classic

http://www.vimeo.com/11838765

1:13 minute video

The top interface layer is what I would label the “digital layer of interaction” and responds to the touch of your finger. I consider this layer *digital* since it is transparent, offers no immediate affordance to its use or function, and is digitally interpreted. To interact with this layer, all that is necessary is to glide your finger in a circular motion in either direction, on top of the white “wheel”. The iPod responds by scrolling through vertical lists of data (e.g., artist or song list), or horizontal attributes (e.g., volume or star rating).

The bottom interface layer is what I refer to as the “physical layer of interaction”. The physical layer consists of a 2-state (down and up) button underneath the jog wheel for the top, right, bottom, and left positions. I consider this a physical layer since you are physical pushing down on the wheel to “click” each button, as you would normally expect to interact with a physical button. The four buttons under the wheel perform functions commonly used when listening to music: play, pause, next song, previous song. The center button, which is not a part of the stacked interface, performs the function of committing to an action in the graphical interface (e.g., choosing a song to play or advancing the menus).

This thesis will explore this concept of combining physical and digital layers of interaction to understand its potential uses and parameters to consider when it is used. I will first provide a background on a prototype I developed before starting work on the thesis, which sparked the stacking concept. This is the starting point, and the beginning of this entire story. Next, I will analyze and explain other devices, like the iPod Classic, that embody characteristics of stacking and will aid in understanding and defining “stacked user inputs”. I have created a small prototype abstracting the concept, and used it as a technology probe to gain feedback on the concept. I will conclude by sharing the results of testing the probe, and how the results impact defining the concept.

BACKGROUND

The concept of stacking user inputs was born out of a previous project of mine, at the end of the first year of the Interaction Design Masters program at Malmö University, in 2009. The framing for this last project was to use “research through design” as the method for creating a knowledge contribution. We were encouraged to create a design that could be tested, to gain insights and feedback on the design, which could then be formulated into research findings. As it is mentioned in a research paper outlining the method, “…it stressed design artifacts as outcomes that can transform the world from its current state to a preferred state [1].”

I was provoked early in the project, while sitting at a small café table with my laptop. I enjoy the portability of my laptop, but am continuously frustrated with my options for providing input to the laptop. There is a trackpad built into the laptop, but I find its size and overall usability to be frustrating. I carry a small wireless external mouse with my laptop, but the small table at the café was only big enough to hold my laptop with no extra room to navigate a mouse. It was all of these frustrations that provoked me to think of a better input device.

A quick survey of portable computers and input devices uncovered design trends and common design decisions. Small laptops and “netbooks” [footnote: definition] are becoming more common, even though their reduced processing power limits the types of applications that are usable. In this case, form has followed function, and the overall size of the netbooks has been reduced too.

I discovered three design decisions that were consistent amongst the netbook designs:

1. The common clamshell design will have both sides of the netbook be equal in size. (Figure 3)

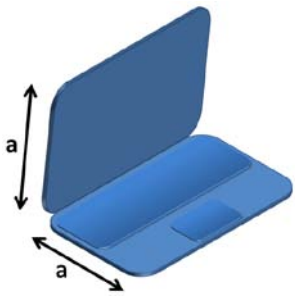


Figure 3: Clamshell design.

2. The keyboard is shrinking to reduce the overall size, but shrinking to a limit since our fingers are not shrinking as well. The keys on the netbook keyboards are smaller than a full-size keyboard, but not getting so small so that they become unusable. (Figure 4)

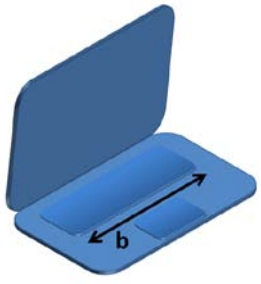


Figure 4: Keyboard shrinking.

Figure 5: Trackpad shrinking.

3. The keyboard is occupying most of the space on its side of the clamshell design, stealing space away from the trackpad input device. (Figure 5)

Input devices for portable computing offered little relief past just making the device smaller to match the portable computer. An abundance of external mouse options are available but appear to be first designed for a desktop computer. To be used for portable computing the mouse was just made smaller, or made wireless, or both. Evolution has made the trackpad the default input device for portable computers, making the trackball and “eraser head” trackpoint input devices ancient history.

“Why can’t these input devices be combined?” is what I thought while sitting at the café that day. The keyboard is a large flat surface that is only used when I am typing. My input to a computer is typically modal: I am providing keyboard input, or I am providing mouse input, but usually not both at the same time. The flat surface on top of the keyboard is a large unused space when I am not typing.

Figure 6: Final TRKBRD Prototype

“What if the trackpad was placed on top of the keyboard?” My idea was to combine the trackpad and the keyboard, to create a trackboard. The keyboard would function as a typical button keyboard that has an “up” state and “down” state for each physical key. A sensor grid would be placed on top of the keyboard to provide the cursor (mouse) input. To move the cursor on the screen, I could simply glide my finger over the keys, being careful not to press the keys down. If I tapped on the key to “click” the mouse, or if I quickly typed a key on the keyboard, the trackboard would be smart enough to know each interaction and provide the correct reaction. Combining the trackpad and keyboard was my theoretical idea, and building the TRKBRD (trackboard) was the actual physical prototype that I built to test the idea. The prototype can be seen in Figure 6, and a video showing the prototype working can be seen in Figure 7.

Figure 7: TRKBRD Working Flash Prototype!

http://www.vimeo.com/9105600

1:16 minute video

It was at this point, when working out the theoretical and mechanical details of the TRKBRD, that I discovered the concept of “stacking inputs”. In the case of the TRKBRD, I was physically stacking one input layer on top of another input layer to provide a better input experience for spatially-limited portable computers.

My initial design decisions for “stacked user inputs” followed the form and function of the TRKBRD design, and became the starting point for this thesis:

1) Multiple input interfaces

2) Input interfaces occupy the same physical space (or within touch proximity of each other)

3) Independent control of each interface

RELATED EXAMPLES

The TRKBRD prototype provoked me to see the “stacking” concept, but it is not the only device that embodies some of the principles of the concept. Additional exploration into devices and interfaces uncovered other designs that show the potential to be labeled “stacked”.

TRKBRD

As stated earlier, the TRKBRD design is what provoked me to pursue this thesis. This design example perfectly illustrates the concept of stacking, since it is the design that created the concept.

Figure 8: TRKBRD installed on top of laptop keyboard, turned ON, and functioning normally.

The laptop keyboard provides the physical layer of interaction, shown in purple in Figure 9. It is a typical computer keyboard that requires a person to push down a key in order for it to register as a keystroke.

Figures 9 through 11 show a side-angle view of the TRKBRD installed on the laptop.

Figure 9: Physical layer.

The small cylinder on the left side of Figure 10 is one of the lasers providing infrared light. The multiple, small, black components are the infrared light sensors. This combination of components produces an invisible field on top of the keyboard to create the digital layer of interaction. When a finger is moving in this invisible field, it is translated to cursor movements on the screen. It is also possible to “tap” and “double-tap” in the field, the same way as if you were using a trackpad or touch-sensitive screen.

Figure 10: Digital layer.

Figure 11: Physical and digital layers.

Together, a digital layer is stacked on top of a physical layer, as seen in Figure 11. A person is able to type as you would normally type on a keyboard. The device is intelligent enough to know when a key is pressed, and then cancel a “tap” command. Each layer of interaction can be independently controlled, and each layer controls a separate device. What was previously two different devices, a mouse and a keyboard, are combined as a single device for the TRKBRD.

Aftertouch

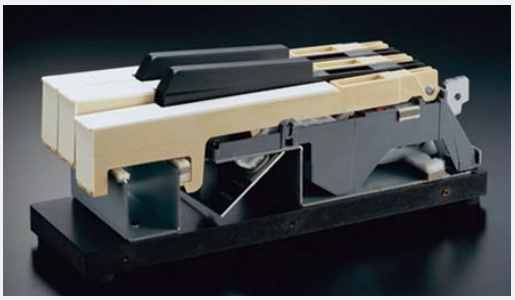


Figure 12: Synthesizer key side-cut view.

"Aftertouch" is pressure sensitivity on an electronic synthesizer, allowing the musician to change the tone or sound of a note after it is struck [10]. For example, it could allow a musician to slightly bend the tone of a note or slowly add vibrato to a tone, by simply pressing a little harder while holding down the key to sustain the note.

I cannot confirm that the photo used for this example is from an aftertouch-enabled keyboard. I am using this photo for demonstration purposes only.

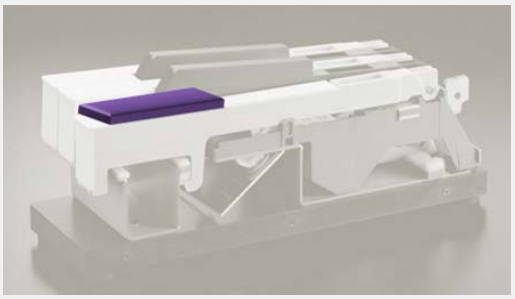


Figure 13: Physical layer.

The top layer of a keyboard is the obvious and most visible layer: the tone-producing key "button" (Figure 13). When a key is pressed, a tone is produced. Depending how advanced the keyboard is, would determine how much control the musician can have on producing the tone. For example, on more technologically advanced keyboards (and usually much more expensive), a musician could produce a tone of varied loudnesses, or varied percussive attacks of the tone. Versus, a cheaper keyboard that produces the same loudness and attack no matter how differently you press the key.

All of these elements pertain to the tone-producing layer of interaction on a keyboard.

The "aftertouch" layer of interaction occurs only after a tone is produced in the upper layer. Presumably there are sensors underneath the key that determine when it is pressed to produce a tone, and also sense when slightly more pressure is given to the press to activate aftertouch (Figure 14). Aftertouch can only be activated once a tone is generated. This fact breaks one of the rules of determining a SUI: interface independence. The upper layer of interaction must be used in order to get to the lower layer of interaction (Figure 15).

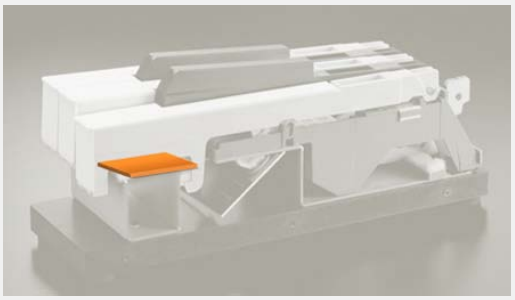


Figure 14: Digital layer.

Aftertouch is not a SUI for multiple reasons:

- There are not multiple interfaces. The upper layer produces a tone, while the lower layer augments the tone. Both are related and interfacing with the tone generating interface of the keyboard.
- Both interfaces described here occupy the same physical space, but it is unclear whether the interfaces described here are separate and plural.
- There is no interface independence. Aftertouch cannot be activated without first interacting with the upper physical layer of interaction.

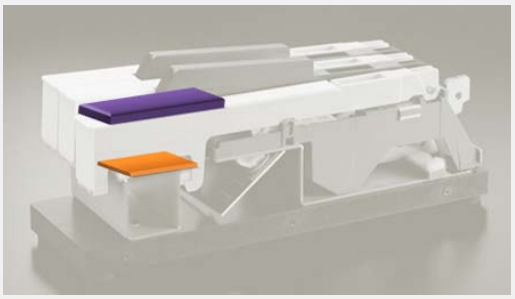


Figure 15: Physical and digital layers.

Apple Trackpad

The most recent design of the Apple Trackpad seems to be an obvious real-world example of stacking interfaces into a single device. It is clear why people think of this device right away when asked about stacked inputs, because it is Apple's daring design decision that has provoked people to think "Do I really need a separate button?" It is a large button that is touch-sensitive on the top.

Figure 16: Apple buttonless Trackpad.

The lower layer of interaction in the trackpad is the button level (Figure 17). When you press down, it is the same result as if you clicked a mouse or tapped the trackpad: a mouse-click event is triggered. Since a press of the trackpad and a tap on the trackpad are both registered as the same event, it could be argued that this button level could be removed from the trackpad with no degradation of interaction. It is my guess that the button level was kept in the design to assist in those few occasions when only using a tap can make an interaction more difficult than using a button.

Figure 17: Physical layer.

Some interactions are more efficient and manually possible through the addition of physicality, instead of attempting to combine a string of touch motions that virtually create the same result.

The lower button level of the trackpad provides access to a single mouse event, click.

Figure 18: Digital layer.

The upper level of the Trackpad provides a plethora of interaction that goes beyond the assumed abilities of a typical trackpad (Figure 18). For full disclosure, I must admit that I do not have an Apple Trackpad and am only stating functionality from what I have witnessed or seen online [11].

All typical interactions are possible: tap, double-tap, drag. Apple expands on these interactions, though, through the use of a multi-touch trackpad: "right-click" when tap with second finger when first finger is already touching, two-finger scroll for a webpage, four-finger up to hide all open applications, four-finger down to show a small icon of each open application, pinch two fingers to zoom, rotate two fingers to rotate an image, hold your thumb down while moving your finger to perform a click-drag, press with two fingers for a "right-click", three fingers swiped to the left or right to move back or forward while surfing the web, four fingers swiped left or right to open the Application Switcher... and on top of all that... you can customize all of these gestures through a preferences pane in the control panel.

Figure 19: Physical and digital layers.

It is not certain if the Apple Trackpad is a SUI. It does not contain multiple interfaces in accordance to how the TRKBRD contains multiple interfaces. If the definition of "multiple interfaces" were adjusted, though, the answer could be "Yes" to the multiple interfaces question.

- The top layer provides cursor movement control, cursor event commands, and access to a limited set of gesture commands.
- The bottom layer provides cursor event commands, too.

Where the definition of "multiple interfaces" could be stretched, and I think where most people are convinced that it is a SUI, is how gestures are incorporated into the overall functionality of the trackpad. Without the gestures, the Apple Trackpad would without a doubt not be a SUI because of the absence of opposed interfaces. The

button and touch-sensitivity are both controlling the same interface. With the inclusion of gestures in the top layer of interaction, the line that separates the interfaces becomes blurred, and the definition of "interface" becomes uncertain.

From my inexperienced understanding of the Apple Trackpad gestures, they are simply shortcuts or "quick-keys" to functionality that is attainable by other means. The gesture is providing the user with a quicker way of performing the function by allowing the user to maintain their hand position and not move their arm to press a button, or by eliminating multiple cursor movements and click commands. Because of this, I do not consider the gestures to be an interface.

Apple iPod Classic

The Introduction introduces the iPod Classic (Figure 20) as a good example of stacked user inputs. Below is a more in-depth study of its interfaces.

Figure 20: Apple iPod Classic.

Using your finger, you can touch the jog wheel and move your finger in a circular motion (Figure 21). This interaction modifies a number of controls on the screen.

- It scrolls whichever list is visible on the screen. A clockwise motion scrolls down, while a counter-clockwise motion scrolls up.
- If a horizontal status bar is visible, it can adjust the current position in the status bar to the left or right (clockwise for right, counter-clockwise for left). This is used for adjusting the volume and adjusting the time position of the song.
- If the star rating is visible, scrolling to the left or right with the jog wheel selects more or less stars.
- To use the Genius playlist control, scrolling the jog wheel to the right moves a selection arrow control, which initializes the Genius feature.

Figure 21: Digital layer.

The jog wheel is typically used as the selection control for the iPod. The device does not have any type of pointing device to make selections, as

someone is typically used to using on a desktop computer for example. Using the jog wheel as a

selection interface is more apparent when trying to play Solitaire on the small device.

Moving your finger to the left or right on the jog wheel moves a "selection hand" on the screen. The hand moves in a linear pattern allowing you to select a card to move in the game. This selection interaction on a desktop computer is strictly given to the mouse or trackpad to provide input.

Underneath the jog wheel are 4 buttons that provide more input to the iPod interaction (Figure 22).

Figure 22: Physical layer.

- The top button (MENU) is typically used as a "back button".
- The right button is used to advance to the next song, or when held down, used to advance the current song by a few seconds.
- The left button is used to go to the beginning of the current song, go back one song, or go back in time in the current song.
- The bottom button is used play or pause the song.

The middle button that is not underneath the jog wheel, and thus not included in the SUI interface, is used to advance through the menus or commit to an action.

The iPod Classic represents a SUI interface:

Figure 23: Physical and digital layers.

- Multiple interfaces
  - Jog Wheel controls menu selection (scrolling through lists), and parameter adjustments (changing volume, place in song, star rating).
  - 3 of 4 buttons control music playing.
  - MENU button function is related to menu selection and not music playing.
- Interfaces occupy the same physical space.

- Each interface can be independently controlled without interfering with the other interface.

The iPod's adherence to the SUI framework is weak for rule #1, though: there is not a strict separation of interface functions.

- One button in the lower layer is used in conjunction with the jog wheel in the upper layer.

- The jog wheel can be used to adjust a music playing parameter.

RELATED RESEARCH

"Tangible interaction" was the starting point to begin researching "stacked user inputs" even though the related examples just mentioned do not directly fall into this domain. This was decided because of the potential larger application of a stacked input device.

Figure 24: Siftables Equation Editor

http://www.vimeo.com/3164229

0:14 minute video

Figure 25: Siftables word game

http://www.vimeo.com/3164983

0:18 minute video

Tangible interaction "…allows users to 'grasp & manipulate' bits in the center of users' attention by coupling the bits with everyday physical objects and architectural surfaces [12]." In other words, it aims to provide the ability to control data or virtual objects on a computer screen or some other type of digital device, with the use of real-world objects that a person could physically hold or manipulate. A good example of tangible interaction would be the MIT research project called "Siftables", which has now been commercialized under the name Sifteo [13]. The Sifteo blocks can be assigned various values, from numbers, to letters, or even colors. When the blocks are placed near each other, or manipulated in numerous other ways. The blocks react to each other as if they have awareness of all the other block values surrounding it.

The video links in Figures 24 and 25 show two of these possible interactions. Figure 24 shows how the last block can display the sum of the other two blocks, just by placing them in the correct order and including the necessary mathematic signs. Figure 25 demonstrates each block's awareness of the surrounding blocks. When lettered blocks are lined up to spell a word, the blocks outline in blue when a correctly spelled word is formed. Both of these examples demonstrate how the physical moving and orienting of the blocks are manipulating digital pieces of data.

While I initially had uncertainty in where to begin researching stacked inputs, I used the tangible interaction domain as the starting point and branched out from there. The tangible interaction framework proposed by Hornecker and Buur [14] organized these early explorations and brainstorming sessions. Their framework focuses on the "interweaving of the material/physical and… the social aspects of tangible interaction" [14] through the use of four themes: Tangible Manipulation, Spatial Interaction, Embodied Facilitation, and Expressive Representation. To better understand these themes, I visualized my interpretation of their research, as seen in Figure 26.

I interpreted their framework as outlining a series of perimeters around a person when they are interacting with tangible interfaces. The first theme of "Tangible Manipulation" was the closest perimeter around a person and defined the individual's direct interaction with objects with his or her hands. The next perimeter of "Spatial Interaction" moved further away from a person and defined how his or her body interacted in "space". The third perimeter of "Embodied Facilitation" grew larger and encompassed multiple bodies socially interacting in a "place". I do not view the last theme of "Expressive Representation" as a perimeter, but as something that can be achieved in any of the perimeters.

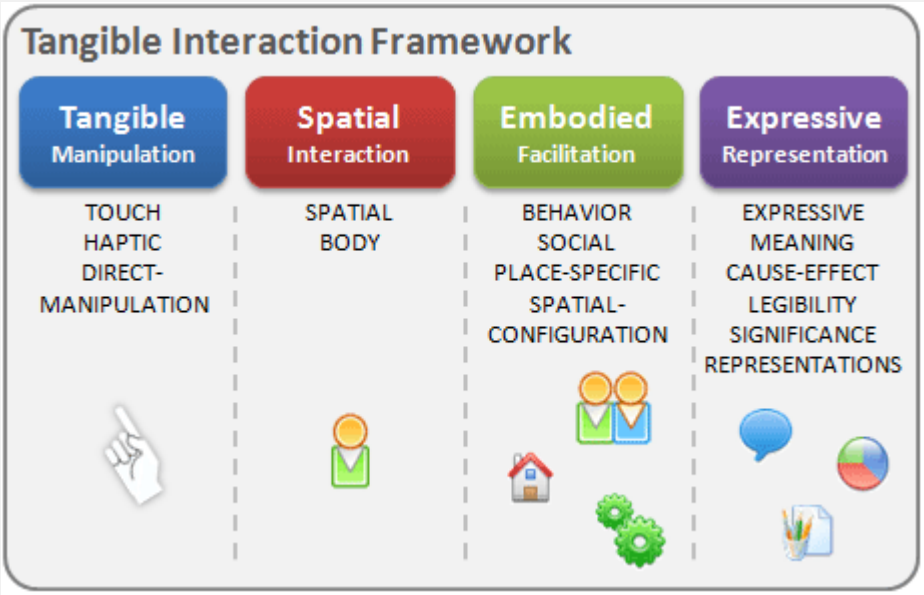


Figure 26: Visual interpretation of Tangible Interaction Framework [14].

The result of using this framework while brainstorming ideas for this thesis is shown in Figure 27. The details of the image are not important, as much as the fact that the framework proved to be invaluable in obtaining the focus for my research. Each theme provided a focus while brainstorming, an organization for ideas, and a way to relate ideas to each other. It was during this session that I came to an important realization, shown on the left side, that I explain in the Research Approach section later.

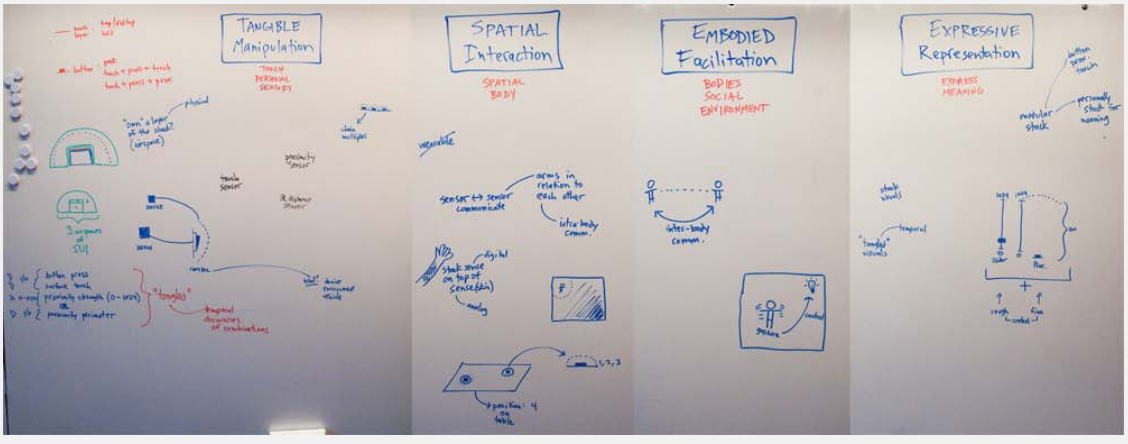


Figure 27: Brainstorming session while using the Tangible Interaction Framework.

Figure 28: Microsoft Arc mouse.

Figure 29: Apple keyboard.

To more adequately classify related examples and position my concept in relation to other research, I included the concepts of "time-multiplexed" and "space-multiplexed" input devices [9]. These classifications define how functions are controlled by an input device. "With space-multiplexed input, each function to be controlled has a dedicated transducer, each occupying its own space. In contrast, time-multiplexing input uses one device to control different functions at different points in time. [9]"

Examples of each classification include the simple mouse and keyboard used on almost every computer. A computer mouse is an example of a time-multiplexed input device (Figure 28). A mouse controls different functions on a computer's graphical user interface at different points in time. At any moment in time, a mouse is used for clicking a button, scrolling a document, highlighting a word, or clicking a link on a web page. An example of a space-multiplexed device would be a standard keyboard for a computer (Figure 29). Each key on the keyboard is available through its own key.

How is each related example classified?

TRKBRD = space-multiplexed
The keyboard input and TRKBRD input both occupy their own physical space. However, if different technology were used that would embed the TRKBRD into the keyboard, they would occupy the "same space".

Aftertouch = space-multiplexed
The key that generates the tone and the sensor that activates the Aftertouch effect occupy their own space. The Aftertouch effect occupies the small space that resides between pressing a key normally, and pressing it just a little bit harder.

Apple Trackpad = time-multiplexed
Although different parts of this interface occupy their own space, time is a factor in determining what function is provided when a person interacts with each interface.

Apple iPod Classic = space-multiplexed/time-multiplexed
Some of the controls on the iPod occupy their own space, such as the Play/Pause and MENU buttons. Other controls are time-multiplexed such as the scrollwheel and center button. Depending on when you use both of these, they control different parts of the graphical user interface. It is important to note that the iPod also places all of these controls in the "same space".

A potential expansion to the concept of "space-multiplexed" could be "same-space-multiplexing". This expansion would adjust the definition to "each function to be controlled has a dedicated transducer, with each transducer occupying the *same* space". Both examples of stacked user inputs, the TRKBRD and iPod Classic, can be classified under this adjusted definition.

Combining layers of interaction is not a foreign idea to interfaces, and can be illustrated with two research projects. For the "Layered Touch Panel" project, an invisible field similar to the TRKBRD was placed on top of a typical touchscreen [15]. This layering was used

to detect a finger hovering over the touchscreen and to provide a "mouseover" type event that is currently not available with touchscreens. The other project involved a complicated collection of technology that would allow someone to physically hold up something they wanted to virtually "pin" to a wall [16], for example, a piece of paper. The technology would quickly capture a photo of what was to be pinned, and then project it onto the wall. Their use of layering input was found in how they created virtual planes in front of the wall (only centimeters apart) so they could detect when a person was holding up something they wanted to "pin". When both planes were broken by the paper and person's hand, they would interpret this as a command to capture a photo of the paper.

Making buttons touch-sensitive is also not a foreign idea. An overlay of sensors was placed on a "clamshell" style mobile phone for the "SmartPad" project, allowing a person to glide their fingers over the keypad and also press the key buttons [17]. They proposed "previewable physical interaction", allowing a person to preview information before committing to an action on the information. In another project with the same researchers, they used a similar technology in a larger number keypad to test similar interactions for desktop computer applications. The "PreSense" keypad could be used for navigating a map or photo library, for example [18].

The TRKBRD design too, has been almost replicated in a research project only a few years before. The "Touch&Type" input device is a normal looking desktop computer keyboard with embedded technology that makes the keys touch-sensitive [19]. Key differences in the two projects include the technology used, what interactions were included, and testing methods. The technology in the TRKBRD is immediately apparent when using it, while the Touch&Type was able to make the technology completely invisible to the user. The TRKBRD included the ability to "tap" and "double-tap" in the touch plane to provide cursor commands, while the Touch&Type used physical buttons on the side of the keyboard that were accessed through a mode switch. The user would have to select the "typing" mode or "touch" mode explicitly, while the TRKBRD included intelligence to predict these modes. The Touch&Type research aimed for quantitative results through rigid usability lab tests. I aimed for qualitative results by testing the experience with people that I literally stopped on the street and showed them the prototype on a bench.

FOCUS

The idea of “stacking layers of interaction” to add awareness and new interactions is not a new concept, proven by the research and devices previously mentioned. Layers of “finger-detecting” planes have been added to walls and touchscreens to interpret a person’s intent while interacting with the device. Touch-sensitive buttons have been added to a mobile phone and number keypad to interpret a person’s interaction as ways to augment the user experience in software. The critical word to point out here is *interpret*. All of these projects have used multiple layers to interpret a person’s behavior, in order to augment or assist the system or device.

“Stacked User Inputs” aims to expand on the previous research of layering interaction, but concentrate on the ability to maintain independent control of each layer for the intent of controlling separate interfaces. It is not that one layer is merely offering a way to adjust another layer. It is to create two separate interfaces, controlling separate devices, and occupying the same physical space.

RESEARCH APPROACH

The research approach for this thesis continues with the “Research through Design” theme that was the starting point for the TRKBRD project. It was the intent for this thesis to create a prototype that embodied the “stacked user input” concept, and then use the prototype to further explore the concept and test with users. This was also the intent for the TRKBRD project, to build a prototype and test it with users. However, this thesis aimed to focus on the *concept of stacking* and not on any specific *contextualization of stacking*. This may seem to be an insignificant and nuanced difference between the projects, but the following explanation of the thesis research approach will explain key differences that separate the projects.

My first major decision was to create and test an *abstract* prototype instead of a fully-contextualized prototype. This was not an easy decision to realize. I had many frustrating brainstorming sessions prior to making this decision, attempting to discover a new prototype to build for this thesis. I needed an additional prototype to further exemplify the stacking concept, but could not expand my ideas past the obvious ones such as an advanced light switch or another computer input device.

It wasn’t until I realized my own personal position in the brainstorming process that I realized my biggest limitation: my body of knowledge is limited to what I know and what I currently know best is computer input devices. I know input devices best because of the research I collected during the TRKBRD project, from building and presenting the TRKBRD project to many groups, and from the fact that I use computers on a regular basis. It was during a decisive brainstorming session that I realized that I was trying to contextualize my concept within another domain I had no prior experience or knowledge. The abstract prototype would demonstrate the stacking concept, without providing input to any specific device or context.

Prototype

To demonstrate the concept, the prototype would need to distill the functionality down to only what is necessary. I decided on using a single button to represent the physical layer of interaction, and making the button touch-sensitive to represent the digital layer of interaction. A touch-sensitive button is what personified the most basic stacked user inputs during my brainstorm sessions. Even if the button was multiplied or aesthetically modified during my idea generating, the core concept would still simplify down to a simple touch-sensitive button.

An Arduino micro-controller would provide the ability to add programming logic to the prototype, and dictate how its functionality would respond to various interactions. The Arduino can capture when the button is pressed and when the top of the button is being touched by a finger. It then takes this data and follows a long progression of software logic statements to understand the interaction and provide the appropriate response.

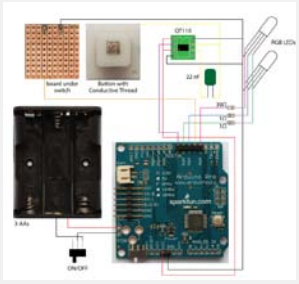


Figure 30: SUI Cube Electronics schematic.

Larger image available in Appendix A.

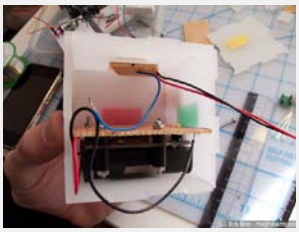


Figure 31: Inside of cube, assembling electronic components.

Two RGB (red/green/blue) light-emitting diodes (LEDs) provide the interaction responses by shining different colors and different brightnesses. Light is the only type of response to the user's interactions, to purposely keep the responses simple and easily understandable. The advantage of using RGB LEDs is that their color is determined through the software. This means, that once the prototype is built, I can easily change the software to perceive new interactions and provide different colored reactions without changing any of the electronic hardware. Figure 30 shows the complete electronics schematic: Arduino Pro, 3-AA batteries, RGB LEDs, QT110 touch sensor chip, and a single silicon rubber button with conductive thread stitched on the top surface. The conductive thread and touch sensor chip are used together to accurately detect the touch of a finger. This specific button was chosen for its aesthetic and functional properties. The color of the button is a simple white (and similar to the cube color described below), and proportionate to the final size of the probe. Its functional advantages include that it is easy to include in a switch circuit, and the button is hollow which allowed easy sticking of the conductive thread.

To house the button and all the necessary electronics, I decided on a small cube (henceforth referred to as "the cube"). I wasn't focusing on aesthetic qualities for the prototype or the tests, but still wanted to achieve a simple design that was unassuming. Hopefully the prototype would not instill any preconceptions to possible interactions or reactions based on the form or materials used for construction. I built a seven centimeter square cube from laser-cut, milky-white plastic. Milky-white plastic was used to diffuse the LEDs inside and create a glow around almost the entire cube.

The size of the cube was just big enough for the Arduino board, LEDs, various other components, and a 3-AA-size battery compartment. I decided to use batteries so that no external wires would be necessary to connect while testing the prototype. It was important to me that the prototype was as simple as possible to use and did not require any external connections in order to operate. Once the cube was switched on, all that was necessary for a tester to think about was how they wanted to interact with the button. The tester could concentrate on their interactions and the cube responses and not have to worry about whether or not they connected it properly.

Technology Probe

Once I built multiple cubes, I used the Technology Probe method to "release the cubes into the wild" to gather research data [20]. I used this method in order to allow an individual a longer period of time to test the prototype. Instead of a standard usability test where a person is tested on specific actions and functions during a short timeframe, my intent for the technology probe was to allow more time for the person to explore the prototype and track their discoveries over time. I wanted each person testing the cube to be relaxed in familiar environments and not feel as though they were

being tested to find the correct answer. “Time” was just as much of a factor in the probe, as the physical technology that the person interacted with.

Multiple attributes inspired the design of the probe, in accordance to the framework described by Hutchinson et al [2]. They describe how a prototype is different than a probe, and thus the corresponding testing method should also be adjusted. I consider my TRKBRD design to be a prototype that I used to conduct “usability-lab-style” tests for qualitative results. I desired a different kind of result with the “cube” tests, however, that would incorporate time as a factor and spark more creative problem solving from the testers. The cube needed to test the ability of interacting with stacked inputs, but also provoke the tester to think past their preconceptions of physical and digital interfaces.

Distinguishing Features

The following five features that distinguish a probe from a prototype grounded my design decisions for the cube, and assisted me in differentiating the cube from my previous TRKBRD prototype [2].

Functionality

“Technology probes should be as simple as possible…” [2]

While sketching early versions of the cube prototype, I had considered other forms and functions for the prototype. I considered a larger configuration of multiple interconnected buttons, a button with multiple touch-sensitive points along the sides, and a button with multiple touch-sensitive points on just the top surface. I simplified the sketch each time and came down to a single button with a single touch-sensitive zone on the top surface. This was the bare minimum in functionality that would be needed in order to test the core concept.

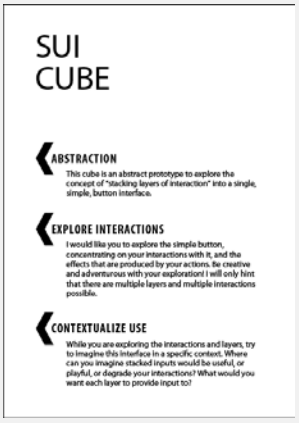


Figure 32: Instructions included with each probe.

Larger image available in Appendix B.

Flexibility

“…they should be designed to be open-ended with respect to use, and users should be encouraged to reinterpret them…” [2]

This was my exact intent for my cube prototype, and was explicitly stated in the directions included with each cube (Figure 32). The purpose of the cube was to abstract the core concept of stacking inputs, gain feedback on the interaction of using the cube, and spark ideas of contextualizing the concept with the tester. The cube was designed to be plain and absent of preconceptions to assist the tester in reinterpreting its use and application.

Usability

“They are not changed during the use period based on user

feedback. In fact, a deliberate lack of certain functionality might be chosen in an effort to provoke the users." [2]

Three different versions of code were written, with each version detecting different types of interactions and providing slightly different responses. A cube would be delivered to a tester with one of the code versions, and stay unchanged throughout testing. Code versions #1 and #2 were written before testing started and then randomly assigned to each tester. Code version #3 was written as a backup version, after the initial tester interviews uncovered some frustration with the interactions.

The sensors in the cube were limited to a button that can be pressed down, and a touch-sensitive area on the top surface of the button. No other sensors were included in the cube.

Logging

"Technology probes collect data about users and help them generate ideas for new technology. Prototypes can collect data as well, but this is not a primary goal." [2]

I was most interested in how each person used the cube probe, and how comfortable they were with the interactions. Time and resources prevented me from embedding logging functionality inside the cube, to retrieve and analyze after each test. This type of quantitative data was not the focus of the probe, though. I wanted to focus on the qualitative results of each tester spending time with the probe, and record video of their reactions to further analyze their nuanced use of the probe.

Design phase

"Technology probes should be introduced early in the design process as a tool for challenging preexisting ideas and influencing future design." [2]

For the context of this thesis, there was no specific future design to influence. Instead, the probe was targeting hypothetical and yet-to-be-seen future designs imagined by the testers or anyone that reads this thesis. This feature further illustrates how the Technology Probe method is better suited for this thesis. The cube probe is not a finished prototype that needs analysis of use to determine its positive and negative attributes; it is the starting point of generating ideas based on the concept. The intention of the probe is to provoke the tester.

Light-weight

The Technology Probe method was only the starting point for my chosen research method. Limitations and obstacles forced me to

modify the method and tailor it to fit my thesis. The intention of the Technology Probe method is to embed a technology in a real-world context for an extended period of time, to gather data on how people use the technology in their daily lives [2]. However, resources and time constraints in my thesis prevented me from following this method exactly. My cube probe was only an abstraction and lacked any contextualization to the real-world. It is only a self-contained probe that cannot be connected to anything. I also did not have an extended period of time to embed the cube with each tester. I was limited by the timeframe of the thesis governed by my university and thus needed to compress the method to accommodate my short timeline.

To accommodate the limitations, I adjusted the initial method according to the "Light-weight Technology Probe" framework created by Langdale et al [3]. Their "intergenerational communications system" suffered from similar limitations and restrictions that forced them to adjust the initial method. The adjustments include compressing the timeframe to a few days, and relying on the tester's imagination to envision future scenarios [3]. Their adjusted light-weight method is outlined in five steps, that I used as inspiration for modifying the method to fit my thesis.

The steps outlined in the Light-weight Technology Probe method and my interpretation of the steps for this thesis:

Step 1

"The users are presented with a description of the existing system." [3]

Each tester received a small box that included a cube probe and short instruction sheet inside a white box. The instruction sheet explains what the cube is, what is requested of the tester, and what to think about while they are testing (Figure 32). I purposely kept the instructions short and not too descriptive. I do not explicitly state how to use the cube, or how it is possible to interact with the cube. That was intentional to motivate the tester to explore possible ways of interacting with the button on the cube.

Step 2

"The users are then asked to design some scenarios that are in keeping with their goals that they think that the system could help with." [3]

The last point on the instruction sheet makes this request (Figure 32). The cube was purposely built as an abstraction of the concept to aid in contextualizing. It was my goal that the testers would imagine contexts where a similar stacked inputs interface could be used.

An additional modification related to time engagement inspired an adjustment to my method. The light-weight method encourages users to think about this step "…off and on for several days…" [3]. Instead of requesting dedicated time for each tester to be engaged in exploring the cube, I

requested casual use when they found themselves provoked or needed a break from their daily work schedule. I hoped each tester would initially explore the interactions of the cube but then go on with their daily work schedule, letting the concept linger in their subconscious for a few days. I knew it would be difficult to expect a busy tester to dedicate a lot of time to testing the probe, so I framed it to the tester in this fashion to ease their "test anxiety". My real goal for this adjustment, though, was to give enough time for the tester's body of knowledge stored deep in their brain to process the concept and envision future scenarios.

Step 3
"The researchers receive the scenarios." [3]

I modified this step, and combined it with Step 4.

Step 4
"The users arrive on-site and attempt to successfully carry out their scenarios." [3]

This step illustrates a major difference between a typical technology probe and this light-weight method. A typical probe would embed the technology in the environment of the testers to get their real-world reactions. This light-weight method brings the testers to the technology in a more stable, though foreign, environment. When a technology is not stable enough for the real-world, this adjustment to the method makes testing still possible.

My adjustment to this step and the previous step was to combine them while interviewing the tester. After a testing period of a few days, I planned on returning to the tester's office to video interview them. I would start the interviews by asking them to demonstrate for me how they explored the cube, and what interactions they discovered.

Step 5
"The users are debriefed…" [3]

The remainder of the interviews would include debriefing the testers. I planned a rough outline of questions to uncover the tester's perceptions of the cube and touch-sensitive button interface, determine their meaning of the visual light feedback, and discuss any contexts they imagined with the interface.

Programming

The software in the cube is what would give life to the multiple interfaces of the cube. Decisions related to selecting hardware

components were made carefully, in order to keep the hardware compatible with multiple versions of software. For example, RGB LEDs were selected because a specific color response could be chosen by changing one line of code, and not determined by the single color of a physical LED. A small opening was cut on one side of the cube to allow the programming board to slide in and interface with the Arduino board to load new versions of code. A minimal amount of code was necessary for detecting when a finger touched the button or pressed the button down. The majority of code is for translating sensor data into interactions and delivering a corresponding response. “Time” added another layer of complexity since many of the possible interactions unravel over time or use specific timing.

“Touch” and “press” were the obvious initial interactions included in the software. It was a design decision to correspond each of these interactions to its own color: blue for touch, and red for press. I decided this to attempt to make a strong disconnect between the two interactions, and create color associations for the tester. I would use this later during the tester interviews to determine if the tester associated meaning to the colors or saw correlations between color and interface. To phrase it another way, blue would correspond to the digital layer of interaction, and red to the physical layer of interaction.

When “time” was added to each interaction, the colored response would also be affected. Quick interactions would result with quick color responses, while longer interactions would result in longer color responses. The initial interactions were expanded to include time as a modifier. If a person touched the button longer, a new response was displayed. If the button was pressed longer, an additional response was displayed. When time was factored into the initial interactions, a new set of possibilities and layer of complexity emerged.

To expand the total sum of possible interactions even further, I used Johan Redström’s concept of *Tangled Interaction* as inspiration. Up to this point, each response is the result of an interaction with only one layer in the stacked inputs. Tangled interaction proposes “…to set layers up in ways that makes something potentially interesting happen also in-between…” [4]. Applying this concept would result in new interactions that combine actions involving both layers in the stacked inputs. This level of complexity is not present in my TRKBRD prototype and is an unexpected exploration for this thesis. The very notion of simultaneously combining interactions with both layers contradicts the third design decision of the initial Stacked User Inputs framework that created the base for this thesis. However, in the context of the abstract cube probe, combining interactions introduces a level of complexity that could assist in determining a tester’s understanding of the layers in the stack. These new tangled interactions would be represented by a “rainbow color” response by the cube; displaying red, blue, and all the other colors in between.

It was my original thought that the list of interactions listed above was only the beginning. I planned on expanding the interactions and making them more complex after conducting the first 2-3

tester interviews. I felt these initial interactions were obvious and did not offer a big enough challenge for the testers, or lay the foundation for valuable test data.

Code Version #1

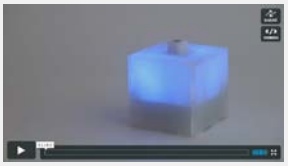


Figure 33: SUI Cube Code #1

http://www.vimeo.com/11683417

1:52 minute video

The first version of code[2] for the cube was intended to be the simplest of all the versions. This version creates the baseline that the other code versions expand on. The base interactions include tapping the top of the button similar to a tap on a trackpad, and pressing the button down. The only "tangled interaction" occurs when the button is held down for longer than one second while also touching the top of the button. The "tangled-ness" is subtle, since it is common to touch a button on its top surface, and it is a known interaction to hold a button down.

The slightly more complex interactions include holding your finger on the top surface for longer than one second, and pressing the button down without touching the top surface. I felt these interactions to be more complex since they are not typical interactions with a button. The button I chose to use for the probe allows someone to press the button down while holding the sides; however, not many button designs in the real world offer this affordance.

The list below describes the interactions and the cube's corresponding response. It is necessary to watch the short video in Figure 33, demonstrating the interactions and responses, to understand the cube interactions.

Tap

A quick tap on the touch sensor without pressing the button down.

> Short blink of blue light, matching the number of taps. For example, if you tap three times at a medium-paced rhythm, it will blink three times after you stop tapping the button.

TapHold

Hold finger on touch sensor for longer than one second, without pressing the button down.

> Blue light will slowly fade to full brightness and then fade to black. This pattern repeats as long as finger is still touching the button.

Press

Push down on the button, pressing for shorter than one second

---

[2] Code available in Appendix C.

Red light when you press down, until you release the button.

PressHoldTouch

Push down on the button, while touching the touch sensor, for longer than one second.

Red light displays initially. After one second red slowly cross-fades to green, then cross-fades to blue, and then cross-fades back to red. This pattern repeats until finger is no longer touching the button, or the button is released.

PressHold No Touch

Press down on the button, without touching the touch sensor, for longer than one second.

Red light displays initially. After one second red slowly fades to black and then back to red again. This pattern repeats until the button is released or the touch sensor is touched.

Code Version #2

The second version of code[3] includes the base and slightly complex interactions from version #1, but modifies the "tangled" interaction. Two quick presses of the button (double-press) changes the cube into a "Press Lock Mode". Once it is in this mode, a Tap and a TapHold interaction changes which rainbow color is displayed. This version of the tangle interaction is not as transparent as the previous one. The cube would not appear to have any "tangled-ness" unless this mode was discovered. I consider this mode to provide a *virtual* tangled-ness, since the physical layer is virtually locked and no longer requires to be pressed down when performing the tangled interactions.

Figure 34: SUI Cube Code #2

http://www.vimeo.com/11683492

1:23 minute video

It was my intention that this version's tangled interaction be less transparent and more difficult to discover. I wanted this interaction to unravel after the tester spent a longer period of time exploring the cube. I didn't feel this interaction would be too complex and unattainable, though. Each tester would have years of personal computer experience and be familiar with a "double-click" interaction with a button interface. The test would be whether or not they apply a mouse-focused interaction to an interface not resembling a mouse.

The list below describes the interactions and the cube's corresponding response. It is necessary to watch the short video in

[3] Code available in Appendix D.

Figure 34, demonstrating the interactions and responses, to understand the cube interactions.

Tap

A quick tap on the touch sensor without pressing the button down.

Short blink of blue light, matching the number of taps. For example, if you tap three times at a medium-paced rhythm, it will blink three times after you stop tapping the button.

TapHold

Hold finger on touch sensor for longer than one second, without pressing the button down.

Blue light will slowly fade to full brightness and then fade to black. This pattern repeats as long as finger is still touching the button.

Press

Push down on the button, pressing for shorter than one second.

Red light when you press down, until you release the button.

PressHoldTouch

Push down on the button, while touching the touch sensor, for longer than one second.

Same response as Press.

PressHold No Touch

Press down on the button, without touching the touch sensor, for longer than one second.

Red light displays initially. After one second red slowly fades to black and then back to red again. This pattern repeats until the button is released or the touch sensor is touched.

Double-Press (Press Lock Mode)

Press down on the button, twice in succession, at a medium-paced rhythm. “Double-click” the button.

Responds the same as a Press, initially, for each press. Immediately after second press, displays red light continuously to represent the “Press Lock Mode”.

If cube was in Press Lock Mode when double-pressed, there is no response while pressing each time. Immediately after second press cube exits Press Lock Mode, and all lights turn off.

Double-Press + Tap
While in Press Lock Mode, tap the top of the button.

Switches to the next color in the progression: red, green, blue. Next color stays on continuously.

Double-Press + TapHold
While in Press Lock Mode, hold finger on top of the button for longer than one second.

Slowly cross-fades to next color in the progression: red, green, blue. This pattern repeats as long as finger is still touching the button. When finger is removed, the cross-fade stops at that color and continuously displays that color.

Code Version #3

The third version of code[4] uses version #1 as the starting point, but is then simplified for quicker interpretations. This version was written after the first few tester interviews revealed the other code versions to be more complex than I had anticipated. The same interactions are possible with this version, but the responses do not maintain a correlation between interaction layer and color, and the time-related responses have a shorter delay.

The list below describes the interactions and the cube's corresponding response. It is necessary to watch the short video in Figure 35, demonstrating the interactions and responses, to understand the cube interactions.

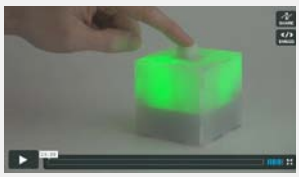


Figure 35: SUI Cube Code #3

http://www.vimeo.com/11683522

0:30 minute video

Tap
A quick tap on the touch sensor without pressing the button down.

Short blink of blue light, matching the number of taps. For example, if you tap three times at a medium-paced rhythm, it will blink three times after you stop tapping the button.

[4] Code available in Appendix E.

TapHold
Hold finger on touch sensor for longer than one second, without pressing the button down.

Green light displays after 0.5 seconds, as long as finger remains touching the button.

Press
Push down on the button, pressing for shorter than one second.

Red light when you press down, until you release the button.

PressHoldTouch
Push down on the button, while touching the touch sensor, for longer than one second.

Red light displays initially to represent the Press. After 0.5 seconds color changes to purple.

PressHold No Touch
Press down on the button, without touching the touch sensor, for longer than one second.

Red light displays initially. After 0.5 seconds red light begins to flash on-off quickly. This pattern repeats until the button is released.

Testers

The next step after building the cube probe was to find people to test it. I was faced with two challenges: who would be available to test the cube within my short timeframe, and who would give me the most valuable test data? Fortunately, I felt both questions could be answered with the same response: experienced designers.

The “Genius Design” method inspired me to use experienced designers as test participants. “Genius design relies almost solely on the wisdom and experience of the designer to make design decisions [5].” I wasn’t going to rely on the test participants to create or make design decisions related to the cube probe or this thesis, but I did want to leverage their design experience. The list of testers was carefully crafted to include local designers that I have met while pursuing my Masters degree. I chose each one because I feel I have learned something from them in the last two years. The designers work at a diverse set of companies with a focus that includes interaction design, software development, industrial design, mission critical interaction design, and mobile

phone interfaces. It was the designer's experience in their respective focus that would be the lens through which they gave feedback on the cube.

Choosing designers as testers also followed the progression of changes I already made to the technology probe method, adapting it to my decision to create an abstract probe. If existing technology was available, that could more closely follow the technology probe method, it would be best to use actual users for the testing since they could use it natively in the wild. My challenge was that I was testing an abstract concept that lacked any context to make it feasible in the wild. A natural progress for track #2 in Figure 36, was to use designers for the tests, and use their domain knowledge as the sounding board for feedback.

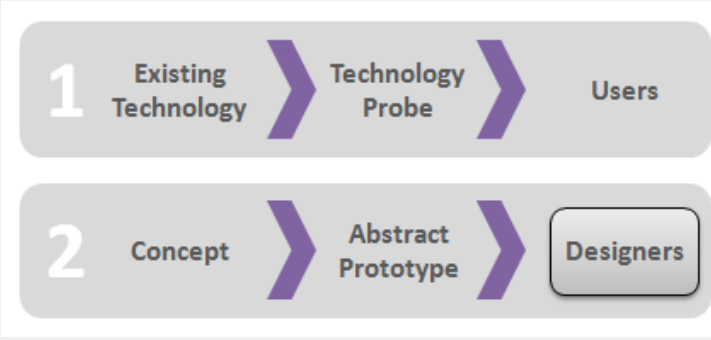


Figure 36: Two possible paths for research method.

Scheduling time with each tester would be a challenge, but was a necessary element for the probe. I wanted to embed the probe with each tester for 3-7 days to allow them enough time to explore the probe, and allow them to explore it at their own pace. With only two cubes constructed, I needed to carefully schedule the tests and follow-up interviews while remaining flexible to each tester's work schedule.

Valuable test data was the most important element in choosing test participants, and motivated my decision to test experienced designers. With an experienced designer in a related field, I would be able to communicate with them on the same level by using a language of design we both understand. With this common link between us during the interviews, I would be able to use metaphors and phrases that we both understand, to frame questions and draw conclusions from their responses. I decided to avoid "on-the-street" testing because of this lack of common language between me and the random person on the street. It would be difficult enough to stop someone on the street and explain an abstract concept to them, aside from also trying to communicate with someone that does not natively speak my language. I focused on test participants that would provide qualitative results, rather than quantitative.

Interviews

To collect the data from each test, the designers would be interviewed and video recorded after the period of exploration time. If more than one designer that is being tested works at the same company, they will be all interviewed together in a single session. The session will be video recorded to allow additional analysis after the test, of how each designer interacted with the cube and the interactions they discovered.

The format of the interview will separate the questioning into two parts: reactions to the cube, and reactions to the concept. The first half will concentrate on interactions they discovered in the cube, how they performed those interactions, and their reactions to the cube probe. The second half will attempt to get their feedback on the “stacked” concept, their understanding of the digital and physical layers, and the use of other examples to provoke their reaction to the concept. I will use the TRKBRD prototype since it was the inspiration for this research and embodies the SUI principles. I will also use the Apple iPod Classic as an example because it represents a real-world product that uses stacked user inputs.

TEST RESULTS

The intent to use a diverse set of designers was to gain a diverse set of test results, and that is exactly what I received during the final interviews. Below is a summary of the designers that tested the cube, how long they explored the cube before the test, and which cube version they tested (Figure 37). The number of designers is also included, since I did not anticipate one company broadcasting the prototype to their entire staff and encouraging everyone to take part in the test. I had only recruited three designers at Ergonomidesign, but the test included a fourth designer and her summarized results from another 10-15 people in the company also testing the cube.

The test results are split into three sections: interaction, concept, and ideas. Each section will include the major themes of research findings across all of the designers that tested the cube.

Figure 37: Summary of Testers

Company	**Do-Fi**	**ALT**	**Unsworn**	**Ergonomidesign**	**TAT**	**Systematic**
Number of Designers	1	2	2	4+	2	1
Tester Background	Interaction Design, Programming	Industrial Design	Interaction Design, Programming	Interaction Design, Industrial Design	Mobile concepts & prototyping	Interaction Design in mission critical domain
Time with Cube	3 days	6 days	7 days	7 days	Cube #1: 3 days Cube #3: 6 days	1 hour
Cube Version	#1	#2	#1	#1	#1, #3	#2

Interaction

Each cube had a range of possible interactions, depending on which version. At the minimum there were five, at the most there were eight. The designers were aware that many interactions existed in the cube, since that was the focus of this thesis and the included instructions clearly stated to explore the “multiple layers and multiple interactions”. Many of the intended interactions were found during the tests, but many were not. During the tests I watched the designers to find out which interactions they discovered, and whether or not they could find the hidden interactions. I asked them series of questions to investigate their understanding of the color responses and perceived independence of the interactions. Other results that emerged unexpectedly were

the importance of time with each interaction, and the various positive and negative affordances of the button interface.

Found/Not Found

Figure 38 below illustrates the compiled findings of all the possible interactions in the three versions of the cube, and which interactions were found or not found with each testing company.

Figure 38: Summary Interactions Found

Company	Do-Fi	ALT	Unsworn		Ergonomidesign	TAT	Systematic
"Tap"	Yes	No	No		No	No	No
"TapHold"	Yes	Yes	Yes		Yes	Yes	Yes
"Press"	Yes	Yes	Yes		Yes	Yes	Yes
"PressHoldTouch"	Yes	Yes	Yes		Yes	Yes	Yes
"PressHold No Touch"	No	Yes	No	Yes	No	Yes	Yes
"Double-Press"	n/a	No	n/a		n/a	n/a	No

The "Tap" interaction was the only interaction that did not have a one-sided or split result. Only one designer out of six found this interaction. Some of the remaining designers found the interaction during the interview, but only as a result of collaborating with their coworkers or purely by chance. In some of the interviews, it was necessary for me to introduce this interaction to the designer, and explain the proper tap speed and the fact that it counts the taps back to them. Each of these designers had an initial perplexed and surprised reaction to the novelty, but soon relaxed after they realized the familiarity of the interaction. Once the speed threshold was mastered, each designer was quite comfortable with the tapping interaction and found delight in the fact that the cube was capable of counting.

"Whoa! That's never happened before! Has that happened to you?" (Ergonomidesign)

"I'm a bit proud. [after discovering the Tap interaction after many attempts, and without any instruction]" (TAT)

Tapping on a touch interface as become so common through the widespread adoption of touchscreens for mobile phones and devices, that it was not a foreign interaction for the designers once they knew that the interaction was possible.

Contrary to the "Tap", the "TapHold" interaction was quickly discovered. These two interactions were not as connected and paired as closely as I anticipated they would be. The affordance of

seeing the metallic threads on top of the button was strong enough to anticipate its function.

"I can see the sensor here, probably where it connects to my skin." (Systematic)

"…it literally raised its hand up and said 'Something is happening here!' That was a clear affordance." (Ergonomidesign)

"…the thread is very eye catching, it's very obvious that its not just a button…" (TAT)

"Press", of course, was the absolute first interaction discovered by each designer. I was not reprogramming the interaction, or affordance, of the button. I wanted the button to perform as a button should perform, with the interaction of a simple press. This interaction was not foreign to any of the designers, and was easily discovered.

"There is an affordance with the object. It's an apparent button, you press it." (Ergonomidesign)

The subtle difference between a "Tap" and a "Press" were completely lost, though, since the "Tap" was not found by most designers. A fair amount of code is necessary in the cube to differentiate between these two interactions. A quick press of the button could be interpreted as a tap, unless these interactions were programmatically accounted for. Just as the TRKBRD prototype had to be smart enough to know the difference between a "click" tap and a key press on the keyboard, the cube was smart enough to know when you tapped on top of the button or quickly pressed the button.

All of them also discovered the "PressHoldTouch" interaction that requires the combination of the "Press" and "TapHold" interactions, executed by simply pushing down on the top of the button for a longer length of time. Only four of the six designers received a reaction to the longer press, being the slow fades through the rainbow of colors. The remaining two designers discovered this interaction but did not receive any reaction from the cube; the cube just remained the red color. Time was an important element to this interaction that sometimes inhibited its discovery.

The "PressHold No Touch" interaction is not one that is typical for a button, which explains why only about half of the designers discovered it. This interaction involves pressing down the button without touching the threads on the top of the button.

"This one for example, it's quite tricky to get this one (PressHold No Touch). By the time you get this one, it's not like you just fiddled around, you have to… press the button in a way that you don't normally do. I never press buttons like this. But that's how I have to press it to get that one." (TAT)

"…it's a button…not a normal way to hold down on the sides." (Do-Fi)

I believe that the designers that found this interaction did so through the use of a loose Scientific Method [7]. After discovering the “Press” and “TapHold” interactions, and deducing what types of sensors were in the cube, their next step was to attempt to remove different components in their interaction to search for a new reaction.

“When I saw that just touching the thread did something, I thought, maybe pressing without touching would do something too.” (ALT)

“What made you think of pressing it that way?”
“Just basically confusion. I didn’t understand why it acted differently so I tried to explore. Can I make it work without touching the middle part? I tried to touch it without the sensors to see if they have an effect and I learned that it has an effect.” (Systematic)

The “Double-Press” interaction was added to one version of the cube to create an advanced, “hidden” interaction. I purposely made it hidden to test if a non-transparent interaction could be found in the interface. I also wanted to test a more advanced interaction that would not be easy to find or replicate. Similar to the “Tap” interaction, the “Double-Press” could be executed by pressing the button down twice, at a rhythm of about two presses in one second. Once the interaction was performed, the cube would turn solid red until the “mode” was exited by performing another “Double-Press” interaction. While in this mode, the “Tap”, “TapHold”, and “Press” interactions were all possible to affect the changing or cross-fading between the rainbow colors.

Neither of the two designers that received the double-press-capable cube, found the “Double-Press” interaction. Each designer needed to be provoked during the test interview in order to think of double-pressing the button. I was careful during the interview to not directly tell them how to perform this interaction, but to ask them to think how they interact with other types of devices and then try to translate those interactions to the cube. The speed threshold of the double-press was a factor in the failure of the designers to find the mode. They would either press the button too fast, too many times, or too slow, in order for the cube to recognize their interaction as a “Double-Press”.

The designer from Systematic illustrates this progression of provoking questions and the Scientific Method [7] to eventually discover the “Double-Press” mode.

“What are interactions you’d use with other buttons?”
“Yes…the hardness of the press, the direction of the press, maybe also the gesture of the press…maybe the number of presses…1, 2, 3 (presses button three times quickly)…”

“Explore the number of presses.”
(pushes button many times, and sometimes holds down button on the last press)

“Simplify it…”
(laughs) “I guess I need to play with it a day or so before I learn it. Sorry, I’m a bit lost.”

The designer continues to casually press and interact with the cube while explaining his thoughts on “hidden interactions” (which will be disclosed later). While talking to me, he inadvertently locks the cube in the “Double-Press” mode.

“Do you notice what state it’s in now? How did you get that?”
“I have no idea…”

“If this was a light switch on the wall…what are some simple interactions…?”
(pauses) “The duration of the press could be a light switch…press (attempts variations in pressing), unpress…(inadvertently locks into mode) There it came!”

“What did you do to get that?”
“I pressed a number of times, I guess.”

“How many times?”
“2, 3, 4?”

“Let’s say this is a mouse at your computer. What interactions would you apply to this?”
“If I apply this to a mouse interaction… (pauses) …I’ll see if I can provoke it… 1, 2, 3…(presses button three time is a row, multiple times, presses variations of multiple presses)…(eventually presses a perfect “Double-Press” interaction) Two presses! So, it’s two presses. And two presses off (unlocks mode). I got it.”

Hidden Interactions

There were two hidden interactions in the cubes, “Tap” and “Double-Press”, and both had a superb performance in remaining hidden for most of the designers. Each cube included the “Tap” interaction and only one designer discovered it. Cube version #2 was given to two of the six companies and included the “Double-Press” interaction to lock the cube in a mode. Neither of the two designers found this “hidden” interaction mode.

The original intent of the “Double-Press” mode was to create an interaction that was not obvious, and to show the depth at which the designers explored and investigated interactions on the button interface. What resulted in the testing, however, was a confirmation to my decision to use experienced designers for testing. It was the designer from Systematic, in particular, that gave valuable feedback on the hidden interactions through the lens of his own professional experience.

“Definitely the modes tricked me. This really proves that modes is a bad thing in most cases. Most are extremely non transparent and hard to grasp for users. If you’re in a mode you need to state it clearly. Without visual indication of the modes, it’s very hard to get that…and that actually confused me a lot because I saw some of the lit-state (“Double-Press” mode) interactions at first, and then I

continued using it and I lost the state without knowing it and I saw more. I was jumping through the matrix (matrix of possible interactions, in and out of the mode)…and didn't know it. It didn't seem to have consistent interaction because I didn't know I was jumping between states. That's a classic error of designing states. This is why you should really avoid it and clearly state states when you have them." (Systematic)

The designer recollects a real-world example of the remote control interface for a (now) old-fashioned slide projector with a carousel of slide photographs.

"They tried to simplify the design and take the 'back' and 'forward' button, and merge into one button. One short press is forward, and one long press is backward. This is a classical transparency error. Because… they did simplify the control, but this merging of controls, to the novice user, was so strange that what they experienced was… sometimes it goes forward, and for some reason it goes backward sometimes. If they don't read the manual, and most people won't… you would definitely need training. And you would need to fine tune that hidden interaction." (Systematic)

The designer's mission-critical interaction design experience brings focus on elements he takes into consideration, and how they relate to the stacked and hidden interactions of the cube.

"Accessibility is a big factor. If I worked in a moving, and high-paced environment, like inside a vehicle, or in the pocket of a soldier, those hidden interactions and fine-tuned small details are very problematic. For a specialist in a control instrument panel situation, it's different. In some accessibility situations all those states are problematic. Because you simply can't enter them with the same context all the time, with the same meaning." (Systematic)

"If you do stacked UI or hidden interactions, you really have to consider the actions and you really have to test to a lot of users just to see if they can hit that hidden interaction every time. If they lose trust in the interaction, it only swipes some of the time, they will get irritated with the system. In my context, it has to work in a stressful situation, so you have to make sure the same interaction performed in 50 different ways does the same thing every time. I have implemented some hidden interactions in some designs, but they have to be fewer and more considered." (Systematic)

Meaning & Independence

I purposely showed red or blue color feedback for certain interactions on the cube to create an association of the color with the action necessary to receive the color. Blue color responses would appear for touch-related interactions, and red color responses for physical interactions.

An unexpected twist occurred while questioning the designers about their perceived meaning to the color responses. While they were almost unanimous in feeling that the colors had no meaning…

“Do the colors mean anything to you?”
“Not really. I discovered red first. Maybe if the thread [touch interaction] turned green, I would have thought yes and no, positive and negative. But since it was red and blue, maybe I should have thought warm and cold, but I didn’t.” (ALT)

“No. To me…in this context they mostly speak prototype.”
“It seems the edge press (“PressHold No Touch”) provides the red one. And the gentle touch (“TapHold”) makes blue. And depression, sinking button makes red. I hadn’t thought the mapping would be significant. I wasn’t looking for a symbolism.”
“I think I was, of course the slow pulsing fits with the gentleness, it’s waves, cool blue. I haven’t had the emotional attachment. For me, it’s been more placeholder, prototype…” (Unsworn, between both designers)

(all shake heads “No”) “I think the blue is the only one that I found some sort of association with as being with the tap or the stroking (“Taphold”). And red always appeared when I pressed it. But it was not an ‘Aha’ realization, it was more of a maybe…” (Ergonomidesign)

“Not really.” (TAT)

“No, I just saw them as rewards.” (TAT)

“Values on a scale. Some of them pulsate and some of them cycle through the scale. Some go on and off. To me it just means value representation… All the colors are equal, they just have a place in the spectrum, that’s all.” (Systematic)

“I connected the blue to the tap action, and the red to the press action.” (Do-Fi)

…When asked if they connected a color to part of the interface, the designers that previously did not see a connection, answered the opposite.

“Yeah, of course. Red is connecting to pressing and blue for touching.” (ALT)

“Yes.” (TAT)

The designers had similar mixed responses when questioned about whether they saw any independence between the physical and touch interfaces on the button. Independence was an important element of the TRKBRD prototype, since someone using the device would need to be able to type normally and control the cursor, without one interaction interfering with the other.

“Saw it with just a tap.” (Do-Fi)

“I didn’t think of it in that way really” (ALT)

“I’m thinking, is the red more important than the blue? Since pressing is more of an action? Touching is more subtle sort of. If you have to rank red and blue, is red higher, or just two different things?” (ALT)

"Yes. I explored that you can use the sensor without the button press. And you have the button press without the sensor. Then you have the combination of the two interaction types. And they all do different things. So yes, they do seem to be independent." (Systematic)

It is a surprise that meaning for the colors and independence of the interactions were not as strongly perceived as I felt I designed them. Multiple reactions were included for both the touch and press interactions, with each reaction staying within its assigned color. I can propose three reasons why meaning and independence failed to make an impression during the tests.

The internal electronics in the cube provided two LEDs to show the color responses, but the LEDs could not be controlled independently. Both LEDs would show the same color at the same time. If the LEDs had been wired independently, and then controlled separately in the programming, two separate colors could have been shown at the same time. This redesign would mean that when the designer used the touch layer, the LED on the right side of the cube would show blue. When the designer used the press layer, the LED on the left side of the cube would show red. When both touch and press layers were used, both LEDs could show the necessary response. By dedicating an LED to each interaction layer, and independently controlling each LED, it may have strengthened the independence of the layers with the designer too.

The time each designer spent exploring the cube was viewed as a playful activity, and not as a scientific exploration. Many of the designers commented on the playfulness of the cube, and how it became a game to find the various interactions. I am pleased that the playfulness became a motivator for exploration, but it was at the expense of gaining more thoughtful reflections on the meaning and independence of their explorations.

The final reason is the lack of context with the cube. The cube was purposely made abstract to explore the concept, and purposely held away from context to not color the feedback with characteristics of the contextualization. This lack of context potentially prevented meaning and independence from being important to the designers. For example, independence is crucial to the TRKBRD prototype. Without independence a user would not be able to type without constantly "clicking" the cursor at the same time. The TRKBRD has context that makes a person aware of the consequences if independence is not existent in the design. Without context in the cube, the consequences were not visible, which in turn allowed the designers to not care or even reflect on the importance of independence.

Timing

As stated earlier, "time" was an important element in the cube tests. Some of the cube reactions do not appear until after a half or a full second on interaction. The reactions also have an element of time when they slowly fade in and out, or fade to another color slowly. I did not anticipate the amount of feedback I would receive, related to time, from one set of designers when they

became frustrated with the cube. I had two designers testing the cube from the company TAT, which works in the mobile device domain. They initially received cube version #1 and explored it for three days. On the third day I interrupted their testing and changed their cube to version #3, which they then explored for six days. The reaction time delays from cube #1 to cube #3 are cut in half, and the slow fading was changed to a quicker blink. This drastic change in cube behavior sparked a flurry of feedback, coincidently, directly related to the domain in which they worked.

The immediate feedback was that cube #3 was better than cube #1. Faster response time meant a better interaction. One designer even likened the delayed reactions and uncertainty to superstition. With some cube reactions being delayed for one second, he didn't know if it was initiated by something he did before, during, or after that delay.

"On this one (cube #1), I thought it was difficult to interpret what the cube understood what I was doing… when it gets that vague, sort of non-direct feedback… people almost start to get… superstitious… 'Was it because I held it [upside down]? Did I hold it like this [on its side]?'" (TAT)

The delay in reactions also prevented some interactions from being discovered quicker, or discovered at all. Some designers did not have the patience to wait for a potential new reaction, or were quick to think that the reaction was the same as another.

"If I just press down and hold, it changes into the green color. Just before it turns green, it fades away, in a way, to almost whitish (when showing fade between red and green, orange and yellow are faint in daylight). When you do like this, ("PressHold No Touch") as soon as it comes there (when red fades out), it seems like it's going the same way in turning into green, so you sort of release it…"nothing else happened." I can remember that I did something like that. I don't know if it was intentionally to not touch the actual touch surface or if it was something else. I remember doing it but I didn't find it because I didn't have the patience to sort of wait and evaluate." (Ergonomidesign)

The designers from TAT explained why "time" is so relevant to the domain they work in. It is not only important for reaction times to sensory input, but also in how time is perceived to someone using their interface on a mobile phone.

"…the technology providers come to us and they say 'We need this to look super fast!' …and then we (TAT) want to do animations and we want to do transitions and stuff like that, and obviously they aren't too keen on that. For example, we have 200 milliseconds to make an animation for opening an application." (TAT)

"For us it's very important to, as soon as possible, tell the user that 'Yes, what you just did had an impact'…and then trying to figure out what that impact is, and tell the user what it was. I feel that is happening way more with this one (cube #3), than this one (cube #1), for some reason. Because this one (cube #3), as long as I'm doing something ("Press" on #3) this one keeps getting me signals on an instant level. Whereas this ("TapHold" on cube #1),

yeah, this one to me is not telling me the whole story. It's not 'Yes you are pressing it, I'm about to go into a long press, and here is the long press ("TapHold")'." (TAT)

And another thought on cube #3 continuously projecting signals to its behavior, and how it reminded one designer of a specific animation side-effect from cartoons produced in the 1950s to 1980s.

"I see that a bit happening here as well (cube #3). When this is fairly instant ("TapHold") and then that happens ("PressHold"). I quite like this one ("PressHold No Touch") because it turns into something else. It's not 'action, time, and then something else'… it's somewhat more continuous." (TAT)

"It's the Hanna-Barbera effect[5] [8]. Hanna-Barbera, because in all of their cartoons, you can always see when something is about to happen and where it's about to happen, because they paint that differently. This (cube #3) gives a bit of that impression also. It's a good thing, because you can recognize when something is about to happen." (TAT)

During the interview I mentioned why I purposely delayed some of the reactions of the cube, to promote the exploration of interactions over a person's time spent with the cube.

"There is time with you and the device, and then there's time for the lifespan of you and the device. For a phone for example, if you can call with it, you can unlock it, it's all these instant things… when you press a button it's suppose to react. There's also time in, for example, finding the long press. It's not something you'd expect a user to do, just picking up the phone, or, accelerometer input is probably something that someone tells you 'hey, have you seen this?' (changes orientation of phone) …or you accidently do it, and find it." (TAT)

"Did you know you could take screen shots on your iPhone? Time is a factor, but I'm not sure. For me I don't think of it as time, it's more visibility I would say, of how to do that thing, or why you should go looking for it. I mean, when I found the screenshot, yes it took time before I found it, but it's also fairly invisible thing to happen. And the invisible part you can control, and the time you can't." (TAT)

For the sake of the abstract cube probe, I do not regret incorporating time into the interactions, and feel it was necessary to explore the "time" element on different levels. But the points that are mentioned while contextualizing the cube into the domain of the designers are valid none the less. There is a difference between spending time to understand how to interact with a device on a basic level, and spending time to uncover hidden interactions that are not necessary to know on a basic level.

---

[5] "…in all of those early Hanna-Barbera cartoons, anything that was going to animate would be drawn in a lighter color than the backdrop scenes." [8]

Affordances

The perceived interactions of the cube and button made the various interactions possible or impossible to discover, as mentioned in the previous "Found/Not Found" section. I intentionally wanted the affordance of a "button" to make a person's initial interaction with the cube obvious and intuitive. The original technology choice for creating the digital layer of interaction (touch sensitive) would have made the digital layer completely transparent and lack any affordance. A virtual "touch field" would have been embedded inside the button, in turn making the outside of the entire button touch-sensitive. Because of difficulty in getting the technology to work, and my timeline in delivery the cubes for testing, I abandoned this first choice and decided on the ease and simplicity of sewing conductive thread through the top of the silicon rubber button. A side-effect of using the thread, however, was a strong affordance to the digital layer of interaction on top of the button.

Design decisions that I made throughout the process of designing the cube probe also created false positive affordances and potentially consumed valuable exploration time of the designers. I attempted to be direct and clear in the initial instruction sheet included with each cube, that the area to explore was the button, but that was apparently not enough to curb the curiosity of the designers attempting other affordances. All but one of the companies explored interactions that were not being sensed in the cube.

"We also tried, though it clearly said that we should work with the button… we tried to speak into it, while and not while pressing the button. It didn't provide any feedback." (Unsworn)

"…I wanted something to happen when tilting it (tilts cube in hand). Or shaking it (shakes it). Or putting it upside down [on the table]. To me that felt like a natural interaction to have something happening to it, but it didn't." (Ergonomidesign)

"I was dead sure it was a joystick. There were four buttons at least."
"It does feel like 4 buttons." (TAT, between both designers)

"I can't feel the difference between moving the button and pushing the button. So I might be lead to think that direction is an option, but it's not." (Systematic)

"One expectation that the thing hasn't met, is that I thought the button itself would do something when squeezed. It has this unique elasticity. It's rare to see a button like that (squeezes button). Maybe you can squeeze it? This far I haven't found any squeeze." (Unsworn)

The following dialog between two designers at Ergonomidesign illustrates how decisions that I made in engineering the cube, choosing materials for construction, and selecting a button for the interface, all affected the overall experience. These decisions created unintended affordances, misguided interaction exploration,

and potentially degraded their impression of the core concept that was my intent for them to explore.

"But I would argue that the size of the box is an affordance to pick it up. By that then you probably expect it to react when you lift it up."

"…the places where there are affordances, those affordances are not obviously made into outputs. Like when you see, yeah, the box wants you to pick it up, and the button wants you to press it, but when you press and tilt (tilts cube) there's no obvious output. And therefore that affordance is probably not utilized as best. And the affordances that are there are the ones most obvious: button needs to be pressed. The joystick… I'm not sure if that's an intended result, or was it the mind playing tricks, or accident."

"I think this whole thing with the tilting is also connected to the engineering of the actual plastic [of the cube]. It's very exact, and the shape of the button is pretty exact too. But then when you press it and you feel that it's going down, wobbling down, then the experience doesn't match the appearance. So that makes you unsure if the tilting is actually intentional. I could speculate. It looks like a piece of precise engineering. But then when you press the button it doesn't behave like what it looks like." (Ergonomidesign)

Concept

Aside from feedback from the actual interaction with the cube probe, I received more general comments later in the interviews once the designers fully understood all aspects of the probe and concept. At this point I had received all the feedback on interactions each designer found and didn't find, and I also fully disclosed when they were correct or incorrect in their assumptions. By removing any uncertainties they had, we were able to have a comfortable dialog about their thoughts on the concept. Two themes emerged as they expressed their frustration in how to talk about the interactions they discovered, and how their definition of "stacked" differed from mine.

Lack of Vocabulary

Many of the interactions that I provided in the cube do not have a place in the vernacular of describing physical interactions, yet. I had to be creative from the beginning in how I labeled these new interactions so they hopefully felt comfortable to say, while adequately describing the interaction. Half of the designers also felt this absence of available vocabulary while communicating their findings during the interview, but also while they collaborated to explore the interactions.

"One of the difficulties with what we are doing, or achieving, is that some of the actions like this sort of button press… 'pressing without touching the wires'… we don't have the vocabulary. We have the word for double-click, or a drag and drop, or a long press. But "only touch the edges please"… we don't have a snappy

word for it, yet, in a way. I don't think there are that many buttons that make that word necessary." (Unsworn)

Another example, this dialog between the two designers at TAT:

"It does mix it up. It's also the lingo around it. A button, you can press it, you can press it twice, press it for a long time. But then comes the touchscreen, or touch interface, and that one you can do other type of stuff. But then the combination of all those things is… I don't know how to express that. It's like, if I say 'double tap' that's…

"Mouse, touchscreen interaction…"

"…it's common knowledge what is suppose to happen, but 'press and somewhat press it sideways without touching the conductive thread' is not something that someone would just go 'Oh, yeah, of course, why not that one.'" (TAT)

When specific terms were unavailable to describe the interaction with the interface, they relied on known terms that simply described their action. During the interview, two designers at Ergonomidesign labeled the "TapHold" interaction as "stroking". I asked them why they used that term, and if it was a part of the design language they use on the team.

"I think that's nothing about a specific language, just the way it happen to show the first time."

"I used the word "stroke" and "rub" as well. I felt those were the most obvious things I was doing to the button and they were very related to projects that I'm doing currently which involve multitouch, and involve two finger stroke and one finger stroke." (Ergonomidesign)

"Stacked" Concept

The second half of each interview involved explaining the SUI concept in more detail, and hearing each designer's reinterpretation in return. It unintentionally became a valuable way to hear initial reactions framed by the designer's experience and design domain. The Unsworn designers in particular, pursued understanding of the concept by attempting to redefine the term from various new angles.

"In my mind I'm moving away from the more general concept of stacked interfaces, to touch sensitive buttons. Even though it doesn't work like this here (TRKBRD). I'm thinking of it more literally, that it's really on top, stacked. Not in the sense of functionality, or layers of functionality, but on top."

"How you mean?"

"There is this functionality above…"

"This sort of contraption (TRKBRD), it's stacked, it's on top of the normal thing." (Unsworn)

One of the designers used various wrenches that were on the table to pile them on top of each other to demonstrate his understanding of “stacked”. They also mention the shutter button in digital cameras, which have a “half press” to autofocus, and a “full press” to capture a photo.

“A stack for me, is also its something where you can’t access this (the bottom wrench in the pile). A stack in programming is a structure where you’d have to remove all these [top wrenches] in order to get down there [to the bottom wrench]. You can’t do this straight away (remove bottom wrench without removing top ones first).”

“No shortcuts…”

“Exactly. Same as the camera button, you can’t shoot without first auto focus.”

“With this one you can, with the edge press (go to a bottom layer without removing top layers).”

“There’s some trickiness there, with the notion of a stack, as something like this (pile of wrenches). In the sense of digital technology, that’s my association with a stack or stacking.”

“If you called it ‘layered’? Would it be more accessible?”

“Yeah (pulls out middle wrench without first removing top wrenches).” (Unsworn)

I understand his definition of “stacking” because of how the TRKBRD prototype physically demonstrates the concept. The prototype rests on top of a laptop keyboard, making the “stacking” obvious and literal. This is not the only potential way of implementing the concept in a TRKBRD though. Other types of technology could make the TRKBRD completely transparent to a user, while it is hidden inside the keys on the keyboard.

I do not see “touch-sensitive buttons” as the only method of implementing a SUI, and therefore avoid using this as a way to describe the concept. It could be argued that all of the examples that I include in this thesis could be distilled down to each being a touch-sensitive button, but I only see that as a limitation of the technology, materials, and creativity currently available today.

Aside from physical interpretation, they also explored if the concept could be represented only by software. They used a remote control as an example. Some buttons on the remote have two outputs possible, with one of them requiring the press of a “Mode” button first. Pressing the button will type an “A” letter. Pressing “Mode” before the button will type a number “1” instead.

“There’s something I’m trying to understand here. If this button only would only perform ‘A’ or ‘1’ then pressing normally would be ‘A’, then pressing for a longer time would be ‘1’, for instance. That’s how some cell phones work. So technically then, it’s just a button, and it’s in the code it’s interpreted. Would that be stacked user inputs? That there is different mode or response possible just

based on temporal differences with how the input is executed. Would that fit SUI paradigm?"

"How would you answer that?"

"I think it would be good for you to include that also, but I'm not sure why I think that. I think that with this contraption (TRKBRD) its physically stacked. Could provide some confusion, for what it means. Perhaps some SUI solutions would require different technical solutions that coexist on the same button. But they could also just be nifty interpretations of the user input. But for the user it's not important how its technically solved.

The iPhone doesn't provide… it doesn't matter how hard you tap, it's just a tap. But some people would like to tap harder, for instance in drum machines. But it can't be done. But you have the seismographer, so if you check that, you can have the sensitivity to drum gently or drum harder, if you check the touch pad and accelerometer.

I think it would be a good idea to say 'Yes it could be both, stacked technologies and stacked interpretations of user input.' I think that could be useful to not mix up the model too much.

It's very interesting, the thing. In a way I think you're providing, you're solving some issues of complexity, but also providing others, in a way. The trick is to make this natural (SUI concept in TRKBRD). This is a new way. It might work if it's crafted well enough. It's a good thing to investigate." (Unsworn)

I appreciate the feedback the designer offers to shape the definition of the concept, but do not feel that "software interpretation" by itself is enough to define a SUI. Software interpretation is a major component of the cube probe, since the interactions of two interfaces are being interpreted into more than two reactions, but it is only a component of the cube and not the only component. The cube uses a combination of hardware and software to create the stacked inputs, and relies on both to fully realize the concept.

I would relate the use of "software interpretation", to how a computer mouse is an example of a time-multiplexed input device [9]. Software interpretation is used to understand how to react to a mouse click, based on the context and digital object that is being manipulated.

Other feedback provided guidelines to consider while designing a SUI.

"It's taking the input from all these things (sensors)… not empowering the user, just making it easier for the user. Sensing what he/she is about." (TAT)

"I think the collision of typing 'L' and when you happen to move the mouse [on the TRKBRD], that's the thing you want to get rid of. I think it's the collision because the tap is close to move, but it's also close to pressing the button (key). Even if you know how to tap, you have to focus in order to do it. For example, when

you're opening a file on your desktop with your mouse, it's a point a click… your brain is not registering what's happening. You're allowed to not be too careful." (TAT)

"I think it's about finding the one simple thing." (TAT)

I appreciate the "one simple thing" feedback, and relate it to the iPod classic scrollwheel interface. It is possible to download and install a new operating system for some iPod models, hacking it to create a new experience and graphical user interface. The scrollwheel can be hacked to perform additional functions than Apple currently provides. But in the end, there is an elegance to the simplicity of the scrollwheel interface in how it is one simple thing and not a complex interface control.

"…I think that is one of the challenges with this one (cube) is to allow that sloppy behavior. I think 'lean back' and 'lean forward' interaction… you're suppose to have this (leans back in chair while trying to interact with button) 'lean back' type of interaction…" (TAT)

"I think one of the major challenges is to make an even more complex stacked interface easy to understand. So easy to understand that, I put my finger on it, I move it, and I click. Maybe eventually we are used to double clicking because we do it on a PC. But it's a challenge… we played around with it, we are experts, we are somewhat motivated to explore, but I think to implement it into a consumer everyday product, the challenge is to allow people to understand very easily." (Ergonomidesign)

Ideas

The third point on the instruction sheet included with each cube, asked the designer to imagine the SUI interface in context, as a potential interface to something in real life. I included this request to provoke the designers to imagine the interface in a situation or context. My expectation for concrete ideas was low, though, since I knew first-hand how difficult this task was. It was simply an extra question at the end of the interviews to collect any immediate ideas that came to mind.

"I work a lot with touchscreens. And one thing you don't have, when you work with touchscreens, is the mouseover. This, in some way, could give you the mouseover effect." (Do-Fi)

"I like the whole layers thing. If you could have a proximity sensor and move your finger and interact that way, and then put your finger down on the touchscreen and do other things. I think that would be awesome. It could also be used for having velocity on a touchscreen. If you have a layer you could pass through really quick. You could do it here (on a cube)… if you do like this (push button) you could measure the time it hits the conductive thread and the button is down and then you can calculate velocity. And that could also…you could do 'hard tap' and 'soft tap'. Which in UIs in general are not that useful, but if you do a drum application…" (TAT)

“One of the things, is that we struggle with these 3D UIs. The current thing about 3D UIs on a phone is that you don’t have any way to manipulate 3D. When you sit with your computer and you have your mouse and keyboard and all these things, you can strafe and go in the Z-axis… and totally manipulate that environment. If you have flat glass that allows you to touch it then you’re pretty screwed. Then you have to have onscreen ways, or gestures, or stuff like that.” (TAT)

“Remote control for a TV. The Samsung TV has a really small thing you get with it, which is just a round disc with right, left, up, and down… I could imagine something similar technology wise. I wouldn’t want it to look like that (like a cube), but, maybe if you could do one thing by pressing, and another thing by swiping, or by tapping. Then you could extend the functionality, but at the same time it’s quite complex because it’s a lot of things you’d have to memorize and learn. Learn and then remember in the long run.” (Ergonomidesign)

“When you have a phone with a limited number of keys… Many of the phones have these letters as well. (letters under numbers) You’re supposed to enter names and stuff. If you could by putting your finger [on the key], and then you select by stroking your finger, you find the actual ‘B’ or ‘C’ and then you press. And then you go to the next and you find it… instead of having to tap multiple times to find the letter.” (Ergonomidesign)

CONCLUSIONS

The results of this thesis include an expanded definition of “SUI”, and a collection of guidelines to consider when designing an input or interface that is “stacked”. This is not a conclusive definition or list of guidelines. This thesis tested limited aspects of the concept with a small group of designers. More exploration, testing, and designing are still necessary to better understand the potential and future of stacking user inputs.

Stacked User Inputs (SUI) =

- “Same-space-multiplexed”, each interface having its own control yet occupying the same physical space.
- Independent control of each interface.
- Definition of “interface” loosened from my original framework, and can include limited or partial access to entire interface’s functionality.

Guidelines to Designing Stacked User Inputs

- **Find the one simple thing.**
  The iPod Classic design excels in having a simple and minimized stacked interface. Consider limiting the functions that are stacked to one essential function, though it could be the most used or most important one.

- **Design hidden interactions with consideration.**
  Hidden modes or interactions will take a person time and/or instruction to discover and feel comfortable performing.

- **Keep independence.**
  Each interface or function within the stack should remain independently controllable from other interfaces or functions in the stack. It should be possible to use on interface without affecting or augmenting the other interface(s).

- **Affordance will affect learning curve and understanding.**

  The materials used, shape of interface, size, color, symbols or markings, necessary motion for controlling, the feeling experienced while controlling, and numerous other factors… all create affordances to use, misuse, and abuse. Consider how affordance will impact each interface on its own, and a person's understanding of the stack.

- **Bimanual control is not common, but not impossible.**

  Consider how the SUI will perform and react while using two hands (bimanual) to provide input. Similar to multitouch input, each hand has the potential to provide input to the digital or physical interfaces of a SUI. Reconsider all guidelines with a focus on bimanual control.

- **Match function to action.**

  To strengthen expectations of an interaction, match the characteristics of the physical action to properties of the function it interfaces. For example, the press of a button could interface to a more exact and committed action such as "start recording", while a gentle touch on the button could be "playback", and holding your finger on the touch sensor could be "slow playback".

- **Response times can be superstitious and advantageous.**

  Slower response times to an interaction could make someone superstitious of the interface and not trust his or her interactions. However, slower response times could offer the ability of creating interactions that develop over time. 200 milliseconds is a preferred response time, for example, for one mobile design company. The "Hanna-Barbera Effect" [8] can offer a person understanding of his or her interaction and the resultant effect.

- **Create a vocabulary.**

  Some interactions proposed by designing a SUI do not currently exist in many other devices. Create your own vocabulary when designing. Only with time will a common vocabulary develop and stick in a common vernacular.

THANK YOU

1scale1
Aaron Mullane
Adam Politanski
Aixiz Lasers
Amanda Bergknut
Anders Bohlin
Apokalyps Labotek (ALT)
Åste Laberg
David Cuartielles
David Sjunnesson
Do-Fi
Electrokit
ELFA
Ergonomidesign
Erik Sandelin
Greger Isaksson
James Haliburton
Jenny Nordberg
Johan Larsby
Johan Redström
Jonas Löwgren
Jörn Messeter
Katrina Anderson
Lena Edman
Lennart Andersson
Magnus Torstensson
Magnus Wallon
Marcus Ericsson
Matt Goble
Mikkel Michelsen
MonkeyMan
Niklas Wolkert
Per Linde
Petra Lilja
Rahul Sen
Rikard Lundstedt
Sebi Tauciuc
Soo Basu
Suzanna Kourmouli
Systematic
The Astonishing Tribe (TAT)
Tony Olsson
Unsworn Industries

REFERENCES

1) Zimmerman, J., Forlizzi, J., & Evenson, S. (2007). *Research through design as a method for interaction design research in HCI*. CHI 2007 Proceedings, 493-502. Retrieved from ACM database.

2) Hutchinson, H., Mackay, W., Westerlund, B., Bederson, B. B., Druin, A., Plaisant, C., Beaudouin-Lafon, M., Conversy, S., Evans, H., Hansen, H., Roussel, N., Eiderbäck, B., Lindquist, S., Sundblad, Y. (2003). *Technology Probes: Inspiring Design for and with Families*. CHI 2003, 17-24. Retrieved from ACM database.

3) Langdale, G., Kay, J., Kummerfeld, B. (2006). *Using an Intergenerational Communications System as a 'Light-weight' Technology Probe*. CHI 2006, 1001-1006. Retrieved from ACM database.

4) Redström, J. (2008). *Tangled Interaction: On the Expressiveness of Tangible User Interfaces*. ACM Trans. Comput.-Hum. Interact. 15, 4, Article 16 (November 2008), 17 pages. DOI = 10.1145/1460355.1460358 http://doi.acm.org/10.1145/1460355.1460358

5) Saffer, D. (2007). *Designing for Interaction: Creating Smart Applications and Clever Devices*. California, New Riders.

6) Apple Keynote event. January 27, 2010. *Apple Announces iPad*. Podcast available in iTunes.

7) *Scientific Method*. Retrieved August 21, 2010, from Wikipedia: http://en.wikipedia.org/wiki/Scientific_method.

8) *Hanna-Barbera Effect*. Retrieved on August 21, 2010, from Ombwah's blog: http://www.dopass.com/node/311.

9) Fitzmaurice, G. W., & Buxton, W. (1997). *An Empirical Evaluation of Graspable User Interfaces: towards specialized, space-multiplexed input*. CHI 1997, 43-50. Retrieved from ACM database.

10) *Aftertouch*. Retrieved on August 21, 2010, from Wiktionary: http://en.wiktionary.org/wiki/aftertouch.

11) *Apple MacBook Pro features*. Retrieved on August 23, 2010, from Apple website. http://www.apple.com/macbookpro/features.html.

12) Ishii, H., & Ullmer, B. (1997). *Tangible Bits: Towards Seamless Interfaces between People, Bits and Atoms*. CHI 1997, 234-241. Retrieved from ACM database.

13) *Sifteo*. Retrieved on August 23, 2010, from Sifteo website. http://www.sifteo.com.

14) Hornecker, E., & Buur, J. (2006). *Getting a Grip on Tangible Interaction: A Framework on Physical Space and Social*

Interaction. CHI 2006, 437-446. Retrieved from ACM database.

15) Tsukada, Y., & Hoshino, T. (2002). *Layered Touch Panel: The Input Device with Two Touch Panel Layers*. CHI 2002, 584-585. Retrieved from ACM database.

16) Stødle, D., & Anshus, O. J. (2008). *Blurring the line between real and digital: Pinning objects to wall-sized displays*. ACM 2008. Retrieved from ACM database.

17) Rekimoto, J., Ishizawa, T., Oba, H. (2003). *SmartPad: A Finger-Sensing Keypad for Mobile Interaction*. CHI 2003, 850-851. Retrieved from ACM database.

18) Rekimoto, J., Ishizawa, T., Schwesig, C., Oba, H. (2003). *PreSense: Interaction Techniques for Finger Sensing Input Devices*. ACM 2003, 203-212. Retrieved from ACM database.

19) Fallot-Burghardt, W., Fjeld, M., Speirs, C., Ziegenspeck, S., Krueger, H., Läubli, T. (2006). *Touch&Type: A Novel Pointing Device for Notebook Computers*. NordiCHI 2006, 465-468. Retrieved from ACM database.

20) Crabtree, A. (2004). *Design in the Absence of Practice: Breaching Experiments*. DIS 2004, 59-68. Retrieved from ACM database.

APPENDICES

A. SUI Cube electronics schematic

B. SUI Cube Instructions sheet

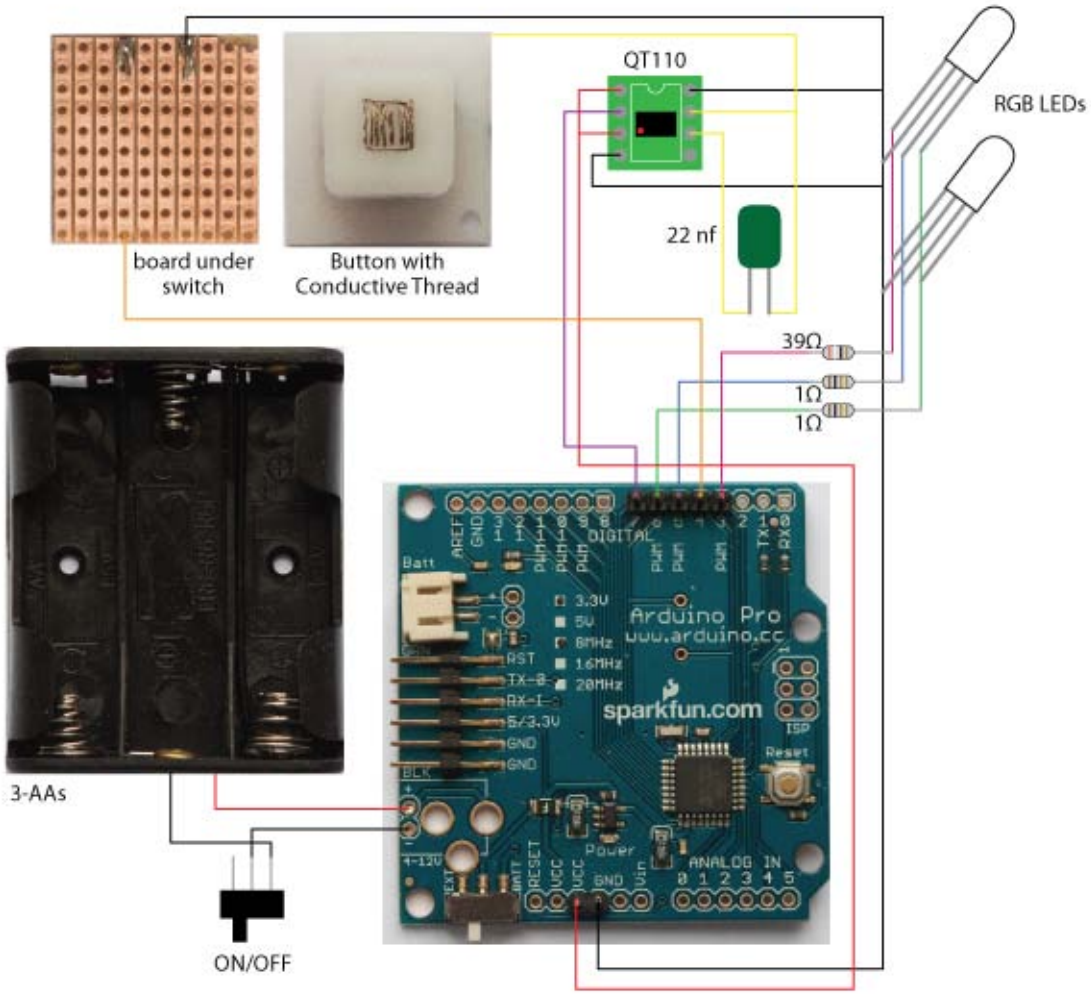
C. SUI Cube Code version #1

D. SUI Cube Code version #2

E. SUI Cube Code version #3

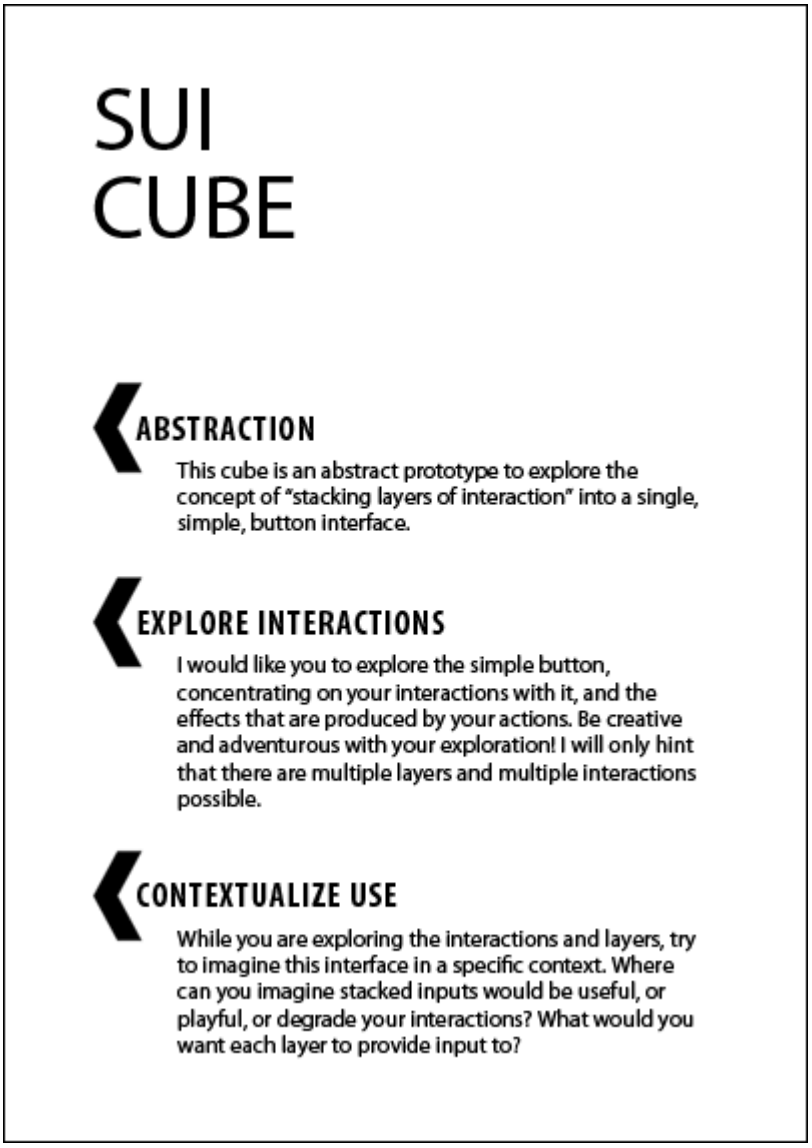
Appendix A

SUI Cube probe electronics component schematic.



Appendix B

SUI Cube instruction sheet included with each probe.



SUI
CUBE

ABSTRACTION

This cube is an abstract prototype to explore the concept of "stacking layers of interaction" into a single, simple, button interface.

EXPLORE INTERACTIONS

I would like you to explore the simple button, concentrating on your interactions with it, and the effects that are produced by your actions. Be creative and adventurous with your exploration! I will only hint that there are multiple layers and multiple interactions possible.

CONTEXTUALIZE USE

While you are exploring the interactions and layers, try to imagine this interface in a specific context. Where can you imagine stacked inputs would be useful, or playful, or degrade your interactions? What would you want each layer to provide input to?

Appendix C

SUI Cube code version #1 Arduino code.

```
/*
 SUI Cube #1
 build 2.2

 April 17, 2010
 Rob Nero

 Notes:   physical "press hold"

*/

#define BUTTON 4      //button switch
#define TOUCH 7       //touch switch

unsigned long tempClock = NULL;
unsigned long tempTime = NULL;

int tapCount = 0;                       //TOUCH variables
unsigned long touchUpClock = NULL;
unsigned long touchDownClock = NULL;
unsigned long touchUpTime = NULL;
unsigned long touchDownTime = NULL;

int pressCount = 0;                     //PRESS variables
boolean pressHold = false;                //used for constant press detection
boolean wasPressed = false;               //prevents a tap when pressed
unsigned long pressDownClock = NULL;
int lastHoldColor = 0;  //store hold color: 1-6
int totalHoldColors = 3; //total number of colors to cycle through

int redPin = 3;  //RGB LED PINS (w/PVM)
int grnPin = 6;  //
int bluPin = 5;  //

//CROSSFADE:
int red[3] = {100,0,0};          //colors
int resetRed[3] = {255,0,0};        //not used for crossfade function!
int blue[3] = {0,0,100};

int orange[3] = {100,65,0};
int yellow[3] = {100,100,0};
int green[3] = {0,100,0};
int purple[3] = {41,13,55};
int purple2[3] = {93,51,93};

int black[3]  = { 0, 0, 0 };
//Set initial color
int redVal = 255;
int grnVal = 0;
int bluVal = 0;
//Store previous color
int prevR = 255;  //set to pure red, color state when first starting PRESS
HOLD
int prevG = 0;
int prevB = 0;
//END

/*
================================================================================
*/
/* CUSTOMIZATIONS */
/*
================================================================================
*/
//CROSSFADE:
int wait = 0;      // 10ms internal crossFade delay; increase for slower fades

/*
================================================================================
*/
void setup(){
```

```
  Serial.begin(9600);  //debug

  pinMode(BUTTON,INPUT);        //set as input
  digitalWrite(BUTTON,HIGH);    //send current

  pinMode(redPin,OUTPUT);
  pinMode(grnPin,OUTPUT);
  pinMode(bluPin,OUTPUT);

  touchUpClock = millis();    //start TOUCH: up clock

  cubeOn();
}


/*
================================================================================
*/
void loop(){
  //PRESS HOLD
  if(pressHold){
    setPressHold();
  }

  //TOUCH
  if(isTouched()){
    if(touchDownClock == NULL){    //1st time down only
      touchFirstDown();
    }

    checkTouchHold();        //if HOLD...
  }

  //NO TOUCH
  else{
    if(touchUpClock == NULL){      //1st time up only
      touchFirstUp();
      if(!pressHold){
        resetCrossfadeVars();
      }
    }

    checkTouchTaps();        //if pause, repeat TAPS
  }


  //PRESS
  if(isPressed()){
    if(pressDownClock == NULL){    //1st time down only
      pressFirstDown();
    }

    pressButton(); //sets RED only

    checkPressHold();
  }

  //NO PRESS
  else{
    colorsOff();

    resetCrossfadeVars();
    lastHoldColor = 0;
    pressHold = false;

    if(pressDownClock != NULL){
      pressDownClock = NULL;
    }
  }
}

/* ------------------------------------------------------------------------ */
/* CUBE ON */
/* ------------------------------------------------------------------------ */
void cubeOn(){
  setColor(red);
```

```
  delay(100);
  setColor(blue);
  delay(100);
  colorsOff();
}

/* ---------------------------------------------------------------------- */
/* NO COLOR / LED OFF */
/* ---------------------------------------------------------------------- */
void colorsOff(){
  analogWrite(redPin,0);
  analogWrite(grnPin,0);
  analogWrite(bluPin,0);
}

/* ---------------------------------------------------------------------- */
/* SET SPECIFIC COLOR */
/* ---------------------------------------------------------------------- */
void setColor(int color[3]){
  int rVal = (color[0] * 255) / 100;
  int gVal = (color[1] * 255) / 100;
  int bVal = (color[2] * 255) / 100;

  analogWrite(redPin,rVal);
  analogWrite(grnPin,gVal);
  analogWrite(bluPin,bVal);
}
void setExactColor(int rCol, int gCol, int bCol){
  analogWrite(redPin,rCol);
  analogWrite(grnPin,gCol);
  analogWrite(bluPin,bCol);
}

/* ---------------------------------------------------------------------- */
/* CLEAR CROSSFADE VARIABLES */
/* so crossfades always start at red */
/* ---------------------------------------------------------------------- */
void resetCrossfadeVars(){
  setCrossfadeVars(resetRed);
}

void setCrossfadeVars(int color[3]){  //if red, use resetRed
  redVal = color[0];
  grnVal = color[1];
  bluVal = color[2];
  prevR = color[0];
  prevG = color[1];
  prevB = color[2];
}

/* ---------------------------------------------------------------------- */
/* IS BUTTON PRESSED */
/* ---------------------------------------------------------------------- */
boolean isPressed(){
  if(digitalRead(BUTTON) == LOW){
    return true;
  }
  else{
    return false;
  }
}

/* ---------------------------------------------------------------------- */
/* 1st PRESS DOWN */
/* ---------------------------------------------------------------------- */
void pressFirstDown(){
  pressDownClock = millis();
}

/* ---------------------------------------------------------------------- */
/* ACTIONS FOR BUTTON PRESS */
/* ---------------------------------------------------------------------- */
void pressButton(){
  if(!pressHold){
    setColor(red);
  }
```

```
  wasPressed = true;
}

/* ------------------------------------------------------------------------ */
/* SET PRESS HOLD COLOR */
/* ------------------------------------------------------------------------ */
void setPressHold(){
  if(lastHoldColor != 0){
    setExactColor(prevR, prevG, prevB);
  }
  else{
    setColor(red);
  }
}

/* ------------------------------------------------------------------------ */
/* CHECK: PRESS HOLD */
/* ------------------------------------------------------------------------ */
void checkPressHold(){
  tempClock = millis();
  tempTime = tempClock - pressDownClock;

  if(tempTime > 1000){
    pressHold = true;
    wasPressed = false;

    if(isTouched()){
      checkTouchHold();
    }
    else{
      fadePress();
    }
  }

  tempClock = NULL;
  tempTime = NULL;
}

/* ------------------------------------------------------------------------ */
/* FADE PRESS HOLD */
/* ------------------------------------------------------------------------ */
void fadePress(){
  while((isPressed()) && (!isTouched())){
    crossFade(black);
    crossFade(red);
  }
}

/* ------------------------------------------------------------------------ */
/* PRESS HOLD + TAP */
/* ------------------------------------------------------------------------ */
void setPressHoldTap(){
  lastHoldColor++;
  if(lastHoldColor == (totalHoldColors+1)){
    lastHoldColor = 1;
  }

  switch(lastHoldColor){
  case 1:
    setCrossfadeVars(green);
    break;
  case 2:
    setCrossfadeVars(blue);
    break;
  case 3:
    setCrossfadeVars(red);
    break;
  }
}

/* ------------------------------------------------------------------------ */
/* PRESS HOLD + TOUCH HOLD */
/* ------------------------------------------------------------------------ */
void fadePressHoldTouch(){
  while(isTouched()){
    if(pressHold){
```

```
      lastHoldColor++;
      switch(lastHoldColor){
        //1st color is default red
      case 1:
        crossFade(green);
        break;
      case 2:
        crossFade(blue);
        break;
      case 3:
        crossFade(red);
        break;
      }
    }
    else{
      checkTouchHold();
    }
  }
}

/* ---------------------------------------------------------------------- */
/* IS BUTTON TOUCHED */
/* ---------------------------------------------------------------------- */
boolean isTouched(){
  int val=digitalRead(TOUCH);
  if(val == LOW){
    return true;
  }
  else{
    return false;
  }
}

/* ---------------------------------------------------------------------- */
/* 1st TOUCH DOWN */
/* ---------------------------------------------------------------------- */
void touchFirstDown(){
  touchDownClock = millis();
  touchUpTime = touchDownClock - touchUpClock;
  touchUpClock = NULL;
}

/* ---------------------------------------------------------------------- */
/* 1st TOUCH UP */
/* ---------------------------------------------------------------------- */
void touchFirstUp(){
  touchUpClock = millis();
  touchDownTime = touchUpClock - touchDownClock;
  touchDownClock = NULL;

  if((touchUpTime > 100) && (touchDownTime < 250)){
    tapCount++;
  }
  touchUpTime = NULL;
  touchDownTime = NULL;
}

/* ---------------------------------------------------------------------- */
/* CHECK: TOUCH HOLD */
/* ---------------------------------------------------------------------- */
void checkTouchHold(){
  tempClock = millis();
  tempTime = tempClock - touchDownClock;
  if(tempTime > 2000){
    if(!pressHold){
      fadeTouch();  //touch only
    }
    else{
      fadePressHoldTouch();  //HOLD + HOLD
    }
  }
  tempClock = NULL;
  tempTime = NULL;
}

/* ---------------------------------------------------------------------- */
```

```
/* CHECK: TOUCH TAPS */
/* ------------------------------------------------------------------------ */
void checkTouchTaps(){
  tempClock = millis();
  tempTime = tempClock - touchUpClock;
  if((tapCount != 0) && (tempTime > 400)){
    if(!wasPressed){
      if(!pressHold){
        blinkTouch(tapCount);
      }
      else{
        setPressHoldTap();  //HOLD + TAP
      }
    }
    else{
      wasPressed = false;
    }
    tapCount = 0;
  }
  tempClock = NULL;
  tempTime = NULL;
}

/* ------------------------------------------------------------------------ */
/* REPEAT BUTTON TAPS/BLINKS */
/* ------------------------------------------------------------------------ */
void blinkTouch(int blinks){
  for(int i=1; i<=blinks; i++){
    setColor(blue);
    delay(200);
    colorsOff();
    delay(200);
  }
}

/* ------------------------------------------------------------------------ */
/* FADE UP/DOWN WHEN TOUCH HOLD */
/* ------------------------------------------------------------------------ */
void fadeTouch(){
  setCrossfadeVars(black);

  while(isTouched()){
    crossFade(blue);
    crossFade(black);
  }
}

/*
  Code for cross-fading 3 LEDs, red, green and blue (RGB)
  April 2007, Clay Shirky <clay.shirky@nyu.edu>
*/
int calculateStep(int prevValue, int endValue) {
  int step = endValue - prevValue; // What's the overall gap?
  if (step) {                      // If its non-zero,
    step = 1020/step;              //   divide by 1020
  }
  return step;
}

int calculateVal(int step, int val, int i) {
  if ((step) && i % step == 0) { // If step is non-zero and its time to change
a value,
    if (step > 0) {              //   increment the value if step is
positive...
      val += 1;
    }
    else if (step < 0) {         //   ...or decrement it if step is negative
      val -= 1;
    }
  }
  // Defensive driving: make sure val stays in the range 0-255
  if (val > 255) {
    val = 255;
  }
  else if (val < 0) {
    val = 0;
```

```
  }
  return val;
}

void crossFade(int color[3]) {
  Serial.print("crossfade:");
  Serial.print(redVal);
  Serial.print(" ");
  Serial.print(grnVal);
  Serial.print(" ");
  Serial.print(bluVal);
  Serial.println(" ");

  // Convert to 0-255
  int R = (color[0] * 255) / 100;
  int G = (color[1] * 255) / 100;
  int B = (color[2] * 255) / 100;

  int stepR = calculateStep(prevR, R);
  int stepG = calculateStep(prevG, G);
  int stepB = calculateStep(prevB, B);

  for (int i = 0; i <= 1020; i++) {
    redVal = calculateVal(stepR, redVal, i);
    grnVal = calculateVal(stepG, grnVal, i);
    bluVal = calculateVal(stepB, bluVal, i);

    //CUSTOM:
    prevR = redVal;  //CUSTOM: moved these lines here!
    prevG = grnVal;  //
    prevB = bluVal;  //

    if((isTouched()) || ((isPressed()) && (pressHold))){
      analogWrite(redPin, redVal);   // Write current values to LED pins
      analogWrite(grnPin, grnVal);
      analogWrite(bluPin, bluVal);
    }
    else{ //CUSTOM 'ELSE' FUNCTION!
      colorsOff();
      break;
    }
    if((isPressed()) && (!pressHold)){
      if(pressDownClock == NULL){    //1st time down only
        pressFirstDown();
      }
      pressButton();
      checkPressHold();
      break;
    }
    if((!isPressed()) && (pressHold)){
      pressDownClock = NULL;
      pressHold = false;
    }
    //END CUSTOM

    delay(wait); // Pause for 'wait' milliseconds before resuming the loop
  }

  // Update current values for next loop
  /*prevR = redVal;
  prevG = grnVal;
  prevB = bluVal;*/
}
```

Appendix D

SUI Cube code version #2 Arduino code.

```
/*
 SUI Cube #2
 build 2.3

 April 20, 2010
 Rob Nero

 Notes:   dbl-press (virtual) "press hold"

*/

#define BUTTON 4      //button switch
#define TOUCH 7       //touch switch

unsigned long tempClock = NULL;
unsigned long tempTime = NULL;

int tapCount = 0;                       //TOUCH variables
unsigned long touchUpClock = NULL;
unsigned long touchDownClock = NULL;
unsigned long touchUpTime = NULL;
unsigned long touchDownTime = NULL;

int pressCount = 0;                     //PRESS variables
boolean pressHold = false;                //used as 2-state on/off
boolean wasPressed = false;               //prevents a tap when pressed
unsigned long pressUpClock = NULL;
unsigned long pressDownClock = NULL;
unsigned long pressUpTime = NULL;
unsigned long pressDownTime = NULL;
int lastHoldColor = 0;  //store hold color: 1-6
int totalHoldColors = 3; //total number of colors to cycle through

int redPin = 3;  //RGB LED PINS (w/PVM)
int grnPin = 6;  //
int bluPin = 5;  //

//CROSSFADE:
int red[3] = {100,0,0};          //colors
int resetRed[3] = {255,0,0};        //not used for crossfade function!
int blue[3] = {0,0,100};

int orange[3] = {100,65,0};
int yellow[3] = {100,100,0};
int green[3] = {0,100,0};
int purple[3] = {41,13,55};
int purple2[3] = {93,51,93};

int black[3]  = { 0, 0, 0 };
//Set initial color
int redVal = 255;
int grnVal = 0;
int bluVal = 0;
//Store previous color
int prevR = 255;  //set to pure red, color state when first starting PRESS
HOLD
int prevG = 0;
int prevB = 0;
//END

/*
================================================================================
*/
/* CUSTOMIZATIONS */
/*
================================================================================
*/
//CROSSFADE:
int wait = 0;      // 10ms internal crossFade delay; increase for slower fades
```

```
/*
================================================================================
*/
void setup(){
  Serial.begin(9600);  //debug

  pinMode(BUTTON,INPUT);        //set as input
  digitalWrite(BUTTON,HIGH);    //send current

  pinMode(redPin,OUTPUT);
  pinMode(grnPin,OUTPUT);
  pinMode(bluPin,OUTPUT);

  touchUpClock = millis();    //start TOUCH: up clock
  pressUpClock = millis();    //start PRESS: up clock

  cubeOn();
}


/*
================================================================================
*/
void loop(){
  //PRESS HOLD
  if(pressHold){
    setPressHold();
  }

  //TOUCH
  if(isTouched()){
    if(touchDownClock == NULL){    //1st time down only
      touchFirstDown();
    }

    checkTouchHold();        //if HOLD...
  }

  //NO TOUCH
  else{
    if(touchUpClock == NULL){      //1st time up only
      touchFirstUp();
      if(!pressHold){
        resetCrossfadeVars();
      }
    }

    checkTouchTaps();        //if pause, repeat TAPS
  }


  //PRESS
  if(isPressed()){
    if(pressDownClock == NULL){    //1st time down only
      pressFirstDown();
    }

    pressButton(); //sets RED only

    checkPressHold();
  }

  //NO PRESS
  else{
    if(pressUpClock == NULL){      //1st time up only
      colorsOff();
      pressFirstUp();
    }

    checkPresses();
  }
}

/* ------------------------------------------------------------------------ */
/* CUBE ON */
/* ------------------------------------------------------------------------ */
```

```
void cubeOn(){
  setColor(red);
  delay(100);
  setColor(blue);
  delay(100);
  colorsOff();
}

/* ------------------------------------------------------------------------ */
/* NO COLOR / LED OFF */
/* ------------------------------------------------------------------------ */
void colorsOff(){
  analogWrite(redPin,0);
  analogWrite(grnPin,0);
  analogWrite(bluPin,0);
}

/* ------------------------------------------------------------------------ */
/* SET SPECIFIC COLOR */
/* ------------------------------------------------------------------------ */
void setColor(int color[3]){
  int rVal = (color[0] * 255) / 100;
  int gVal = (color[1] * 255) / 100;
  int bVal = (color[2] * 255) / 100;

  analogWrite(redPin,rVal);
  analogWrite(grnPin,gVal);
  analogWrite(bluPin,bVal);
}
void setExactColor(int rCol, int gCol, int bCol){
  analogWrite(redPin,rCol);
  analogWrite(grnPin,gCol);
  analogWrite(bluPin,bCol);
}

/* ------------------------------------------------------------------------ */
/* CLEAR CROSSFADE VARIABLES */
/* so crossfades always start at red */
/* ------------------------------------------------------------------------ */
void resetCrossfadeVars(){
  setCrossfadeVars(resetRed);
}

void setCrossfadeVars(int color[3]){
  redVal = color[0];
  grnVal = color[1];
  bluVal = color[2];
  prevR = color[0];
  prevG = color[1];
  prevB = color[2];
}

/* ------------------------------------------------------------------------ */
/* IS BUTTON PRESSED */
/* ------------------------------------------------------------------------ */
boolean isPressed(){
  if(digitalRead(BUTTON) == LOW){
    return true;
  }
  else{
    return false;
  }
}

/* ------------------------------------------------------------------------ */
/* 1st PRESS DOWN */
/* ------------------------------------------------------------------------ */
void pressFirstDown(){
  pressDownClock = millis();
  pressUpTime = pressDownClock - pressUpClock;
  pressUpClock = NULL;
}

/* ------------------------------------------------------------------------ */
/* 1st PRESS UP */
/* ------------------------------------------------------------------------ */
```

```
void pressFirstUp(){
  pressUpClock = millis();
  pressDownTime = pressUpClock - pressDownClock;
  pressDownClock = NULL;

  if((pressUpTime > 100) && (pressDownTime < 200)){
    pressCount++;
  }
  pressUpTime = NULL;
  pressDownTime = NULL;
}

/* ------------------------------------------------------------------------ */
/* CHECK: TOUCH PRESSES */
/* ------------------------------------------------------------------------ */
void checkPresses(){
  tempClock = millis();
  tempTime = tempClock - pressUpClock;
  if((pressCount != 0) && (tempTime > 400)){
    if((pressCount == 2) && (!pressHold)){
      pressHold = true;
      lastHoldColor++;
    }
    else if((pressCount == 2) && (pressHold)){
      colorsOff();
      resetCrossfadeVars();
      pressHold = false;
      lastHoldColor = 0;
    }
    pressCount = 0;
  }
  tempClock = NULL;
  tempTime = NULL;
}

/* ------------------------------------------------------------------------ */
/* ACTIONS FOR BUTTON PRESS */
/* ------------------------------------------------------------------------ */
void pressButton(){
  if(!pressHold){
    setColor(red);
  }
  wasPressed = true;
}

/* ------------------------------------------------------------------------ */
/* SET PRESS HOLD COLOR */
/* ------------------------------------------------------------------------ */
void setPressHold(){
  if(lastHoldColor != 0){
    setExactColor(prevR, prevG, prevB);
  }
  else{
    setColor(red);
  }
}

/* ------------------------------------------------------------------------ */
/* CHECK: PRESS HOLD */
/* ------------------------------------------------------------------------ */
void checkPressHold(){
  tempClock = millis();
  tempTime = tempClock - pressDownClock;

  if(tempTime > 2000){
    if((isPressed()) && (!isTouched())){
      fadePress();
    }
  }

  tempClock = NULL;
  tempTime = NULL;
}

/* ------------------------------------------------------------------------ */
/* FADE PRESS HOLD */
/* ------------------------------------------------------------------------ */
```

```
void fadePress(){
  setCrossfadeVars(resetRed);

  while((isPressed()) && (!isTouched())){
    crossFade(black);
    crossFade(red);
  }
}

/* ---------------------------------------------------------------------- */
/* PRESS HOLD + TAP */
/* ---------------------------------------------------------------------- */
void setPressHoldTap(){
  lastHoldColor++;
  if(lastHoldColor == (totalHoldColors+1)){
    lastHoldColor = 1;
  }

  switch(lastHoldColor){
  case 1:
    setCrossfadeVars(resetRed); //resetRed
    break;
  case 2:
    setCrossfadeVars(green);
    break;
  case 3:
    setCrossfadeVars(blue);
    break;
  }
}

/* ---------------------------------------------------------------------- */
/* PRESS HOLD + TOUCH HOLD */
/* ---------------------------------------------------------------------- */
void fadePressHoldTouch(){
  while(isTouched()){
    lastHoldColor++;
    switch(lastHoldColor){
      //1st color is default red
    case 1:
      crossFade(red); //regular red
      break;
    case 2:
      crossFade(green);
      break;
    case 3:
      crossFade(blue);
      break;
    }
  }
}

/* ---------------------------------------------------------------------- */
/* IS BUTTON TOUCHED */
/* ---------------------------------------------------------------------- */
boolean isTouched(){
  int val=digitalRead(TOUCH);
  if(val == LOW){
    return true;
  }
  else{
    return false;
  }
}

/* ---------------------------------------------------------------------- */
/* 1st TOUCH DOWN */
/* ---------------------------------------------------------------------- */
void touchFirstDown(){
  touchDownClock = millis();
  touchUpTime = touchDownClock - touchUpClock;
  touchUpClock = NULL;
}

/* ---------------------------------------------------------------------- */
/* 1st TOUCH UP */
```

```
/* ------------------------------------------------------------------------ */
void touchFirstUp(){
  touchUpClock = millis();
  touchDownTime = touchUpClock - touchDownClock;
  touchDownClock = NULL;

  if((touchUpTime > 100) && (touchDownTime < 250)){
    tapCount++;
  }
  touchUpTime = NULL;
  touchDownTime = NULL;
}

/* ------------------------------------------------------------------------ */
/* CHECK: TOUCH HOLD */
/* ------------------------------------------------------------------------ */
void checkTouchHold(){
  tempClock = millis();
  tempTime = tempClock - touchDownClock;
  if(tempTime > 2000){
    if(!pressHold){
      if(!isPressed()){
        fadeTouch();  //touch only
      }
    }
    else{
      fadePressHoldTouch();  //HOLD + HOLD
    }
  }
  tempClock = NULL;
  tempTime = NULL;
}

/* ------------------------------------------------------------------------ */
/* CHECK: TOUCH TAPS */
/* ------------------------------------------------------------------------ */
void checkTouchTaps(){
  tempClock = millis();
  tempTime = tempClock - touchUpClock;
  if((tapCount != 0) && (tempTime > 400)){
    if(!wasPressed){
      if(!pressHold){
        blinkTouch(tapCount);
      }
      else{
        setPressHoldTap();  //HOLD + TAP
      }
    }
    else{
      wasPressed = false;
    }
    tapCount = 0;
  }
  tempClock = NULL;
  tempTime = NULL;
}

/* ------------------------------------------------------------------------ */
/* REPEAT BUTTON TAPS/BLINKS */
/* ------------------------------------------------------------------------ */
void blinkTouch(int blinks){
  for(int i=1; i<=blinks; i++){
    setColor(blue);
    delay(200);
    colorsOff();
    delay(200);
  }
}

/* ------------------------------------------------------------------------ */
/* FADE UP/DOWN WHEN TOUCH HOLD */
/* ------------------------------------------------------------------------ */
void fadeTouch(){
  setCrossfadeVars(black);

  while(isTouched()){
    crossFade(blue);
```

```
    crossFade(black);
  }
}

/*
  Code for cross-fading 3 LEDs, red, green and blue (RGB)
  April 2007, Clay Shirky <clay.shirky@nyu.edu>
*/

int calculateStep(int prevValue, int endValue) {
  int step = endValue - prevValue; // What's the overall gap?
  if (step) {                      // If its non-zero,
    step = 1020/step;              //   divide by 1020
  }
  return step;
}

int calculateVal(int step, int val, int i) {
  if ((step) && i % step == 0) { // If step is non-zero and its time to change
a value,
    if (step > 0) {              //   increment the value if step is
positive...
      val += 1;
    }
    else if (step < 0) {         //   ...or decrement it if step is negative
      val -= 1;
    }
  }
  // Defensive driving: make sure val stays in the range 0-255
  if (val > 255) {
    val = 255;
  }
  else if (val < 0) {
    val = 0;
  }
  return val;
}

void crossFade(int color[3]) {
  Serial.print("crossfade:");
  Serial.print(redVal);
  Serial.print(" ");
  Serial.print(grnVal);
  Serial.print(" ");
  Serial.print(bluVal);
  Serial.println(" ");

  // Convert to 0-255
  int R = (color[0] * 255) / 100;
  int G = (color[1] * 255) / 100;
  int B = (color[2] * 255) / 100;

  int stepR = calculateStep(prevR, R);
  int stepG = calculateStep(prevG, G);
  int stepB = calculateStep(prevB, B);

  for (int i = 0; i <= 1020; i++) {
    redVal = calculateVal(stepR, redVal, i);
    grnVal = calculateVal(stepG, grnVal, i);
    bluVal = calculateVal(stepB, bluVal, i);

    //CUSTOM:
    prevR = redVal;  //CUSTOM: moved these lines here!
    prevG = grnVal;  //
    prevB = bluVal;  //

    if((isTouched()) || ((isPressed()) && (!isTouched()))){
      analogWrite(redPin, redVal);   // Write current values to LED pins
      analogWrite(grnPin, grnVal);
      analogWrite(bluPin, bluVal);
    }
    else{ //CUSTOM 'ELSE' FUNCTION!
      colorsOff();
      break;
    }
    //END CUSTOM
```

```
    delay(wait); // Pause for 'wait' milliseconds before resuming the loop
  }

  // Update current values for next loop
  /*prevR = redVal;
  prevG = grnVal;
  prevB = bluVal;*/
}
```

Appendix E

SUI Cube code version #3 Arduino code.

```
/*
 SUI Cube #3
 build 2.4

 May 2, 2010
 Rob Nero

 Notes:   each action is another color:
          tap=blue, press=red, taphold=green, presshold=purple,
          pressholdnotouch=red blink

*/

#define BUTTON 4      //button switch
#define TOUCH 7       //touch switch

unsigned long tempClock = NULL;
unsigned long tempTime = NULL;

int tapCount = 0;                       //TOUCH variables
unsigned long touchUpClock = NULL;
unsigned long touchDownClock = NULL;
unsigned long touchUpTime = NULL;
unsigned long touchDownTime = NULL;

int pressCount = 0;                     //PRESS variables
boolean pressHold = false;                //used for constant press detection
boolean wasPressed = false;               //prevents a tap when pressed
unsigned long pressDownClock = NULL;
int lastHoldColor = 0;  //store hold color: 1-6
int totalHoldColors = 3; //total number of colors to cycle through

int redPin = 3;  //RGB LED PINS (w/PVM)
int grnPin = 6;  //
int bluPin = 5;  //

//CROSSFADE:
int red[3] = {100,0,0};           //colors
int resetRed[3] = {255,0,0};        //not used for crossfade function!
int blue[3] = {0,0,100};

int orange[3] = {100,65,0};
int yellow[3] = {100,100,0};
int green[3] = {0,100,0};
int purple[3] = {41,13,55};
int purple2[3] = {93,51,93};

int white[3]  = { 100, 100, 100 };
int black[3]  = { 0, 0, 0 };
//Set initial color
int redVal = 255;
int grnVal = 0;
int bluVal = 0;
//Store previous color
int prevR = 255;  //set to pure red, color state when first starting PRESS
HOLD
int prevG = 0;
int prevB = 0;
//END

/*
================================================================================
*/
/* CUSTOMIZATIONS */
/*
================================================================================
*/
//CROSSFADE:
int wait = 0;      // 10ms internal crossFade delay; increase for slower fades
```

```
/*
================================================================================
*/
void setup(){
  Serial.begin(9600);  //debug

  pinMode(BUTTON,INPUT);        //set as input
  digitalWrite(BUTTON,HIGH);    //send current

  pinMode(redPin,OUTPUT);
  pinMode(grnPin,OUTPUT);
  pinMode(bluPin,OUTPUT);

  touchUpClock = millis();    //start TOUCH: up clock

  cubeOn();
}


/*
================================================================================
*/
void loop(){
  //PRESS HOLD
  if(pressHold){
    setPressHold();
  }

  //TOUCH
  if(isTouched()){
    if(touchDownClock == NULL){    //1st time down only
      touchFirstDown();
    }

    checkTouchHold();        //if HOLD...
  }

  //NO TOUCH
  else{
    if(touchUpClock == NULL){      //1st time up only
      touchFirstUp();
      if(!pressHold){
        resetCrossfadeVars();
      }
    }

    checkTouchTaps();        //if pause, repeat TAPS
  }


  //PRESS
  if(isPressed()){
    if(pressDownClock == NULL){    //1st time down only
      pressFirstDown();
    }

    pressButton(); //sets RED only
    checkPressHold();
  }

  //NO PRESS
  else{
    colorsOff();

    resetCrossfadeVars();
    lastHoldColor = 0;
    pressHold = false;

    if(pressDownClock != NULL){
      pressDownClock = NULL;
    }
  }
}

/* ------------------------------------------------------------------------ */
/* CUBE ON */
```

```
/* ---------------------------------------------------------------------- */
void cubeOn(){
  setColor(red);
  delay(100);
  setColor(blue);
  delay(100);
  colorsOff();
}

/* ---------------------------------------------------------------------- */
/* NO COLOR / LED OFF */
/* ---------------------------------------------------------------------- */
void colorsOff(){
  analogWrite(redPin,0);
  analogWrite(grnPin,0);
  analogWrite(bluPin,0);
}

/* ---------------------------------------------------------------------- */
/* SET SPECIFIC COLOR */
/* ---------------------------------------------------------------------- */
void setColor(int color[3]){
  int rVal = (color[0] * 255) / 100;
  int gVal = (color[1] * 255) / 100;
  int bVal = (color[2] * 255) / 100;

  analogWrite(redPin,rVal);
  analogWrite(grnPin,gVal);
  analogWrite(bluPin,bVal);
}
void setExactColor(int rCol, int gCol, int bCol){
  analogWrite(redPin,rCol);
  analogWrite(grnPin,gCol);
  analogWrite(bluPin,bCol);
}

/* ---------------------------------------------------------------------- */
/* CLEAR CROSSFADE VARIABLES */
/* so crossfades always start at red */
/* ---------------------------------------------------------------------- */
void resetCrossfadeVars(){
  setCrossfadeVars(resetRed);
}

void setCrossfadeVars(int color[3]){  //if red, use resetRed
  redVal = color[0];
  grnVal = color[1];
  bluVal = color[2];
  prevR = color[0];
  prevG = color[1];
  prevB = color[2];
}

/* ---------------------------------------------------------------------- */
/* IS BUTTON PRESSED */
/* ---------------------------------------------------------------------- */
boolean isPressed(){
  if(digitalRead(BUTTON) == LOW){
    return true;
  }
  else{
    return false;
  }
}

/* ---------------------------------------------------------------------- */
/* 1st PRESS DOWN */
/* ---------------------------------------------------------------------- */
void pressFirstDown(){
  pressDownClock = millis();
}

/* ---------------------------------------------------------------------- */
/* ACTIONS FOR BUTTON PRESS */
/* ---------------------------------------------------------------------- */
void pressButton(){
```

```
  if(!pressHold){
    setColor(red);
  }
  wasPressed = true;
}

/* ------------------------------------------------------------------------ */
/* SET PRESS HOLD COLOR */
/* ------------------------------------------------------------------------ */
void setPressHold(){
  if(lastHoldColor != 0){
    setExactColor(prevR, prevG, prevB);
  }
  else{
    setColor(red);
  }
}

/* ------------------------------------------------------------------------ */
/* CHECK: PRESS HOLD */
/* ------------------------------------------------------------------------ */
void checkPressHold(){
  tempClock = millis();
  tempTime = tempClock - pressDownClock;

  if(tempTime > 500){
    pressHold = true;
    wasPressed = false;

    if(isTouched()){
      checkTouchHold();
    }
    else{
      fadePress();
    }
  }

  tempClock = NULL;
  tempTime = NULL;
}

/* ------------------------------------------------------------------------ */
/* FADE PRESS HOLD, NO TOUCH */
/* ------------------------------------------------------------------------ */
void fadePress(){
  while((isPressed()) && (!isTouched())){
    setColor(red);
    delay(100);
    setColor(black);
    delay(100);
  }
}

/* ------------------------------------------------------------------------ */
/* PRESS HOLD + TAP */
/* ------------------------------------------------------------------------ */
void setPressHoldTap(){
  lastHoldColor++;
  if(lastHoldColor == (totalHoldColors+1)){
    lastHoldColor = 1;
  }

  switch(lastHoldColor){
  case 1:
    setCrossfadeVars(green);
    break;
  case 2:
    setCrossfadeVars(blue);
    break;
  case 3:
    setCrossfadeVars(red);
    break;
  }
}

/* ------------------------------------------------------------------------ */
```

```
/* PRESS HOLD + TOUCH HOLD */
/* ---------------------------------------------------------------------- */
void fadePressHoldTouch(){
  while(isTouched()){
    if(pressHold){
      setColor(purple);
    }
    else{
      checkTouchHold();
    }
  }
}

/* ---------------------------------------------------------------------- */
/* IS BUTTON TOUCHED */
/* ---------------------------------------------------------------------- */
boolean isTouched(){
  int val=digitalRead(TOUCH);
  if(val == LOW){
    return true;
  }
  else{
    return false;
  }
}

/* ---------------------------------------------------------------------- */
/* 1st TOUCH DOWN */
/* ---------------------------------------------------------------------- */
void touchFirstDown(){
  touchDownClock = millis();
  touchUpTime = touchDownClock - touchUpClock;
  touchUpClock = NULL;
}

/* ---------------------------------------------------------------------- */
/* 1st TOUCH UP */
/* ---------------------------------------------------------------------- */
void touchFirstUp(){
  touchUpClock = millis();
  touchDownTime = touchUpClock - touchDownClock;
  touchDownClock = NULL;

  if((touchUpTime > 100) && (touchDownTime < 250)){
    tapCount++;
  }
  touchUpTime = NULL;
  touchDownTime = NULL;
}

/* ---------------------------------------------------------------------- */
/* CHECK: TOUCH HOLD */
/* ---------------------------------------------------------------------- */
void checkTouchHold(){
  tempClock = millis();
  tempTime = tempClock - touchDownClock;
  if(tempTime > 500){
    if(!pressHold){
      fadeTouch();  //touch only
    }
    else{
      fadePressHoldTouch();  //HOLD + HOLD
    }
  }
  tempClock = NULL;
  tempTime = NULL;
}

/* ---------------------------------------------------------------------- */
/* CHECK: TOUCH TAPS */
/* ---------------------------------------------------------------------- */
void checkTouchTaps(){
  tempClock = millis();
  tempTime = tempClock - touchUpClock;
  if((tapCount != 0) && (tempTime > 400)){
    if(!wasPressed){
```

```
      if(!pressHold){
        blinkTouch(tapCount);
      }
      else{
        setPressHoldTap();  //HOLD + TAP
      }
    }
    else{
      wasPressed = false;
    }
    tapCount = 0;
  }
  tempClock = NULL;
  tempTime = NULL;
}

/* ------------------------------------------------------------------------ */
/* REPEAT BUTTON TAPS/BLINKS */
/* ------------------------------------------------------------------------ */
void blinkTouch(int blinks){
  for(int i=1; i<=blinks; i++){
    setColor(blue);
    delay(200);
    colorsOff();
    delay(200);
  }
}

/* ------------------------------------------------------------------------ */
/* FADE UP/DOWN WHEN TOUCH HOLD */
/* ------------------------------------------------------------------------ */
void fadeTouch(){
  while(isTouched() && (!isPressed())){
    setColor(green);
  }
}

/*
  Code for cross-fading 3 LEDs, red, green and blue (RGB)
  April 2007, Clay Shirky <clay.shirky@nyu.edu>
*/
int calculateStep(int prevValue, int endValue) {
  int step = endValue - prevValue; // What's the overall gap?
  if (step) {                      // If its non-zero,
    step = 1020/step;              //   divide by 1020
  }
  return step;
}

int calculateVal(int step, int val, int i) {
  if ((step) && i % step == 0) { // If step is non-zero and its time to change
a value,
    if (step > 0) {              //   increment the value if step is
positive...
      val += 1;
    }
    else if (step < 0) {         //   ...or decrement it if step is negative
      val -= 1;
    }
  }
  // Defensive driving: make sure val stays in the range 0-255
  if (val > 255) {
    val = 255;
  }
  else if (val < 0) {
    val = 0;
  }
  return val;
}

void crossFade(int color[3]) {
  Serial.print("crossfade:");
  Serial.print(redVal);
  Serial.print(" ");
  Serial.print(grnVal);
  Serial.print(" ");
```

```
  Serial.print(bluVal);
  Serial.println(" ");

  // Convert to 0-255
  int R = (color[0] * 255) / 100;
  int G = (color[1] * 255) / 100;
  int B = (color[2] * 255) / 100;

  int stepR = calculateStep(prevR, R);
  int stepG = calculateStep(prevG, G);
  int stepB = calculateStep(prevB, B);

  for (int i = 0; i <= 1020; i++) {
    redVal = calculateVal(stepR, redVal, i);
    grnVal = calculateVal(stepG, grnVal, i);
    bluVal = calculateVal(stepB, bluVal, i);

    //CUSTOM FOR SUI CUBE:
    prevR = redVal;  //CUSTOM: moved these lines here!
    prevG = grnVal;  //
    prevB = bluVal;  //

    if((isTouched()) || ((isPressed()) && (pressHold))){
      analogWrite(redPin, redVal);   // Write current values to LED pins
      analogWrite(grnPin, grnVal);
      analogWrite(bluPin, bluVal);
    }
    else{ //CUSTOM 'ELSE' FUNCTION!
      colorsOff();
      break;
    }
    if((isPressed()) && (!pressHold)){
      if(pressDownClock == NULL){    //1st time down only
        pressFirstDown();
      }
      pressButton();
      checkPressHold();
      break;
    }
    if((!isPressed()) && (pressHold)){
      pressDownClock = NULL;
      pressHold = false;
    }
    //END CUSTOM

    delay(wait); // Pause for 'wait' milliseconds before resuming the loop
  }

  // Update current values for next loop
  /*prevR = redVal;
  prevG = grnVal;
  prevB = bluVal;*/
}
```