DESIGNING INTERACTIONS THAT COMBINE PEN, PAPER, AND COMPUTER

SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE

AND THE COMMITTEE ON GRADUATE STUDIES

OF STANFORD UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

Ron Bing Yeh

December 2007

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Scott R. Klemmer – Principal Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Terry A. Winograd

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Andreas Paepcke

Approved for the University Committee on Graduate Studies.

Abstract

Pen and paper are powerful tools for visualizing designs, penning music, and communicating through the written language. This coupling is mobile, flexible, graspable, and robust. It has even evolved in response to technology: we print out electronic articles to read, and scribble annotations on them before meeting with colleagues. The introduction of digital pens that capture handwriting has now made it feasible to augment forms, notepads, and maps with computation. Applications can recognize handwriting, upload notes to the Web, and detect pen gestures for initiating a search. In doing so, we combine paper's affordances with the benefits of technology, including search, redundancy, and remote collaboration. We refer to these as paper + digital interfaces.

Developing paper + digital interfaces is challenging. Programmers need to abstract input into high-level events, coordinate interactions across time, and manage output on devices. This is difficult, because interface programmers are accustomed to working with graphical applications that provide real-time feedback on a single display. Additionally, debugging paper interfaces requires added effort, such as printing an interface before testing. As a result, few people currently build paper + digital interfaces.

This dissertation explores how pen and paper can be used in concert with computers to make tasks more efficient, engaging, and robust. It contributes new software tools and

paper + digital interaction techniques. It comprises PaperToolkit, a platform for building paper applications; ButterflyNet, a paper notebook that automatically structures field data; and GIGAprints, large paper prints that support collaboration and visual search.

PaperToolkit helps programmers build applications with digital pens and paper, introducing abstractions and tools to enhance development and testing of these interfaces. We evaluated the toolkit through a class deployment with 17 teams (69 students), an analysis of the code produced by those teams, and extended use in our own research projects. The evaluation found the abstractions to be highly effective. The approach provides a learnable programming model, and also lowers the barrier to debugging multi-device interactions.

PaperToolkit's design was influenced by our experience building paper interfaces. Field scientists, such as biologists, rely on paper notebooks to capture and structure field data. This practice inspired ButterflyNet. With ButterflyNet, a scientist captures handwritten notes and photographs using a digital pen, notebook, and camera. When this content is uploaded to a computer, it is automatically organized and presented in a multimedia browser. A lab study with 14 biologists shows that ButterflyNet enhances the field capture of notes and digital media.

Beyond the paper notebook, we explored the class of large printed interfaces that augment walls and tables. GIGAprints comprises augmented posters and maps that present a large amount of high resolution graphical content, yet still can be rolled up and taken into the field. Graphical feedback can be displayed on a handheld device, or overlaid on the GIGAprints. Our evaluation found that the large size provides added visual context, and enhances both collocated and remote collaboration.

Using PaperToolkit, applications such as ButterflyNet and GIGAprints could have been built in much less time. The toolkit is open source, and is used today in research laboratories around the world, including labs in Paris, Siegen, Darmstadt, and at Stanford.

Acknowledgments

I would like to thank my adviser, Scott Klemmer, for his guidance on my dissertation research. His unwavering trust in his students' abilities has constantly motivated me to improve my research, my presentations, and my writing. I am grateful to Andreas Paepcke for introducing me to the world of field biology, and for the delightful conversations we have shared, research and otherwise. Andreas' care for the details, and his ability to connect them to the larger research picture, is inspiring. Before Scott arrived at Stanford, I was Terry Winograd's student. Terry introduced me to Human-Computer Interaction, and taught me to always dig deeper to uncover the core of the research. Scott, Terry, and Andreas have all provided me great encouragement, and have also imparted their insight into the research.

I am grateful to Maneesh Agrawala and Roy Pea for serving on my orals committee. Maneesh has been one of the main supporters of my work, as he enthusiastically used my toolkit as a platform and teaching tool in his class. Thanks also to the other researchers who have encouraged my research and asked me to present and share it with their groups: François Guimbretière, Eric Brewer, Genevieve Bell, and many others.

This work could not have been completed without the support of scientists in the field biology community. Rodolfo Dirzo acted as a springboard for my early work, and Jeannie Stamberger was my chief liaison to the Biology department. I have also had great

experiences learning from and interacting with Cindy Wilber, Nona Chiariello, Bill Gomez, Gary Nielsen, Philippe Cohen, Jessica Shors, and many others at Jasper Ridge.

I have learned much from my collaborators. In particular, I thank Joel Brandt, Brian Lee, Chunyuan Liao, Mor Naaman, and Doantam Phan. I have also enjoyed dispensing advice to the younger students, who have in return helped me hone my mentorship skills. To Michael Bernstein, Jonas Boli, Boyko Kakaradov, Avi-Robinson Mosher, Nikhil Raghavan, Hao Jiang, Jürgen Steimle, Jim Hefner, Marcello Bastéa-Forte, Thaala Monsti, and Eric Su: hopefully you learned as much from me as I did from you. I also enjoyed sharing food and conversations with the members of Stanford's HCI Group; also, special thanks to my office mates, who have made space for my equipment.

The National Science Foundation has supported my research over the last several years. Aside from the funding, another stress reducer has been the amazing support staff in the graphics laboratory. John Gerth, Heather Gentner, and Ada Glucksman are amazing at their jobs. Please give these folks raises.

No quest for a Ph.D. is completed without a support cast of graduate school friends, who share in the joys of passing qualifying exams (and the pains of getting papers rejected). Much of this experience was shared over great Bay Area food, late night chats, awesome video games, TV, and collaborative surfing of Wikipedia. My graduate school cohorts include Doantam Phan, Dan Morris, Merrie Ringel Morris, Neel Joshi, Augusto Román, Jim Chow, Jeff Klingner, Kayvon Fatahalian, Joel Sandin, Leith Abdulla, and many many others. If I forgot to list you here, please IM me and I will treat you out to a hot bowl of Phở.

Finally, I thank my loved ones for the support they have given me, despite the fact that I only ever told them that I do research on *computers* and that I would graduate *someday*.

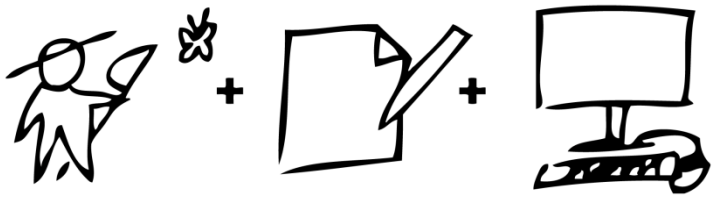
To *Sherry, Frank, Rick,* and *Sue*: thanks a billion.

Contents

Figures

Tables

1



Introduction

This dissertation introduces an approach to creating software that combines pen, paper, and computer. In these applications, people author content on paper and issue commands to a computer through paper interface components and pen gestures; the computer provides graphics and sound as feedback. This chapter demonstrates the need for such applications, describes challenges encountered when building them, and provides an overview of the solution. This work's research insights are embodied in PaperToolkit, a software architecture, library, and suite of tools for authoring applications in this genre. Novel interactions are also demonstrated through two paper + digital computer systems, ButterflyNet and GIGAprints. The design of each system is inspired by observations of a potential user community. They are evaluated with users in laboratory or long-term studies.

1.1 A Need for Augmented Paper Interactions

Though computers are integrated into our work and play, the paper tools of centuries past are not yet obsolete. In *The Myth of the Paperless Office*, Sellen and Harper show that the affordances of paper—grasping, carrying, folding—have encouraged its heavy use, even in today's digital world [Sellen and Harper 2001]. Architects, artists, and designers rely on paper because it is easier to sketch in a notebook than draw on a computer. Scientists use laboratory notebooks as a permanent record of procedures and observations. Office workers handle large volumes of paper every day. Printed forms provide a physical representation for their daily tasks, such as filing financial information for an employee.

Pen-and-paper are powerful tools for visualizing designs, penning music, and communicating through art and the written language. They provide clear advantages for these domains. Users do not have to wait for a notepad to boot up, and do not need to apply security patches. Paper's use has adapted to technology: people print email attachments to read and mark up. Yet, despite paper's reliability, mobility, flexibility, and readability, it cannot compete with computers on aspects such as search, storage, and sharing. To share annotations with a remote collaborator, we scan or photograph pages, and then email them.

By adding computation, users can receive the best of both worlds. This approach was first explored by Pierre Wellner in DigitalDesk [Wellner 1993], which sought to bring paper into the world of computing. DigitalDesk provided interactions to blend the physical and digital desktops. For example, when a user points to a number written on a sheet of paper, the computer enters that value into a digital calculator.

There are many technologies that can augment paper, including scanners, cameras, and digitizing tablets (see SECTION 2.1 for details). The introduction of digital pens that capture a person's handwriting on paper (*e.g.*, [Anoto AB 2007, EPOS 2007]) has now made it feasible to augment forms, notepads, and maps with computation. This dissertation uses the Anoto pen for its functionality, battery life, and robustness. The pen captures handwritten input when used on regular paper overlaid with a location-specifying dot pattern. As the pen lays ink on the paper, it timestamps and saves each stroke into memory. The user plugs the

pen into a computer to upload the digitized notes. The pen can also stream coordinates to software in real-time, over a Bluetooth wireless connection.

Digital pen-and-paper offers tradeoffs relative to other pen-based computing platforms. Like PDAs and digitizing tablets, digital pens take advantage of our ability to sketch and handwrite. The pen's primary advantage is that it lays down actual ink, so if the capture mechanism fails, the user has a paper backup. The main disadvantage (*e.g.*, compared with Tablet PCs) is that the writing surface does not provide real-time graphical feedback.

1.2 Challenges in Creating Paper+Digital Interactions

With digital pen and paper, applications tend to include off-the-desktop interactions, outside the confines of a graphical user interface (GUI). Users interact not with a keyboard and mouse, but through a device ensemble [Schilit and Sengupta 2004] which includes the digital pen, paper notebook, and desktop or mobile computers that back up, process, or share the handwritten content. For example, a user might write on a paper pad. This note is recognized, and immediately uploaded onto a web calendar (sent through a mobile phone in the user's pocket). These *ensemble interactions* comprise user actions that may take place across multiple interfaces and devices (see Figure 1.1).

Creating these interfaces is time consuming (*i.e.*, many lines of code to deal with pen hardware, digital ink, and devices) and requires much developer expertise (*i.e.*, keeping a mental model of this unfamiliar genre of application is difficult). Current applications are

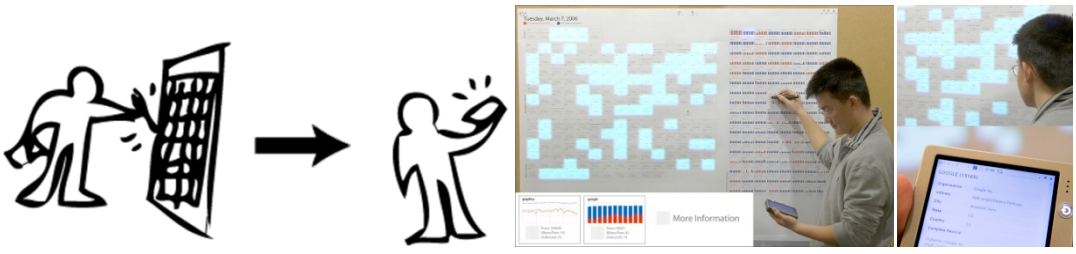


FIGURE 1.1. Software that integrates the use of paper documents with digital artifacts are referred to as *paper + digital applications*. Specifically, this work concentrates on scenarios where a user writes with a pen, and receives feedback on a handheld or desktop computer. Digital pens capture the input, and send this information to PaperToolkit, our platform for coordinating user input and application output.

not aware of digital pens. Programmers must work with the pen hardware and abstract raw input into high-level events. Program code must coordinate interactions across time and space, and send output to devices. This is difficult, because interface programmers usually work with applications that provide instant feedback on a single display.

Beyond the engineering details of processing input and output, the programmer must also design interactions that do not burden the user, but instead build on existing practices. The programmer must also cope with the difficulties in debugging such applications, since the interactions differ from standard GUI interactions. For example, some paper applications are designed to process input in batch, only after a user plugs his pen into the upload cradle. It is not clear what the best techniques are for testing such an application. Additionally, many of these applications would benefit from recognition of pen gestures, which may be difficult to specify and test. As a result, few programmers build paper + digital applications.

1.3 Overview of the Solution

This work addresses these challenges by applying the event-based approach, common in GUIs, to paper interfaces (see Figure 1.2). The work extends this model to address paper's

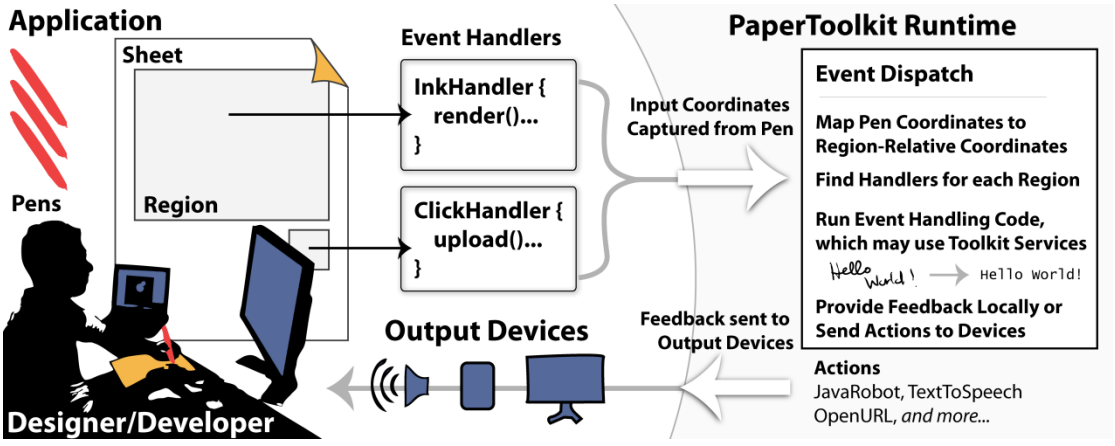


FIGURE 1.2. PaperToolkit helps programmers build paper applications. The main architecture is inspired by existing GUI architectures, where interface events are mapped to components and dispatched to event handlers. Feedback is displayed on output devices, since paper cannot provide graphical updates.

distinguishing characteristics, such as the inability to provide real-time graphical updates. We introduce techniques to unify real-time and batched input handling, coordinate interactions across devices, and debug paper applications.

1.3.1 Thesis Statement

This dissertation demonstrates a learnable and effective method for creating paper + digital applications. To do this, it:

- starts from the traditional GUI model of dispatching input events to interface components and event handlers.
- provides techniques to account for the differences between paper and graphical interfaces, such as the lack of real-time graphical feedback.
- augments the programmer's workflow with development and debugging techniques that take advantage of the unique aspects of working with pen and paper.

PaperToolkit was designed to help programmers build applications with today's digital pens and paper (see Figure 1.2). It introduces abstractions and tools to enhance development and testing of these interfaces. The approach, based on traditional GUI architectures, provides a learnable and efficient programming model, and lowers the barrier to debugging multi-device interactions. While PaperToolkit uses the Anoto hardware, its ideas can apply to alternate pen technologies (reviewed in CHAPTER 2).

1.3.2 Approach

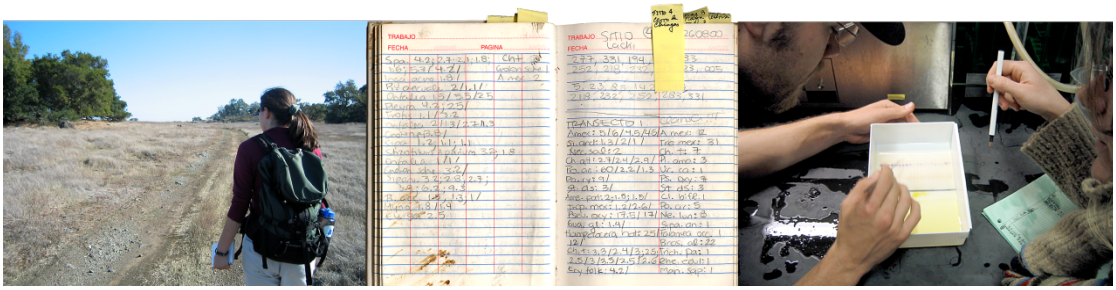


FIGURE 1.3. Our observations of biologists at work in the field and lab inspired augmented paper interfaces for mobility and collaboration. The requirements distilled by this field study factored into the design of PaperToolkit.

Three threads run throughout this work. First, the research efforts are all inspired by *observation* of a user community, whether they are biologists or programmers. We determine potential needs of the users before writing any code. Second, the *design* of the interface and system reflects the opportunities identified in the field observations. Finally, we *validate* the designs through multiple methods, including laboratory studies, heuristic measures, long-term deployments of the system, and source-code analysis.

To begin the work, the author spent 400 hours observing and working alongside field biologists (see Figure 1.3). This fieldwork inspired a system called ButterflyNet (see Figure 1.4, left). ButterflyNet provides a set of interactions that support the workflow of these mobile professionals, who rely on paper notebooks to capture and structure field data. With ButterflyNet, a biologist captures handwritten notes and photographs using a digital pen, notebook, and camera. When this content is uploaded to a computer, ButterflyNet

FIGURE 1.4. The challenges in building ButterflyNet and GIGAprints inspired our research into toolkits and development environments for paper + digital applications.

automatically organizes and presents it in a multimedia browser. Laboratory studies and field tests showed that ButterflyNet enhances the practices of field scientists.

Beyond the paper notebook, this work explored a class of large printed interfaces to augment walls and tables. GIGAprints comprises augmented posters and maps that present a large amount of high resolution graphical content (see Figure 1.4, right). Depending on the use case, graphical feedback is displayed on a handheld device, or overlaid on top of the GIGAprints. The prints do not completely sacrifice mobility, as they can be rolled up and taken into the field (*e.g.*, big maps). A lab evaluation found that the large size of GIGAprints provides added visual context, and enhances both collocated and remote collaboration.

Finally, we distilled the lessons learned from building applications into the basic abstractions for a paper + digital toolkit. We evaluated PaperToolkit through extended use in research projects, a class deployment with 69 students (17 teams), and an analysis of the source code produced by those teams. The evaluation found the abstractions to be learnable and efficient. Further iterations resulted in novel techniques for designing and debugging. In total, PaperToolkit presents an effective solution for creating paper + digital applications.

Thus, the toolkit design has taken into account *both* the software developer *and* end users. Using PaperToolkit, applications such as ButterflyNet and GIGAprints could have been built in less time. The toolkit is open source (http://papertoolkit.googlecode.com) and is used today in research laboratories around the world: In|Situ| in Paris, University of Siegen, Darmstadt University of Technology, University of California–Riverside, and in labs at Stanford University.

This work serves several goals: in one respect, it is a pragmatic solution to the gap in affordances between paper and computer. Trends indicate that future interfaces will combine the benefits of tablet computers, digital paper, and electronic ink [E Ink 2007]. But until we reach that future, programmers must cope with today's hardware. However, the tools also provide an epistemic benefit: programmers can design and experience the interactions of that future world. Finally, the research insights detailed in this work contribute fundamentally to the design of architectures for paper and digital ensembles.

1.3.3 Outline of Contributions

This dissertation contributes software tools and paper + digital interaction techniques to make tasks more efficient, engaging, and robust. It offers contributions in three areas:

First, it presents toolkit support for augmented pen-and-paper interactions. PaperToolkit introduces a software architecture that:

- Lowers the barrier for developing mobile and collaborative applications that acquire input from digital pens, and present output through devices.
- Makes development more efficient through techniques to speed up the search for API calls, and reduce the amount of interface code the programmer needs to write.
- Enhances debugging through a monitoring technique and facilities to review and replay test sessions.

Second, it introduces interaction techniques to support the mobility and collaboration of field workers. The challenges we met while creating these applications inspired our research into toolkit support for pen-and-paper interactions.

- ButterflyNet provides a smart notebook interaction and a desktop interface to help field scientists capture, access, and transform data.
- Interactive Gigapixel Prints (GIGAprints) introduces interaction around large paper interfaces, to support the collaboration needs of scientists and designers.

Third, we extend work in user-centered methods for designing and evaluating toolkits, by:

- Observing developers at work to learn about existing practices, and discover opportunities for design.

FIGURE 1.5. Students created tools for many tasks, including (from left to right): paper blogs, location-based search, web design, and music composition. Going beyond the retrieval and form-filling tasks in paper + digital prior work, these apps explore real-time control of screen elements and recognition of informal input. The student projects are described in CHAPTER 7.

- Deploying PaperToolkit in a long-term study with 17 teams (69 students) to evaluate its learning threshold (see Figure 1.5).
- Using source-code analysis to discover usage patterns for the toolkit. These patterns informed the iterative design process.

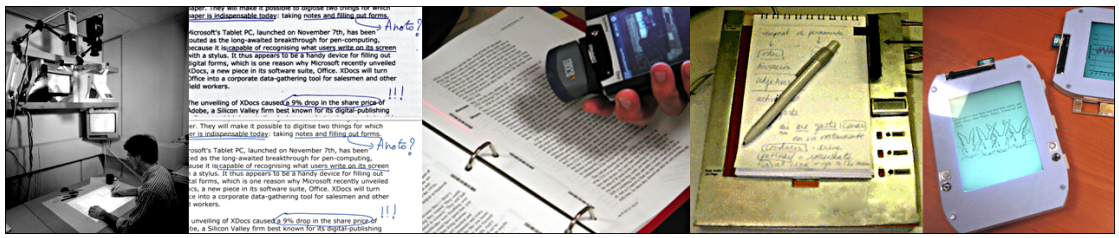
1.3.4 Roadmap

This dissertation is targeted toward interaction designers, researchers, and toolkit architects. Interaction designers will be interested in the range of paper + digital interaction techniques described in SECTION 2.1 and CHAPTERS 4 and 5. Designers can also benefit from the field observations of biologists and the user-centered design of ButterflyNet and GIGAprints, laid out in CHAPTERS 3-5. Toolkit researchers and developers will find value in SECTIONS 2.2-2.4, and CHAPTERS 6-8.

- CHAPTER 2 reviews previous work in the area of augmented paper, including related paper + digital systems and interactions. We describe existing platforms for creating these interfaces, and show how the approaches inspired our work.
- CHAPTER 3 presents an observational study of field biology researchers, to motivate the work in the rest of the dissertation. The user requirements we distilled, including support for mobility and collaboration, motivated ButterflyNet and GIGAprints.

- Chapter 4 presents ButterflyNet, a smart notebook interface that supports the work of field biologists. The chapter presents the design process and evaluation, and provides architecture and implementation details.
- Chapter 5 presents gigaprints, large printed interfaces that support collaboration. It describes the range of interactions we explored in this space, and the resulting implications for toolkit design.
- Chapter 6 describes the architecture and abstractions of PaperToolkit, the core part of this dissertation. The approach builds on existing graphical architectures to make the transition to building paper uis approachable for interface programmers.
- Chapter 7 evaluates the core approach through multiple methods, including a longitudinal deployment of the toolkit. We examine the projects created by developers, and through source code analysis, show that PaperToolkit reduces the time and expertise required to make paper + digital applications.
- Chapter 8 details the design response to the evaluation in chapter 7. It describes the novel interactions for development and debugging. These interactions support the programmer's workflow by making increasing efficiency and reducing errors.
- Chapter 9 summarizes the lessons learned and points to future directions, including ideas for hybrid interfaces, where paper and digital surfaces are intermixed.

2



Related Work

The PaperToolkit research builds on work in augmented paper, interface platforms, toolkit evaluation methods, and taxonomies for interactions. Though the core research is on *tools* for creating paper + digital interfaces, we begin by examining augmented paper systems (*i.e.*, the *applications* PaperToolkit might support). SECTION 2.1 covers a range of research and commercial systems that augment computers with paper's affordances. To do this, systems have taken various approaches. Most applications try to improve how people VIEW digital documents. Others enhance CAPTURE—the practice of writing handwritten notes on paper. A third approach provides ways to LINK physical and digital artifacts. The final approach explores the use of augmented paper as a way to COMMAND computer programs. Individual systems can draw elements from one or more approaches.

To organize these applications, SECTION 2.2 offers a design space for paper + digital applications (developed as a result of the ButterflyNet, GIGAprints, and PaperToolkit projects). This taxonomy helps us reason about paper + digital applications, and is used throughout this dissertation (*e.g.*, in CHAPTER 6, it helps outline toolkit requirements).

The second half of the chapter details research related to the toolkit aspects of the work. SECTION 2.3 explores related interface platforms, for paper applications and for the larger space of physical-digital computing. It also overviews multiple methods to evaluate toolkits. SECTION 2.4 discusses how visual design tools can help lower the threshold for using a toolkit. Since tools are *user interfaces* for programmers, they should be designed around programmers' current practices; related studies on today's development and debugging techniques are also presented in this section. The chapter concludes with a summary of the main contributions this research offers beyond the existing work.

2.1 Integrating the Physical and Digital Worlds

The problem of combining the affordances of paper and computer can be tackled from one of two directions. One way is to *augment paper documents* by introducing computing features (left side of Figure 2.1). A sharing solution that takes digital photos of paper notebooks would fall on this side of the spectrum. From the other direction, we can *redesign computers* to include paper's benefits. Today's smart phones provide the mobility of paper notebooks; Tablet PCs provide pen-based interactions.

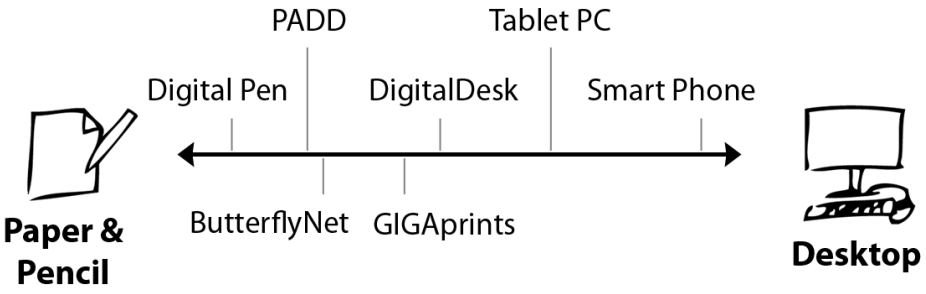


FIGURE 2.1. The problem can be tackled from either direction. Some projects take printed documents and introduce computation (*e.g.*, by digitizing handwriting). Other projects start with computers and introduce mobility, robustness, and pen-based input (*e.g.*, Tablet PC). Our work, ButterflyNet and GIGAprints, augments paper with digital pens and devices.

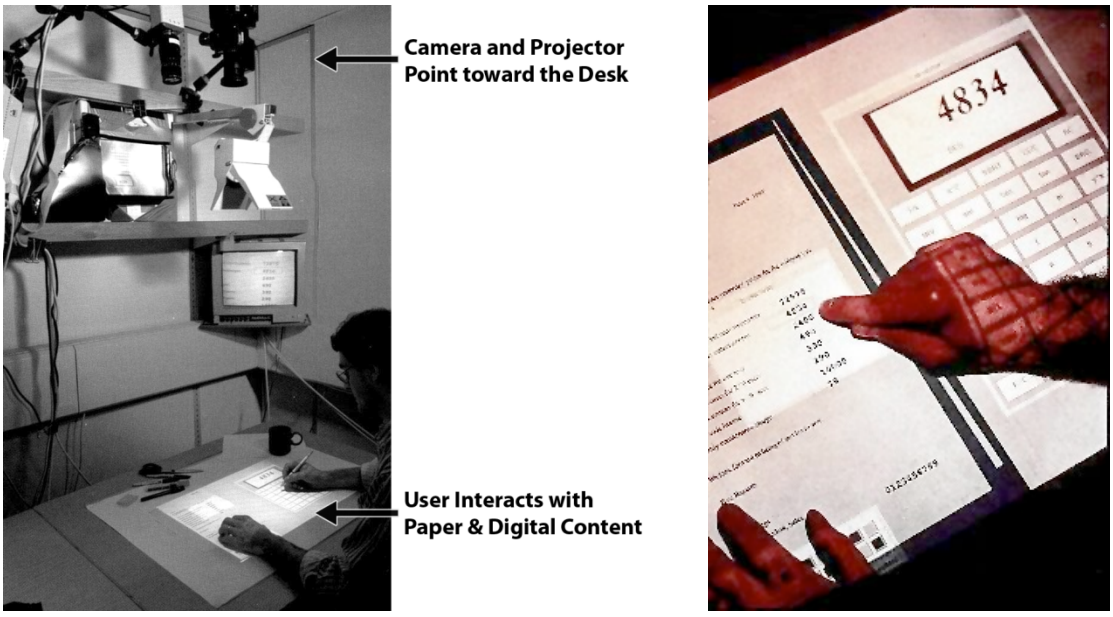


FIGURE 2.2. Pierre Wellner's DigitalDesk [Wellner 1993] allows users to work with digital and physical documents in tandem. *Left*) it employs a top-mounted camera to track user input and a projector to display digital content. *Right*) a user can point at a number on a printed document to send it to a digital calculator, which is projected on the desk.

Existing systems fall along the spectrum in Figure 2.1, and offer different techniques to coordinate artifacts in the paper and digital worlds. An early research example is the DigitalDesk, a seminal work by Wellner and colleagues [Wellner 1993]. DigitalDesk comprises a projector and a camera, both mounted in the ceiling and pointed down at a physical desk (see Figure 2.2, left). The camera handles input by tracking the movement of the user's hands. The projector provides output by overlaying digital graphics onto sheets of paper. Together, they enable the user to coordinate interactions on paper with computer programs. For example, pointing to a number on a printed document will send it to a graphical calculator (see Figure 2.2, right). This work inspired much of the paper + digital research in the 1990s. When building GIGAprints (see CHAPTER 5), we took DigitalDesk as inspiration for experimenting with overlaid projector output. GIGAprints provides finer-grained control of the input, but gives up some flexibility in that it requires the use of a pen (DigitalDesk's vision algorithm allowed the use of fingers to designate items of interest).

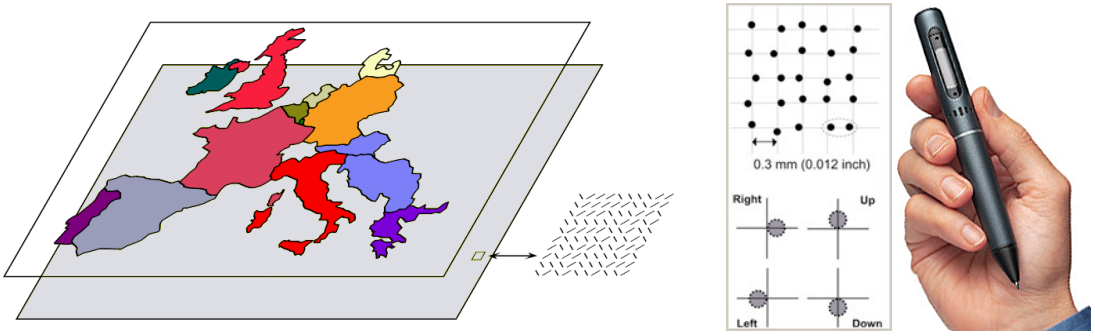


FIGURE 2.3. *Left*) In 1998, Dymetman and Copperman described the Intelligent Paper approach, where a pen tracks a 2D barcode (Xerox DataGlyph) barely visible to the human eye. *Right*) The Anoto pen uses a related approach—jittered dots encode the location. LiveScribe's smartpen is a product using this technology to associate audio with handwritten notes.

Building on ideas from Xerox research [Dymetman and Copperman 1998], Anoto introduced in 2001 a hardware digital pen that captures notes written on paper notebooks [Anoto AB 2007] (see Figure 2.3). With this approach, paper is augmented with a tiny dot pattern, which encodes locations in a notebook. The dot pattern can uniquely encode ~73 trillion sheets of letter-sized paper, so within a single notebook, every page is unique. While a user writes, an internal camera near the tip of the pen (pointed at the page) tracks the dots. This provides a mobile and robust solution to handwriting capture. As a result, much of the recent pen-and-paper research (including our own) is based on the Anoto system.

The augmentation technique has impact on the FORM FACTOR and USE cases of the application (see Figure 2.4). For example, DigitalDesk's top-mounted system works well for tasks within an office. On the other hand, our research chooses digital pens to enable mobile work (*e.g.*, for field biologists). There are tradeoffs: the digital pens and notebooks are small

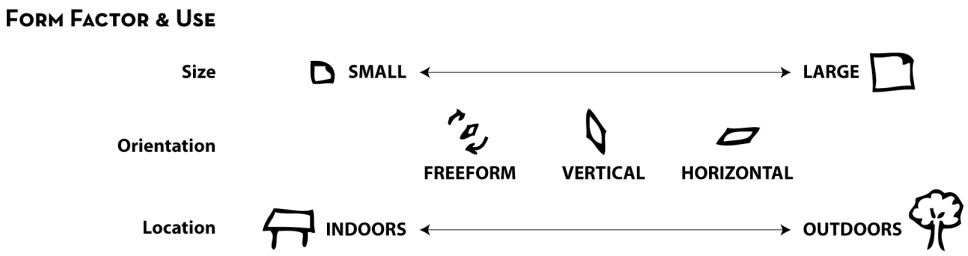


FIGURE 2.4. FORM FACTOR & USE determine the size and orientation of the paper interface, and where it will be used. These design parameters are interrelated. For example, a small and freeform paper notebook will better enable mobile interactions (outdoors).

and robust, but they cannot provide graphical feedback like a Tablet PC or a desktop LCD.

2.1.1 Reading and Working with Documents

To enable users to find and view content, researchers have developed techniques for tracking, manipulating, and reading documents. Inspired by DigitalDesk, several systems explored ways to locate physical documents on a desk (see Figure 2.5). PlayAnywhere is a portable camera-and-projector system that can detect sheets of paper (and other objects) on a table [Wilson 2005]. It supports input with computer vision (optical flow tracks a user's hands), allowing zoom, pan, and rotate gestures. Kim's document tracking system adds digital search capability onto the physical desktop [Kim, Seitz, *et al.* 2004]. The camera tracks the motion of documents, and allows a remote user to browse the stack of paper through a direct manipulation interface (see Figure 2.5, right). While computer vision provides flexibility (*e.g.*, the use of hands as input), it is not a mobile solution (a scientist cannot bring the system out into the field). In our work, projectors are used to provide overlaid graphics for indoor scenarios (see SECTIONS 5.2 and 7.5). Our mobile interfaces of our work avoids these camera-and-projector techniques.

Sometimes, information is encoded in how we handle and manipulate paper artifacts. Mackay studied air traffic control and found that the paper flight strips used by controllers (people directing the flights) offered a flexibility that could not be matched by computer

FIGURE 2.5. Two recent projects extend Wellner's ideas. *Left*) PlayAnywhere tracks desktop objects (including paper) with edge detection [Wilson 2005]. *Right*) The Video Based Document Tracking system allows digital searches of physical documents placed on a desk [Kim, Bergman, *et al.* 2004].

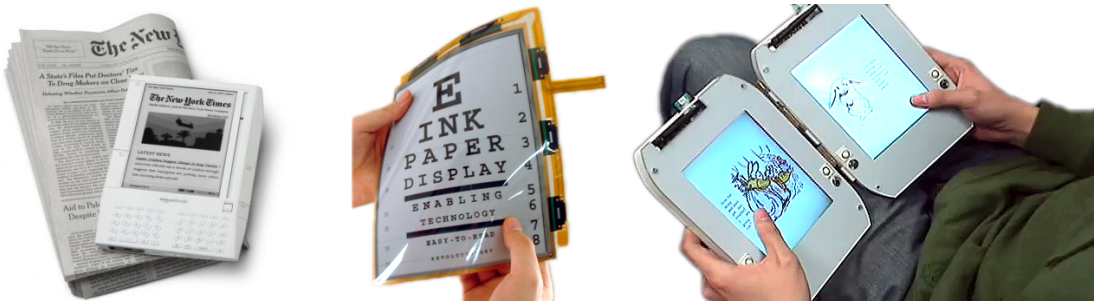


FIGURE 2.6. *Left*) E-Book readers such as the Amazon Kindle provide improved readability and battery life compared with conventional digital screens. *Middle*) Modern e-books are based on electronic paper technology (*e.g.*, E-Ink), which can sustain a high contrast image with little to no power. *Right*) Chen *et al.* have contributed new techniques to navigate e-books (*e.g.*, using flipping gestures) [Chen, *et al.* 2007].

interfaces [Mackay 1999]. The strips were annotated using pens, and passed between controllers. Strips were laid side-by-side and offset to represent the timing of incoming flights. In the end, Mackay's team chose not to replace the flight strips with a computer-only solution, but instead *augmented* them by using them as an interface to the software system. For example, one technique tracks the paper flight strips using an augmented stripholder (with metal contacts and resistors) [Mackay, *et al.* 1998].

Finally, there are ways to improve how people read documents. Paper provides good readability, as it does not strain the eyes, and is legible in the bright sunlight. Recent e-book readers (*e.g.*, Amazon's Kindle [Amazon 2007]) have attempted to achieve paper's readability and mobility (see Figure 2.6, left). They use electronic paper technology [E Ink 2007] to improve battery life and readability of the digital screens; e-ink uses charged ink particles that can maintain a stable image with little to no power. To improve navigation within documents, Chen *et al.* designed techniques to use physical flipping gestures to scan through documents (see Figure 2.6, right). We have also seen that people are comfortable using physical printed interfaces to navigate and control media. Listen Reader is an augmented paper book that allows a user to control audio streams by moving her hands near different parts of a page [Back, *et al.* 2001].

These techniques for tracking, manipulating, and reading documents can enhance the ButterflyNet and GIGAprints applications we present in CHAPTERS 4 and 5. In particular,

once electronic paper (charged ink particles) gains the ability to capture input from a stylus or inking pen, it can largely replace the digital paper (location-specifying dots on traditional paper) we use in our augmented notebooks and maps.

2.1.2 Capturing and Augmenting Handwritten Input

The tracking techniques in the previous section allow computers to provide foot-scale interactions. For example, Wilson's PlayAnywhere can project a photograph onto a sheet of paper, allowing the user to move and rotate the photo. To support finer interactions, the computer needs to capture user input at the granularity of pen taps or handwriting.

There is a class of work that enhances the capture of handwritten notes and sketches. Commercial approaches such as the Anoto pen [Anoto AB 2007] and Crosspad [IBM Pen Technologies 2007] digitize pen input so that it can be viewed on screen. The software enables users to export their notes to an image file for sharing (*e.g.*, through email). Additional processing with handwriting and sketch-recognition software (*e.g.*, Vision Object's MyScript) provides value beyond the digitized ink.

Other work uses the idea of capture-and-access to enhance note taking. Stifelman's Audio Notebook is a paper notebook that allows users to correlate their notes with audio recorded during a lecture [Stifelman, *et al.* 2001]. Users write on a pad that synchronizes handwriting with audio streamed into a microphone (see Figure 2.8, left). To retrieve audio, the user taps his pen near some ink on his notepad. Audio recorded at the time he wrote those annotations then plays through the speaker. This technique has been commercialized by Livescribe, a company that plans to release an audio smartpen (with integrated microphone and flash storage) in 2008 [Livescribe Inc. 2007] (see Figure 2.3, right).

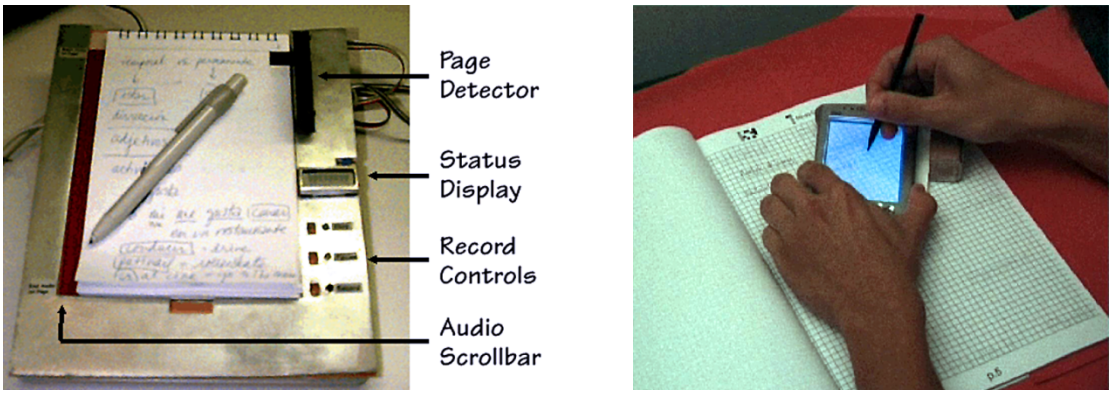


FIGURE 2.8. *Left)* Stifelman's AudioNotebook correlates handwritten notes with recorded audio. Users retrieve audio by tapping the pen to a location on the notepad. *Right)* Mackay's A-Book lets lab biologists add links and metadata to their handwritten notes. These digital annotations are stored in an information layer that is invisible in the physical notebook, but made visible through the interaction lens (a PDA).

Other techniques help users annotate and structure handwritten input (see Figure 2.8, right). Mackay's A-Book enhances the capture of research data in laboratory notebooks [Mackay, *et al.* 2002]. In the lab, a biologist uses a PDA as an *interaction lens* to add data to a virtual information layer. Users can create a table of contents, references between pages, and bookmarks to external sources. This work demonstrates the importance of augmenting scientists' practices, and was one of the main inspirations for ButterflyNet (see CHAPTER 4).

Handwriting is only one form of USER INPUT (see Figure 2.7). In applications, pen-

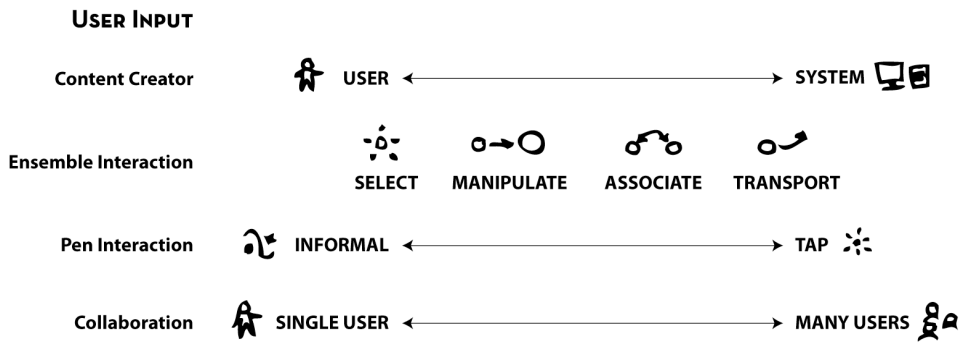


FIGURE 2.7. USER INPUT represents how the end user works with the paper interface and any supporting computers. The paper content can be generated mostly by the user (*e.g.*, handwriting) or by the system (*e.g.*, map imagery). The ensemble interaction category describes how the user affects digital data through paper-based interactions. When working on paper, pen interactions can be informal sketches, or they can be constrained (*e.g.*, gestures or taps). Finally, paper + digital applications can support one or more users.

based interactions can vary from informal handwriting and sketching to more constrained gestures, drags, and taps. Ensemble interactions describe how users work with multiple tools in tandem. For example, the A-BOOK provides an auxiliary device that helps users associate digital and physical content. Finally, these systems mainly support single-user scenarios. In contrast, our GIGAprints support collaboration between two, four, or more people.

2.1.3 Coordinating Paper and Digital Artifacts

The idea of connecting paper and digital artifacts arises from the observation that content printed or written on paper often has a digital counterpart. For example, a scientific citation printed in this dissertation (*e.g.*, [Yeh, Liao, *et al.* 2006]) corresponds to a digital document accessible on the Web. Similarly, a sticky note containing "Meet Frank, 2PM @ Starbucks" might mirror an appointment entered into a mobile phone. Much of the augmented paper research has attempted to reconcile these artifacts, to enable actions on one to affect the other. For example, Paper PDA [Avrahami, *et al.* 2001, Heiner, *et al.* 1999] demonstrated a synchronized digital and paper day planner. Handwritten pages are scanned in, recognized, and inserted into a digital calendar (see Figure 2.9, left). This technique does not record the

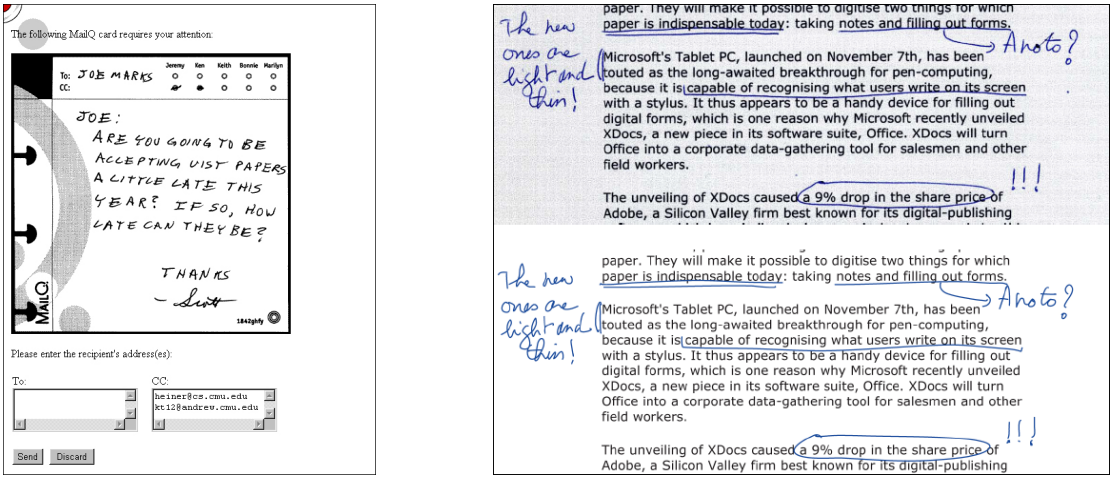


FIGURE 2.9. *Left)* Paper PDA allows entries on a paper planner to be synchronized to calendaring software [Heiner, *et al.* 1999]. Pages are scanned in and recognized by the system. *Right)* The PADD system overlays handwritten annotations onto the original digital document [Guimbretière 2003]. Multiple systems have built on top of PADD's infrastructure (*e.g.*, [Conroy, *et al.* 2004, Liao, *et al.* 2005]).

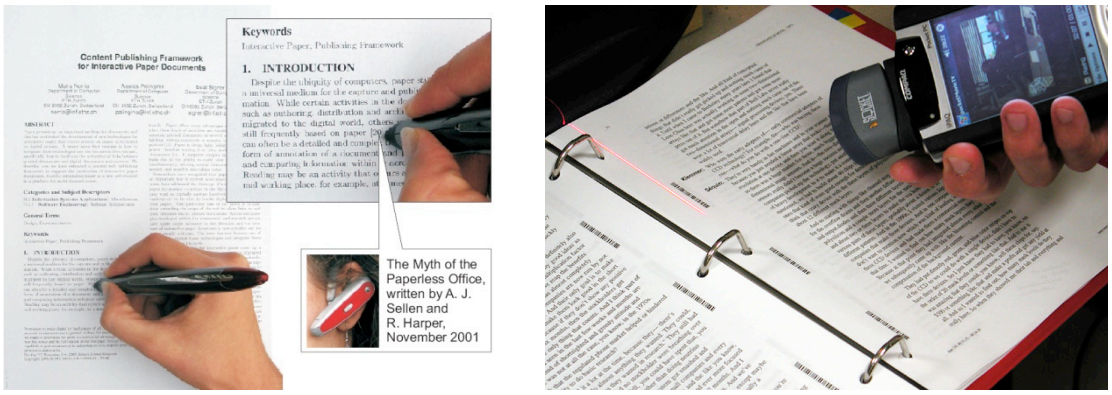


FIGURE 2.10. *Left)* With Print-n-Link, a reader can tap a digital pen to a scientific citation to hear the authors' names and related details [Signer 2006]. *Right*) Books with Voices allows users to retrieve media by scanning a barcode embedded in the oral history transcript [Klemmer, *et al.* 2003]. The retrieved media is played on the handset.

timing of individual ink strokes, necessary for media correlation (*e.g.*, see CHAPTER 4).

Guimbretière later demonstrated with PADD [Guimbretière 2003] that annotations on paper can be automatically overlaid onto the original digital document (see Figure 2.9, right). This enhances remote collaboration, as a teacher can handwrite comments on a printout of an essay and quickly share her annotations by emailing them to her student. PADD treats the printout and the PDF as physical and digital *views* of the same document *model*, to use the model-view-controller (MVC) terminology [Krasner and Pope 1988]. Thus, when the end user acts through a view, the effects propagate (eventually) to all of the views. As follow-up to PADD, ProofRite integrates these annotations into a word processor [Conroy, *et al.* 2004]. While the author integrates changes, remaining annotations will reflow along with the text.

Instead of coordinating entire documents, a different approach provides links from *within* a document. For example, Print-n-Link enables a reader to tap on a scientific citation in a paper document, and hear it in a Bluetooth audio ear piece [Norrie, *et al.* 2006]. The links are created automatically by the system (see Figure 2.10). The user can also retrieve the PDF from a document server.

Leveraging the fact that paper can be used as a physical token, WebStickers allows users to associate web pages with a sticky note [Ljungstrand, *et al.* 2000] that can be passed

between collaborators. A user loads the target URL by scanning a barcode attached to the note. Klemmer's Books with Voices also uses barcodes, but to link to locations *within* a media file [Klemmer, *et al.* 2003]. Barcode indices are printed throughout an oral history transcript. Scanning a link with a barcode reader (attached to a PDA) will open the video to the closest point, allowing a reader to efficiently retrieve the source media associated to the transcript (see Figure 2.10, right). ButterflyNet (CHAPTER 4) also provides linking between pages and media, but unlike the applications in this section, it allows end users to *author* those links in the field (see SECTION 4.2). In comparison, the links in Books with Voices are prepared by the historian, but used by the reader.

2.1.4 Issuing Commands

The final approach to augmenting paper (beyond reading, capture, and document linking) is to enable users to issue *commands* to a computer. Some of these commands map to existing pen-and-paper practices. For instance, a squiggle over printed text normally means *delete*. If the computer can recognize this annotation, it can automate the tedious steps of integrating changes into a digital document.

The PapierCraft system [Liao, *et al.* 2005] maps a set of gestures to document editing functions, including copy and paste. For example, a user can mark an image in a source document and gesture to have it pasted into a destination document. Once the pen is synchronized, PapierCraft processes these gestures and generates the output document. PapierCraft's gesture set inspired ButterflyNet's gesture-based linking.

Signer and Norrie's PaperPoint (see Figure 2.11, left) allows a user to control a PowerPoint slide deck from a printed version of the slides [Signer and Norrie 2007]. The user taps her pen to the thumbnail, and PowerPoint advances to the correct slide. Annotations on the slide are displayed on screen.

Some paper-based systems act as *tangible interfaces* for controlling software systems. SiteView [Beckmann and Dey 2003] builds on the Papier-Mâché toolkit [Klemmer, *et al.* 2004] to allow users to plan a home automation system by manipulating physical icons, such as a paper light bulb. However, since paper currently does not provide real-time graphical updates, the commands from gestures and physical icons are usually unidirectional. One solution is to georeference the graphical output by projecting it over or under the physical token (*e.g.*, as in Designers' Outpost [Klemmer, *et al.* 2001]). In the GIGAprints work, we use projectors to provide overlaid application output. However, for mobility, most applications should provide co-located output (on a nearby handheld device), or remotely (on the web). These graphical output alternatives can be referred to as *spatial coordination* of the paper+digital application (see Figure 2.12). Output can also vary in time. Synchronous feedback is ideal for most applications. However, a field scientist may prefer to process her field notes asynchronously (once she returns to the laboratory).

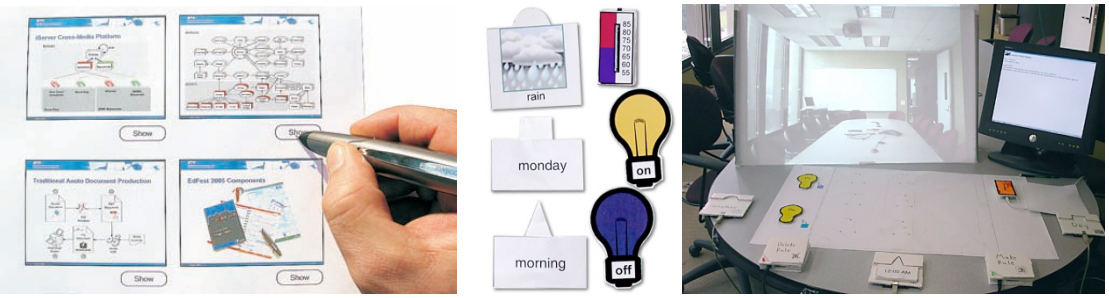


FIGURE 2.11. *Left)* With PaperPoint, a presenter can navigate his slideshow by interacting with a paper printout [Signer 2006]. *Right)* SiteView allows users to control a home automation system with physical paper icons [Beckmann and Dey 2003]. This prototype was built on the Papier-Mâché tangible input toolkit [Klemmer 2004].

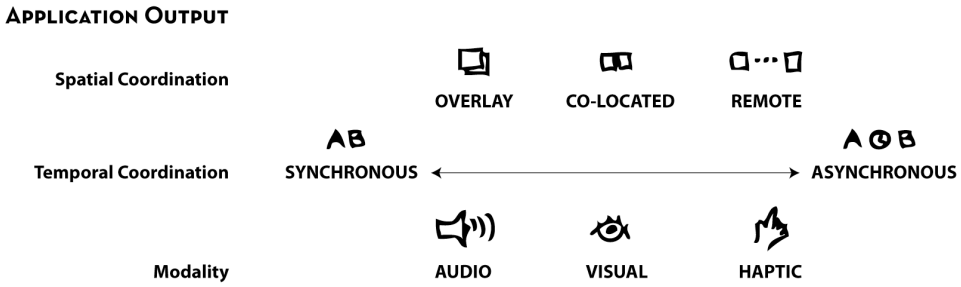


FIGURE 2.12. APPLICATION OUTPUT refers to how the device ensemble provides feedback to the user. Graphical output can be overlaid on top of printed content, displayed on a nearby device, or presented remotely (*e.g.*, on the web). Feedback can happen in real-time, or be processed in batched (*e.g.*, when a user returns to his office). Visual feedback is the most common, but mobile scenarios may benefit from audio or haptic feedback.

Finally, some systems explore different modalities of input and output. For example, NISMap and Rasa [Cohen and McGee 2004] show how pen-and-paper interfaces can provide robustness in field situations, and provide a class of multimodal interactions with paper. The systems contribute fusion techniques to integrate speech with pen strokes. Cohen and McGee have since commercialized their efforts in a company called Adapx. None of our systems or tools support multimodal fusion to disambiguate input. However, we were inspired by this work and explored audio-only output in the AudioGuide GIGAprint (see CHAPTER 5).

Technology for Augmenting Paper

The interactions in SECTIONS 2.1.1-4 are based on different enabling technologies, each with tradeoffs. DigitalDesk relies on computer vision to process interactions. While the ideas have been miniaturized in PlayAnywhere, the large-scale camera-based solution does not provide fine-grained capture and also constrains mobility. Paper PDA and systems such as XAX [Johnson*, et al.* 1993] and Palette [Nelson*, et al.* 1999] used slower imaging equipment (*e.g.*, scanners, fax machines, and barcode readers). Scanners provide high-resolution data, but do not capture the timing of individual pen strokes. Digitizing tablets can capture strokes in real-time. With the Cross Pad, a user places a notepad on a tablet, which detects the pen input. However, this tablet adds extra weight, which is also undesirable in mobile scenarios.

Barcodes (1D and 2D) are useful for attaching unique identifiers to objects. Books with Voices used barcodes to provide a way to index into an interview transcript [Klemmer, *et al.* 2003]. Barcodes also can be combined with computer vision techniques to track sheets of paper. This allows software to project graphics onto paper in a sketching environment (*e.g.*, [Haller, *et al.* 2006]), or present graphics in a head-mounted display for an augmented reality experience (*e.g.*, see [White, *et al.* 2006]).

Currently, the most popular method for augmenting paper is the pen-top computer, which provides the benefits of the digitizing tablet in a small form factor (see Figure 2.13). The Anoto digital pen is one type of pen-top computer, and is the platform of choice for today's research. PaperToolkit uses Anoto for its combination of mobility, robustness, and capture resolution. As discussed earlier, the internal camera (inside the tip and pointed toward the page) tracks a grid of dots, which encode the unique location within a notebook. As the user writes, the pen saves the location, time, and force of each stroke, while laying down physical ink on the paper. The strokes can be sent to a PC in batched mode (when a user drops his pen into the USB cradle), or in streaming mode (coordinates are sent in real-time to a Bluetooth port on a nearby PC). ButterflyNet relies on batched-mode processing, while GIGAprints concentrates on real-time input (as it enables richer ensemble interactions [Schilit and Sengupta 2004]). PaperToolkit supports both.

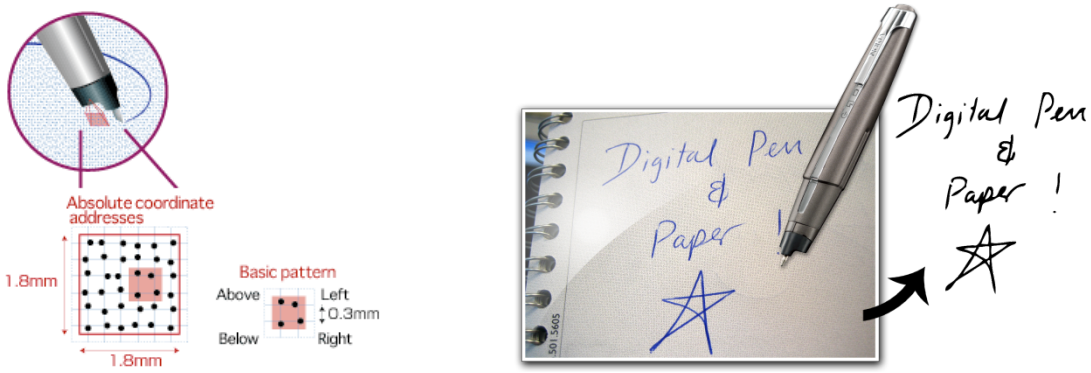


FIGURE 2.13. *Left*) The Anoto digital pen system uses a small camera inside the pen to track a dot pattern printed on the page. The dot pattern encodes the X and Y location within a notebook (each page is unique). The pen also captures the force and timestamp of each pen stroke, and stores this information in its internal flash memory. *Right*) PaperToolkit uses the Anoto platform to acquire pen coordinates in real-time and batched modes.

2.2 Design Space for Paper + Digital Applications

In the previous section, we presented three categories of design parameters for augmented paper systems. Combined, this design space can help researchers reason about paper applications and interactions (see Figure 2.14). A given system may encompass multiple points along a dimension. For example, our ButterflyNet system contains parts that are used outdoors (the field notebook) and parts that are used indoors (the browser). The design space was developed through our experience building systems such as ButterflyNet and GIGAprints, and will be used in later chapters to explain our design choices.

The space includes three categories: form factor & use, user input, and application output. FORM FACTOR & USE determines the size of the interface, its orientation, and whether it is used inside or out in the field.

USER INPUT describes how one or more people invoke actions on the device ensemble. The user might generate most of the content (*e.g.*, by handwriting) or the system may provide most of the content (*e.g.*, printed maps). The paper and digital pieces may coordinate in various ways. A pen gesture on paper might *select* a digital resource. An action

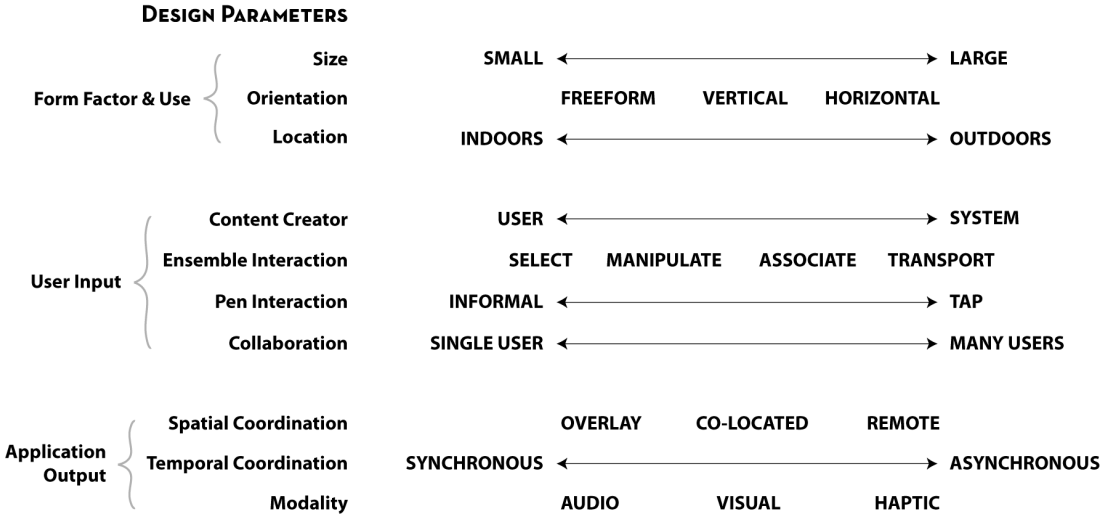


FIGURE 2.14. This design space describes the range of applications explored by previous work, and helps to determine areas that have not been explored. For example, Audio Notebook, A-Book, and PADD are freeform interfaces where the user generates most of the content. They are small paper interfaces, and do not support collaboration.

could *transport* content from paper to a handheld display. A button press might *associate* a photograph on the computer with a location in a paper notebook. Pen interactions can be constrained taps and gestures, or they may take the form of freeform ink.

Finally, APPLICATION OUTPUT can come in the form of audio, visual, or haptic cues. It can be provided near the paper input surface (collocated) or in a remote fashion (*e.g.*, on the web). Digital graphical output can also be overlaid on paper. This output can happen in real-time (synchronous) or at a later time (asynchronously).

We can compare the systems described in this chapter using the design space. For example, Wellner's DigitalDesk provides overlaid digital graphics, and is meant primarily for a single user. In comparison, Mackay's A-book provides digital feedback in a co-located manner, on a nearby PDA.

Print-n-Link supports tap interactions (on content printed by the system), while PADD accepts informal inking interactions, with content generated primarily by the end user. AudioNotebook only provides audio feedback (beyond the written notes), while other researchers have looked into haptic pen feedback (*e.g.*, [Lee*, et al.* 2004, Liao*, et al.* 2006]).

The design space can help software developers reason about design decisions. By choosing a target task to support, the developer automatically determines values for some axes. For example, to create an augmented field notebook for journalists, the developer should create an application that supports *outdoors* and *single-user* scenarios. Then, the developer can decide what types of feedback would be appropriate. Audio feedback may be preferable, because the user will not have access to a handheld device. Alternatively, visual feedback might be preferred, as the user will need to retrieve and read content in the field.

2.3 User Interface Platforms

The PaperToolkit platform builds on existing techniques in user interface tools and toolkits. Most related are the existing platforms for building paper + digital software. The basic Anoto tools allow a programmer to 1) create and print physical forms augmented with dot-pattern, 2) retrieve the data from a wired pen synch, and 3) communicate with the pen

over Bluetooth wireless to retrieve coordinates in real time [Anoto AB 2007]. This SDK does not model event handling through interface components, but rather allows the retrieval of ink strokes attached to page areas, reflecting their concentration on form-filling applications.

Other modern platforms build on the Anoto platform to provide features beyond ink capture and rendering. For example, Guimbretière's Paper Augmented Digital Documents (PADD) automatically augments forms with the dot pattern, and overlays ink annotations back onto the original digital document [Guimbretière 2003]. While PADD handles only batched input, iPaper [Signer 2006] provides real-time retrieval of media associated to areas on paper, and its active components provide basic event handling (see Figure 2.15). Finally, LiveScribe is introducing a Java-based SDK in 2008 to allow programmers to build for their smartpen [Livescribe Inc. 2007]. Unlike PADD and LiveScribe's platforms, PaperToolkit supports real-time events. It goes beyond the iPaper work by introducing novel design techniques and evaluating the toolkit with a long-term external deployment.

Learning from Sketch-Based Toolkits and Authoring Environments

Since the primary input device in this dissertation is a digital *pen*, we can learn from research on sketching toolkits and environments. Recall that instead of augmenting paper with computation, we can improve on the advantages of pen-and-paper with portable computers (see Figure 2.1). Tablet PCs do this by including a stylus, providing methods to invoke

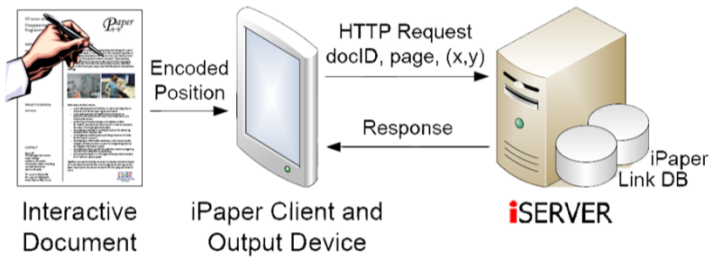


FIGURE 2.15. iPaper provides real-time media and code retrieval through a client device. PaperToolkit contributes beyond the retrieval model with an explicit notion of interface handlers, an in-depth evaluation of the toolkit with external developers, and novel design tools and techniques to make developing and debugging paper interfaces easier.

commands through pen gestures, and by processing input with handwriting recognition. Microsoft provides SDK support for tablets [Microsoft 2007], but most of it centers around the input and display of digital ink. Windows Vista includes single stroke pen gestures, including stylus *flicks*, which act as PAGE UP and PAGE DOWN. In building PaperToolkit, we replicate these modeling, rendering, and recognition features in Java.

Hong's SATIN [Hong and Landay 2000] is a research toolkit for tablet and stylus interactions. In addition to including rendering facilities for ink strokes, SATIN provides support for pen gestures, ink simplification, and marking menus. Many of these techniques can inform other pen-based toolkits. For example, PaperToolkit includes spline-based ink rendering and stroke simplification. SATIN has been used to build sketch-based projects (see Figure 2.16), including the DENIM web design environment [Newman, *et al.* 2003].

Looking toward sketching environments, one important result is that users can be more effective if they operate on digital ink as *object clusters*, instead of the underlying vector strokes or bitmap graphic. Saund *et al.*'s ScanScribe [Saund, *et al.* 2003] used smart clustering algorithms to allow users to manipulate sketches acquired through a scanned sheet or whiteboard (see Figure 2.17, left). ScanScribe introduced a way to organize clusters in a flexible lattice data structure, which allows the end user to choose which groups of ink to manipulate. PaperToolkit can be extended with these techniques to help end users work with their sketches after they are imported into a graphical editor.

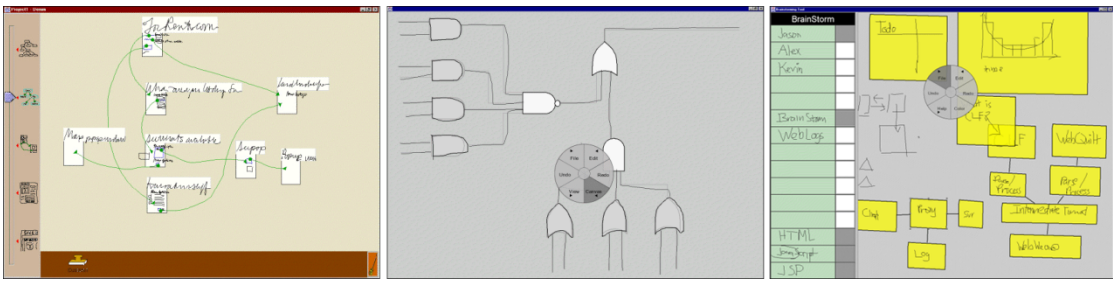


FIGURE 2.16. Hong's SATIN toolkit [Hong and Landay 2000] supports input from digitizing tablets and tablet PCs. DENIM, SketchySPICE, and Brainstorm were sketch-based applications built on SATIN. PaperToolkit's digital ink support was inspired by SATIN's techniques.

Another sketch-based system is DENIM, which allows web site designers to use a digitizing tablet (or Tablet PC) to design an entire website (see Figure 2.17, right). It provides pen gestures to activate common commands (*e.g.*, panning, undo, and redo). DENIM adopts an approach of not immediately recognizing ink strokes, as displaying recognition results while the user sketches is distracting. Recognition errors also frustrate users.

Haller *et al.* have used digital pens to create a large table-oriented sketching environment [Haller, *et al.* 2006]. Users can draw on the table and on paper sheets; the pen input is translated into Adobe Illustrator commands. The output is projected onto the table, so feedback is georeferenced with user input (see Figure 2.18). To support applications like these, PaperToolkit would need a way to align projected output with pen-based input.

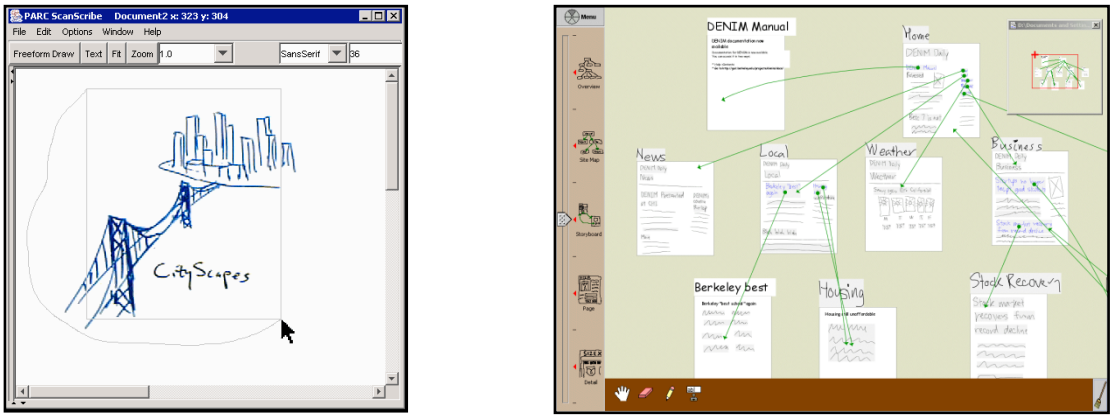


FIGURE 2.17. *Left*) ScanScribe provides ways to cluster and select ink strokes. Pictured is a technique to overload the selection command by combining the lasso and rubber band operators. *Right*) DENIM provides semantic zooming for ink objects (web pages). It hides the results of ink stroke recognition, since visibly incorrect recognition inhibits fluid interaction.

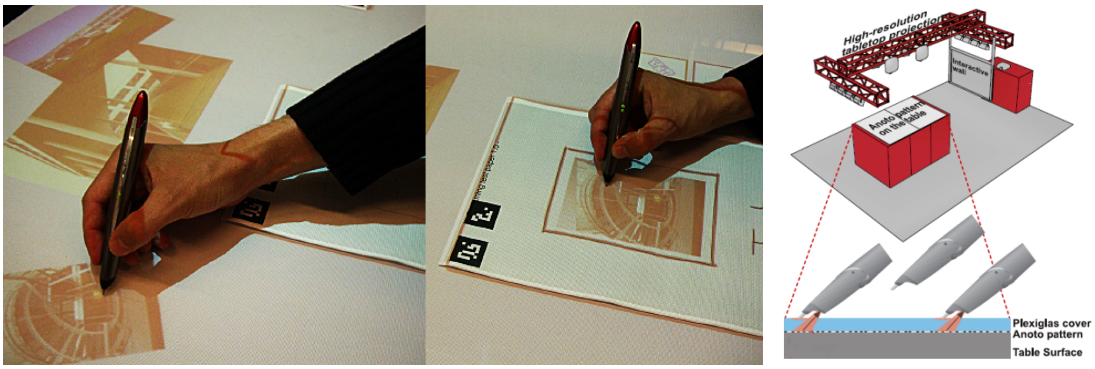


FIGURE 2.18. Haller's sketching environment allows interactions such as a pick-and-drop. The prints are tracked using 2D barcodes. To create the interactive table, acrylic is layered over a large sheet of Anoto dots.

For wall-oriented sketching environments, Smart Technologies has developed multiple systems, including digital whiteboards that use vision to detect touch (DViT) and cameras that can capture brainstorming sessions in front of a regular whiteboard [Smart Technologies 2007]. While GIGAprints can support the capture of sketches, they cannot provide real-time interactive graphics like these digital whiteboards.

Learning from Graphical and Tangible User Interface Toolkits

PaperToolkit's programming model and architecture was informed by existing toolkits for graphical user interfaces, subArctic [Hudson, *et al.* 2005] and Java Swing [Sun Microsystems 2007]. subArctic is an extensible GUI toolkit that demonstrates methods to maintain state in event handlers, and methods to compose multiple event handlers to achieve new interactions. To provide a low learning threshold for Swing programmers (PaperToolkit's target audience), our handlers are modeled after Swing *listeners*. As in Swing, a programmer adds listener objects to the UI component that receives the input.

We also applied techniques from tangible toolkits to designing PaperToolkit. For example, Klemmer's Papier-Mâché ties computer vision, 2D barcodes, and RFID tags together into a high-level architecture for input in tangible computing systems [Klemmer, *et al.* 2004]. Papier-Mâché provides a monitoring view that reports events when an augmented object is sensed, is continued to be sensed, or is removed (see Figure 2.19). This view inspired

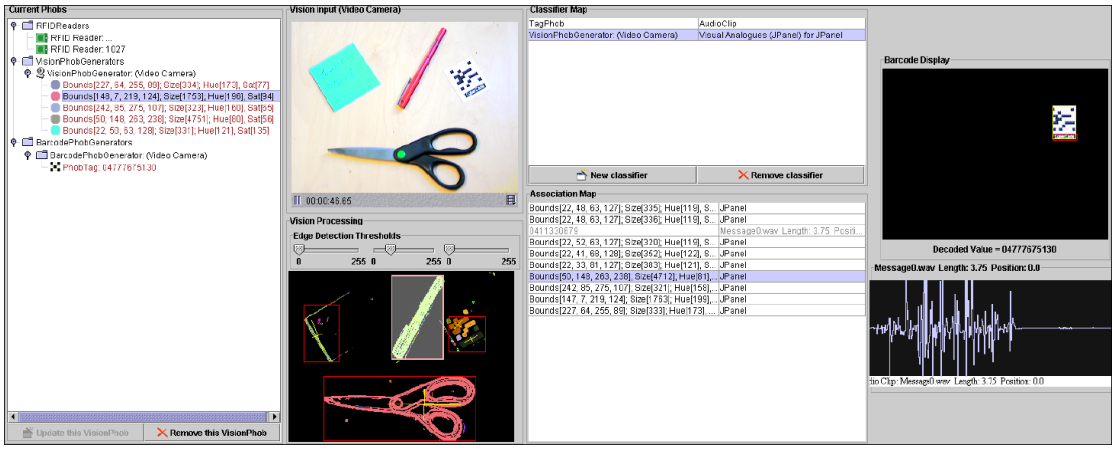


FIGURE 2.19. Papier-Mâché provided a monitoring view to help developers understand the toolkit's internal state [Klemmer 2004]. For example, it displays the physical objects currently sensed by the system. This tool inspired PaperToolkit's monitoring view (see SECTION 8.6).

PaperToolkit's monitoring tool (see SECTION 8.6). Papier-Mâché models ambiguity because vision input is inexact. In contrast, PaperToolkit does not handle ambiguity, as incoming pen coordinates are treated as analogous to mouse coordinates for GUIs.

2.4 Evaluating Toolkits

The main challenge in conducting toolkit research is the evaluation of the resulting abstractions and design tools. At the lowest levels, we can try to determine whether the APIs are easy to learn and remember. At the same time, we might gauge whether the design tools make the programmer more efficient and less error-prone.

Previous work has demonstrated a number of techniques to measure these aspects of a toolkit. In particular, Klemmer demonstrated a mixed-method approach to evaluate toolkit research—combining laboratory study, long term use, and inspection of source code produced by developers [Klemmer 2004]. In one case, he used source code metrics to support the evaluation of Papier-Mâché. He found that SiteView's vision input component took 30 lines of Papier-Mâché code, while comparable functionality in Designers' Outpost (built without Papier-Mâché) required thousands. We extend these methods by analyzing

source code manually and through automatic scripts to calculate statistics (on API usage and development strategies) to inform PaperToolkit's iterative design.

Building the New and Rebuilding the Old

An effective way to demonstrate that a toolkit improves on prior efforts is to show that it allows programmers to create applications that were not possible before. For example, in the d.tools hardware prototyping toolkit, Hartmann and collaborators prototyped a novel application which provided physical drawers for a digital table [Hartmann, *et al.* 2006].

The toolkit developer can also rewrite existing applications to demonstrate which toolkit features would save the programmer time and effort (*e.g.*, as measured by lines of code). For example, Klemmer's and colleagues used Papier-Mâché to reimplement several applications that inspired their work [Klemmer 2004]. The reimplemented version of Books with Voices required far fewer lines of code than the original; additionally, it was straightforward to swap in alternate capture technologies.

PaperToolkit uses these techniques to demonstrate its utility. CHAPTER 7 shows how students in our lab have created peer-reviewed research using the toolkit. The author also reimplemented the basic parts of ButterflyNet and one map-based GIGAprint to show that the developers would have used less time and effort had PaperToolkit existed earlier.

Laboratory and Longitudinal Studies

Clarke reported that Microsoft's Visual Studio group evaluates the usability of APIs through laboratory studies, personas, and applications of the cognitive dimensions framework [Clarke 2004, 2005]. Laboratory studies are useful for studying how programmers approach a toolkit for the first time, or can also be used to observe programming and debugging strategies. For example, programmers can be asked to debug small programs while they are observed by an experimenter (*e.g.*, see [Romero, *et al.* 2003, Rosson and Carroll 1996]). With PaperToolkit, we instead deployed a longer-term user study (~6 weeks) and also made it available to the research community. While we were not able to control the tools and tasks people chose, the long-term deployment shed light on how novices learn the

Cognitive Dimension	Description
Abstraction Gradient	Provide an appropriate level of encapsulation for details.
Closeness of Mapping	The notation should map well to the problem domain.
Consistency	If part of a tool is learned, other parts should be guessable.
Diffuseness/Terseness	How much space the tool requires to produce a result.
Error-Proneness	Does the tool induce mistakes?
Hard Mental Operations	Does the user need external cognitive aids?
Hidden Dependencies	Are cause/effect relations (or other dependencies) visible?
Juxtaposability	Can parts be compared side-by-side at the same time?
Premature Commitment	Tasks can be done in any order, and can be undone.
Progressive Evaluation	Can the user evaluate an incomplete solution?
Role Expressiveness	The purpose of each component should be obvious.
Secondary Notation	Does the tool provide extra hints to aid usage?
Viscosity	It should not require much effort to make changes.
Visibility	Required actions should be easily identifiable.

TABLE 2.1. The cognitive dimensions [Green and Petre 1996] are a useful way to reason about the usability of programming tools. This method does not require users, and should be used in parallel with laboratory and/or longitudinal evaluations.

system over time, and how they eventually reach the limits of the toolkit.

Heuristic Evaluation

Green's cognitive dimensions framework [Green and Petre 1996] is a set of design guidelines that can be used to analyze the effectiveness of programming tools (or compare tool with each other). Cognitive dimensions is a *discount usability method*, and can be used without users to analyze the tradeoffs within the toolkit. The 14 axes provide a way to perform a heuristic evaluation, akin to Nielsen's checklist [Nielsen 2007]. The dimensions are listed in Table 2.1, with short explanations. We refer to the cognitive dimensions in CHAPTERS 7 and 8, to evaluate the toolkit and its design and debugging tools.

The Threshold, Walls, and Ceiling of Programming Tools

Finally, we can attempt to analyze tool support by inspecting its threshold, ceiling, and walls [Myers, *et al.* 2000, Shneiderman, *et al.* 2005]. The learning *threshold* expresses how easily a novice can pick up a programming model and API (the left side of Figure 2.20). The *ceiling*

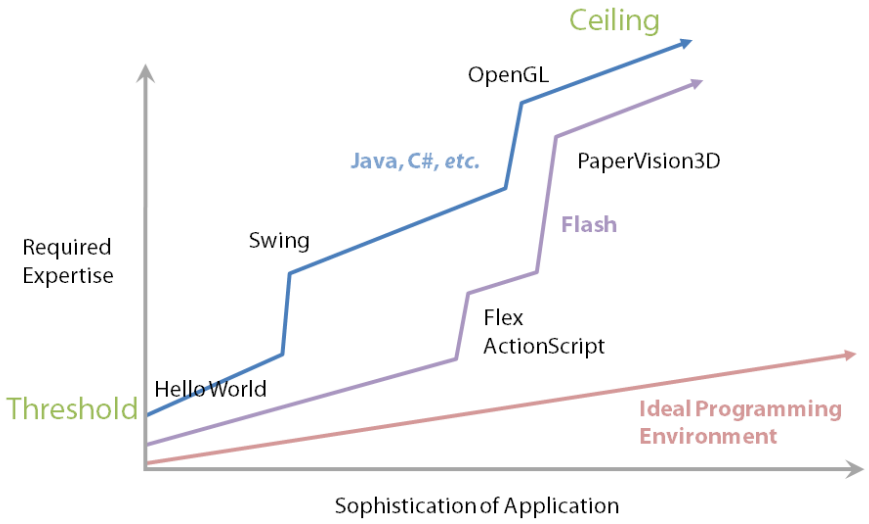


FIGURE 2.20. This picture depicts learning curves for different programming environments (adapted from Brad Myers' presentations). The graph communicates the relative difficulties of learning different APIs and programming tools (the shape of the curve is not empirical). An ideal environment provides a shallow learning curve, to enable programmers to make sophisticated programs with little expertise.

expresses the point at which expert programmers reach the limits of what can easily be implemented using the tools. The *walls* address the creativity support for programming tools, whether programmers will be able to make a wide variety of solutions. Ideally, a toolkit has a low threshold, wide walls, and a high ceiling.

PaperToolkit concentrates on techniques to lower the threshold, but without sacrificing the ability to program in Java (high ceiling). We looked for inspiration in low-threshold programming tools such as Pane's HANDS, which provided a learnable programming system for children [Pane 2002]. Pausch and colleagues created the Alice programming system to teach children (and especially young girls) how to tell stories through animation [Dann, *et al.* 2005, Kelleher and Pausch 2005]. HANDS and Alice use visual techniques to communicate application state to the programmer; PaperToolkit's debugging tools use iconic representations to communicate toolkit state, such as the pen's activity.

Even with general-purpose languages (*e.g.*, Java), projects such as Ko's WhyLine show that there is much value in making coding and debugging easier [Ko and Myers 2004]. This work demonstrates that we can speed up debugging by allowing programmers to ask why and why not questions when locating a bug [Ko and Myers 2004]. PaperToolkit's debugging tools (see CHAPTER 8) are inspired by some of these techniques. In particular, the monitoring tool provides a timeline visualization of events (like WhyLine Alice's timeline of causality). The main difference is that PaperToolkit's monitoring happens passively, while WhyLine requires a user to ask an initial question.

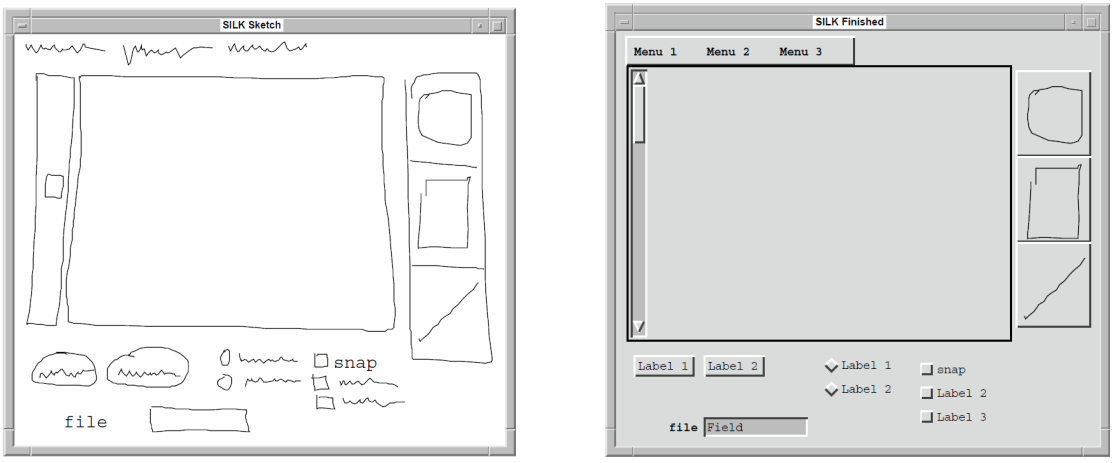


FIGURE 2.21. Landay's SILK enabled designers with no programming experience to generate working graphical user interfaces by sketching them with a digitizing tablet.

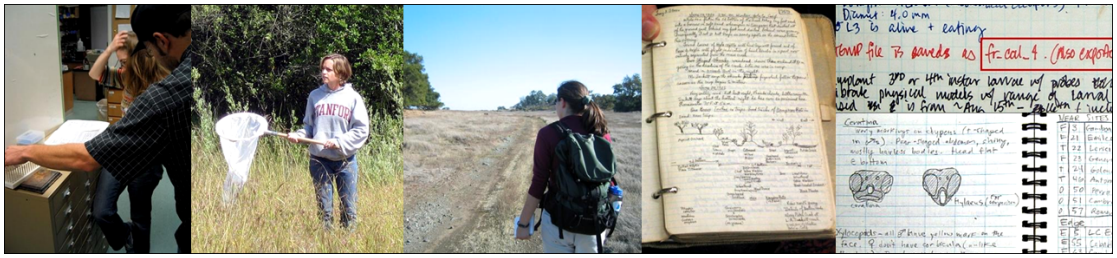
Better visuals can help a programmer debug, but lower-threshold specification techniques can help programmers create programs faster and with fewer errors. Landay tackled the problem of user interface prototyping [Landay and Myers 1995]. With SILK, designers were able to quickly create a working graphical user interface by sketching on a digitizing tablet (see Figure 2.21). Most standard layouts can be generated without writing a single line of code. PaperToolkit also allows developers to sketch interfaces, and adds extensibility by allowing users to write the names of handlers, instead of relying on a fixed set of templates.

2.5 Summary

This chapter presented the body of work that informed many of the design decisions for PaperToolkit. Systems such as AudioNotebook and A-Book helped us identify requirements for the ideal toolkit, such as the ability to correlate incoming ink with other media, or integrate with handheld devices as an auxiliary display. Existing platforms such as PADD and iPaper demonstrate different approaches to providing digital pen interactions. While existing work gives us high-level lessons, it does not outline the specific requirements for our toolkit, or the applications to be built. The next chapter presents our field study of biologists, in an attempt to distill these requirements.

3

Learning from Work Practices



that Integrate Pen and Paper

Before designing the toolkit, we must first determine the range of applications the toolkit should support. One method is to examine an extreme user group, and generalize findings from that group to the larger population. This chapter describes a study of field biologists, and presents themes to inform technology support for field biology research. The author spent more than 400 hours observing and working with biologists. The study includes structured interviews of 13 biologists, a collaboration on field research, and a 20-week class to train as a researcher and docent at the Jasper Ridge Biological Preserve (JRBP). The study revealed that field biologists depend on paper's affordances, but can benefit from software

that integrates handwritten content with existing software, such as spreadsheets, mapping, and statistics. The findings map onto other domains, including design, law, and engineering.

3.1 Background and Methods

Our observational study draws from the ethnographic methods used by anthropologists, adapted for technology design. For example, Forsythe describes how ethnographies should be done by outsiders, who can detect "tacit knowledge," invisible to insiders (*i.e.*, biologists) [Forsythe 1999]. The author, as an outsider, spent time in the field working alongside biologists, observing their tools and work practices. Observations were recorded through photographs and entries in a notebook.

The second method is inspired by the contextual design process [Beyer and Holtzblatt 1997]. We began by conducting structured interviews of 12 biologists affiliated with Stanford, JRBP, and the California Academy of Sciences (CAS). Each interview took place at biologist's primary workplace, either in the field or lab. For example, on one interview, the author hiked for four hours with the biologist, to and from her field site at Jasper Ridge (see Figure 3.1). We captured notes, photographs, and audio recordings during the interviews. We also examined the biologist's tools and artifacts by reviewing their field and lab notes.

Third, the author took steps to learn to *become* a biologist, taking a 20-week class at Jasper Ridge to become a researcher and docent. He subsequently collaborated on active research projects. This method is inspired by Klemmer's work on Books with Voices, where

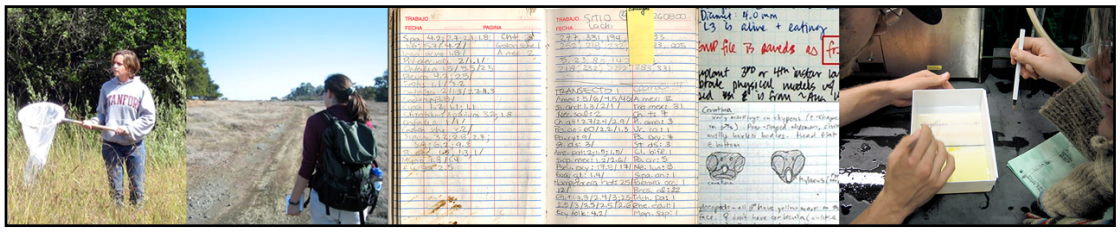


FIGURE 3.1. *Left*) Field biologists use nets and other tools to capture physical specimens, and notebooks and datasheets to capture measurements from field experiments. *Middle*) The paper notebooks contain rich hand-authored content, including drawings, tables of measurements, and text describing procedures. *Right*) Back in the laboratory, biologists run lab experiments, and also use computers to process and analyze their data.

he learned to become an oral historian [Klemmer, *et al.* 2003]. In total, this dissertation work comprises 400 hours of field work observing, talking to, learning from, and working alongside more than thirty field biologists. The process was documented with digital photographs, audio, video, and handwritten notes.

3.2 Themes that Emerged from the Study

This section summarizes the major themes that emerged from the field study: mobility, robustness, flexibility, collaboration, and transformation. These themes describe the field biologist's tasks, working environment, and the requirements needed to support their work. Each theme is supported by specific notes made during interviews or by field observations.

3.2.1 Mobility

The first observation was the need to support a field biologist's mobility (see Figure 3.2). For the biologists who work at Jasper Ridge (located near Stanford's campus), many of their experiments are set up away from the central field station. For example, during the first interview, we hiked two hours out to the biologist's field site, where she took measurements on the growth of the Yellow Starthistle plant. Biologists at Jasper Ridge also use electric carts (or even pickup trucks) to access the more remote sites. This is more pronounced when the field site is far removed from the comforts of Stanford's campus. Many biologists spend field seasons in places such as the Colorado Rocky Mountains, the Yucatán, or the plains of India.

FIGURE 3.2. These pictures (from a field season in Costa Rica) show biologists using nets to capture bees at different field sites. Any additional equipment (*e.g.*, cameras) must be carried in a backpack or hung around the neck.

Field time is valuable, and much more limited than time in the field station or laboratory. In one interview of a bee biologist, we learned that it was important for his team to move between field sites efficiently: "We do five or six sites a day, pretty early in the morning. We set traps out, do the netting, move on to the next site." In the afternoon, they retrieve the traps, as the rain damages the equipment. They spend the entire evening in the field station, identifying and measuring bees.

Field tools must be *lightweight* to support mobile users. For example, a paper notebook is physically lighter than Tablet PCs. Additionally, electronic field tools must have long lasting batteries (or require little to no power). Or, at minimum, the batteries must be easily replaceable while in the field (*e.g.*, using standard AA-sized batteries). Biologists currently carry digital cameras and GPS devices, and sometimes even handheld computers (*e.g.*, PDAs). Finally, electronic displays must be readable in the sun. Currently, Tablet PCs are unusable in bright sunlight.

3.2.2 Fluidity

Mobile users prefer tools that afford fluidity. Fast and smooth interaction can come from different sources, including the time to access and navigate the tool, and the input flexibility it provides. With mobile computing, Starner reports that there seems to be a time threshold beyond which users will not use the device. In their study on appointment scheduling, Starner *et al.* found that many participants who claimed to use a planner or PDA actually used scraps of paper in practice, because it was faster to retrieve the paper. Starner compared this effect with Miller's observation in 1968, that user efficiency dropped significantly when a computer's response time increased beyond two seconds [Miller 1968]. Our interview data support this argument. One birder hangs her paper notebook and pen around her neck (with string) alongside her binoculars. Upon sighting a bird, she drops her notebook and pen (so that they would hang in front of her) and picks up her binoculars to zoom in on the bird. This solution enables her to swap between tools fluidly.

With respect to flexible input, we see that biologists choose paper notebooks and datasheets because they allow for efficient yet flexible data capture. In the field, the biologist captures qualitative observations about the field site, and quantitative measures from her experiment. Back at the field station, she can record more data from lab experiments, or even insert printed visualizations or notes. Data capture can be structured; data entry sheets are used for their efficiency. However, pen and paper also afford unstructured input, as a biologist can scribble notes in the margin of a data sheet, or draw sketches on a page in her notebook. Thus, interactions with technology in the field should be quick and flexible; they should be no more difficult to use than the scientist's current tools.

3.2.3 Robustness

A key requirement of mobile technologies is their robustness. Electronics used by field scientists (such as GPS loggers) are ruggedized to withstand moisture and rough handling. We have seen evidence of rain, mud, and blood on data sheets, demonstrating the robustness of pen and paper as a capture technology.

Because paper is robust, biologists treat their notebooks as the *definitive record* of procedures, decisions, measurements, and results. One interviewee reported that she could reconstruct corrupt data by reviewing tables and procedures from her calculations. Another showed us how she could backtrack through experiments, to debug generations of *E. coli* culture to track an outlier's lineage. In one of our visits to the California Academy of Sciences (CAS), we were able to view field notes from Tracy Storer, written in 1925. We also recall Duguid's remarks on his studies of 18th century documents, that the old letters he read "may well be around in another 250 years," whereas the files and disks on which he recorded their text are "unlikely to last twenty five" [Brown and Duguid 2002]. The permanence of paper tools is reassuring.

FIGURE 3.3. Biologists collaborate both in the field and in the lab. In the field, they divide up responsibilities (*e.g.*, designating one person to write down measurements while the others collect specimens). In the lab, they take turns transcribing and analyzing data.

3.2.4 Collaboration

Current paper-based tools have unique tradeoffs for collaborative activity (see Figure 3.3). The main disadvantage of using pen and paper is that content is not easily shared between remote collaborators. Hand-drawn sketches must be photographed or faxed to the recipient. For example, the Museum of Vertebrate Zoology at UC Berkeley is currently digitizing the field notes of Joseph Grinnell, so that they can be shared on the Web [MVZ 2005].

We observed several types of collaboration. Some research projects involve multiple graduate student researchers. These biologists will work in small teams (many times in pairs) to set up field sites and collect data. Many research projects are organized in a pyramid fashion. The professor and the main doctoral student organize the experimental procedure, while undergraduates and volunteer researchers will help to carry out the details of the work.

In these collaborations, paper tools are shared between the participants. For example, we saw a calendar which a doctoral student and her undergraduate assistant used to coordinate their tasks. They planned out their field season with one calendar, and wrote down tasks they accomplished on another. The researcher described to us the ways in which the calendars were used:

> The monthly calendars were kept side-by-side on the wall at our base camp, which was the hub of all our work that summer.... Tasks were written on there

> every few hours or every day. As the season went on, and one field assistant finished and another started, the calendar helped convey the state of the project.

The physical presence of the calendars on the wall of the field station helped to focus the tasks of the collaborators.

The author also helped to enhance the workflow of the camera trapping project at Jasper Ridge. This project is directed by one biology professor, and is run by two post-doctoral researchers and a Ph.D. student. It involves many volunteers who fill in data sheets and help service the 24 cameras. To provide awareness of the project as a whole, the data sheets and photographs are all shared on the web (http://flickr.com/jrbpcameras). Collaborators can log on to see the product of their work. This effort opened up questions on how we could use digital pen interactions to provide better sharing, since photographing and uploading data sheets requires extra work by the volunteers.

3.2.5 Transformation

In one respect, paper notebooks only serve as temporary storage, since data that remains unanalyzed does not contribute to proving or disproving experimental hypotheses. We saw that biologists spent much time transcribing data from notebooks to spreadsheets, and correlating handwritten data with other media, such as photographs and sound recordings.

For example, Brandon spends six months out of every year studying bees in Costa Rica. At the end of each field day, his team identifies and measures bees, and writes down the data onto paper data sheets. They then collect these sheets together to enter the data into an Excel spreadsheet. Brandon uses the pivot tables feature to explore and help clean the data. Brandon can benefit from tools to help data entry or expose data lineage.

Jonathan, who studies birds in India, spends his field season identifying birds and capturing audio of the bird calls. When he returns to his office at Stanford, he needs to enter this data into his Microsoft Access database. To verify the identifies of birds (especially those entries he made with low confidence, marked with a question mark), he will need to extract the bird calls into sound files to send to the local expert in India. Jonathan can

benefit from software that automatically correlates his audio samples with his data sheet entries.

The Centrality of Paper Notebooks

The common thread through all of these observations was the centrality of paper notebooks in biology work practice. In the field, biologists use notebooks to capture observations of their experimental sites; these observations may lead to new hypotheses. This practice was shaped by Joseph Grinnell's work [MVZ 2005]; it emphasizes careful documentation with descriptions of the day's work, the time and date, weather, participants' names, and pictorial annotations such as maps.

Few of today's field biologists place as much detail into their handwritten notes, as observations are frequently complemented by digital photos and other streams of digitally captured data. For example, portable, inexpensive, low power, and reliable sensors such as the iButton [Embedded Data Systems 2005] have enabled environmental data collection in harsh situations, and the advent of battery-powered wireless sensor networks [Culler and Mulder 2004] offers even richer environmental monitoring.

There has also been recent interest in electronic systems specifically supporting biology research (*e.g.*, [Arnstein*, et al.* 2002, Butler 2005]). For example, CyberTracker is software for handheld computers that allows field assistants to enter observations [CyberTracker 2007]. By displaying icons for animals, it allows non-literate field assistants to participate in the animal tracking process. LeafView is a Tablet PC-based electronic field guide that supports identification of botanical species in the field [White*, et al.* 2007]. Currently, these tablet-based solutions work best in laboratories, where power outlets are plentiful, and an infrastructure is available to provide electronic communication and backup. PDA-based solutions are more suitable for data capture in the field, but still trail behind paper in flexibility and robustness. Thus, paper notebooks still remain the medium of choice outdoors. They are the central organizing artifact and *the* permanent record of work in the field and in the lab.

3.3 Feasibility of Digital Pens

To explore the feasibility of using digital pens in the field, we deployed Anoto digital pens (Nokia SU-1B) to an undergraduate research class in the Los Tuxtlas rainforest in Mexico (see Figure 3.4). In this study, 10 biologists (9 students and one professor) used digital pens and notebooks for nine days. We observed the use of the pens to learn how well they integrated into the field practices. However, we note that this deployment studied undergraduate researchers, and that professional biologists may differ in their patterns of collaboration and notebook usage. For example, we have spoken with a few biology researchers who prefer to use PDAs for field work, because it saves time on data entry.

The biology students were able to use these pens in the field, even though the pens were thicker and more cumbersome than standard pens. The pens helped the students digitize their handwritten notes. They were also able to distribute their handwritten notes to perform data transcription in parallel. The hardware also survived the rainforest. Notes were digitized successfully even when the notebook was moist.

FIGURE 3.4. We deployed digital pens (Nokia SU-1B) to 10 biologists working in the Los Tuxtlas rainforest in Mexico. We observed that the digital pens and notebooks integrated well into the biologists' work practices.

Data Transcription

We also experimented with a data transcription technique. We asked the nine undergraduates to 1) enter data directly from a paper notebook, and 2) enter data using our data transcription aid, which displays digital ink next to the point of entry (see Figure 3.5). The pilot study was a within-subjects design, and counterbalanced so that five of the students used the transcription aid first.

The digital aid provided no significant effect on the time to complete the overall task of transcribing data. However, the time spent was *shifted* from typing into navigation (between pages of data). If we can match the performance of flipping pages of a physical notebook, we can perhaps achieve a significant speed-up in data entry. This would be an interesting avenue for future work.

3.4 Design Implications

This field study provided insights on what types of tools we could build for biologists, which in turn informed the design of PaperToolkit. This section lists a few key observations.

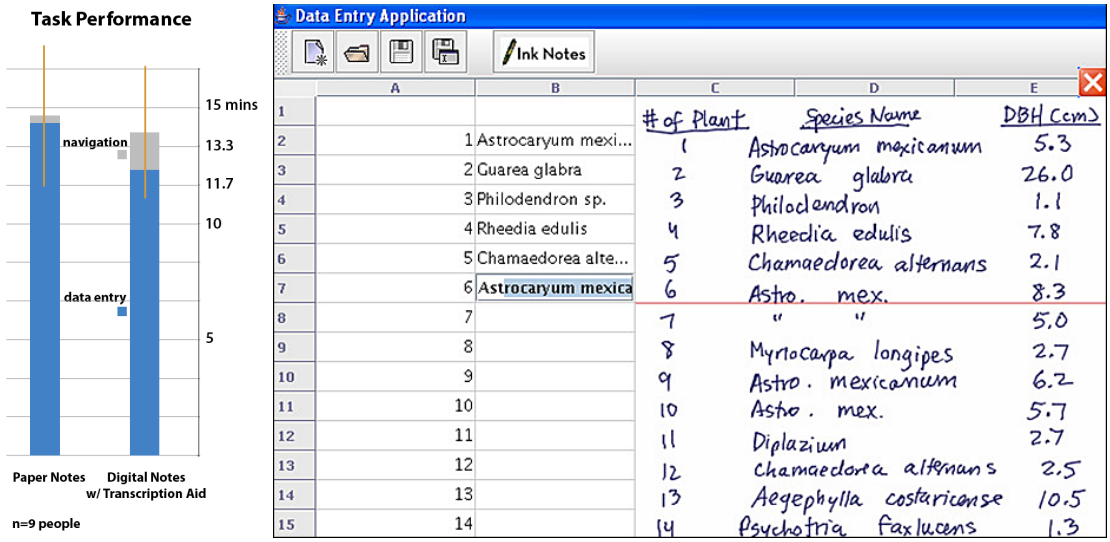


FIGURE 3.5. Compared to working with only a paper notebook, the transcription aid sped up data entry, but increased the time needed to navigate between pages. Specifically, flipping physical pages is faster than selecting a region of digitized notes to import into the smart spreadsheet. The difference was not significant, but inspired us to look further into this area. If we can reduce the time spent working with the GUI, we can achieve an overall speed-up.

Field and Lab Tools

The field tools should augment the current practice of using maps and notebooks in the field. Since field time is precious, paper applications should not slow down the workflow. Thus, support for batched pen interactions is important, as the biologist will not need to work with other digital devices in the field. These tools should degrade gracefully. If the pen fails to capture digitally, it should still lay ink down on the paper notebook (this is a nice property of Anoto pens).

Paper tools should aid biologists in their collaborative tasks. To support co-located collaboration, tools can capture ink input from multiple users, or present content on a surface large enough to support simultaneous viewing.

Toolkit Requirements

At minimum, the toolkit must allow the display and export of digital ink, to allow end users to view and share handwriting input. To support capture and access tools, the toolkit should provide abstractions for correlating ink input with other content. The toolkit can provide facilities to cluster ink input by time and space, and provide ways to query ink by timestamps. To support remote collaboration, the toolkit should support the transportation of ink and interactions across devices.

3.5 Generalizing the Findings to Other Domains

The findings in this chapter can also inform tool design for other domains. In particular, the requirements distilled by the field study can be collected into *personas*, which are hypothetical archetypes of actual users [Cooper 1999, Cooper 2007]. These personas do not exist, but represent actual biologists or groups of biologists (see Table 3.1). In general, personas help development teams focus the design of software features. For example, an

Persona	Description / Need
Amanda	College Student, 20 years old Docent and Volunteer Researcher at Jasper Ridge Draws pictures of birds and scenery to describe her field trips Capturing, Search, and Sharing of Notes and Sketches
Bruce	M.S. in Biological Sciences, 25 years old Field Assistant in Los Tuxtlas Captures media of all types (Photos, Notes, Audio, Sensor Data) Data Correlation for Better Retrieval
Carol	Ph.D. student, 26 years old Studies Plant-Animal Interactions in Los Tuxtlas Spends day taking measurements. Spends nights filling up spreadsheets. Transcription Aids for Efficient Data Entry and Analysis
Darren	Postdoctoral Fellow, 28 years old Directs a research project at the University Designs studies. Creates datasheets to support collection across his team. Printing Support for Data Sheets and Forms
Evelyn	Ph.D. in Biological Sciences, 27 years old Professional Biologist working with the California Academy of Sciences Captures leaf specimens in the field. Measures them back at the lab. Organize and Find Physical Specimens
Fred	Ph.D. student at Stanford, 25 years old Tracks small mammal activity. Carries a GPS device in the field, and enters all data into ArcGIS GPS Tagging of Notes and Printing of High Resolution Maps
Gwen	Professor, Biological Sciences, 42 years old Directs a team of 6 Ph.D. students studying small mammals Works with colleagues locally (on campus) and remotely in (Costa Rica). Sharing and Collaboration for Notes, Sketches, and other Media

TABLE 3.1 These personas demonstrate common needs identified in our observations.

augmented notebook designed for Bruce (a field assistant) may have different features than a notebook designed for Gwen (a professor) since she delegates much of her work and performs less data collection in the field. Here, the personas help to illustrate how our findings might extend to other domains.

Looking at the personas, we can identify qualities that map onto users in other disciplines. For example, any person who captures handwritten notes or drawings in a notebook can benefit from digitization, recognition, search, and sharing. Lawyers and office workers can benefit from handwriting recognition, to support later text search.

Any person who documents his workday through handwritten notes and photographs can benefit from a tool that offers improved search and retrieval for media. Designers and students may fit this characterization. Scientists and engineers who collaborate using geographic information systems (GIS) can benefit from augmented paper maps.

To confirm this, we conducted interviews with seven construction professionals—six architects and one contractor. This work revealed that in the early design stages, augmented paper tools can provide much benefit, supporting earlier work in investigating Tablet PC support for architects [Elliott and Hearst 2002]. Like our persona Gwen, architects currently use paper (and paper-like) surfaces for ideation and communication. Initial designs are sketches on paper. They are redrawn in CAD, and then printed onto large *whiteprints*. The whiteprints are tacked onto walls, where teams can gather to draw revisions, either directly on the prints, or on translucent films used as information overlays (*e.g.*, Mylar). We also found that the most senior architects do not know how to use CAD; they create on paper and hand off their designs to junior architects who input them into the system. Also, four architects work with Google Sketchup. They report that this early-stage 3D modeling program is more intuitive than AutoCAD, but lacks CAD's sophisticated capabilities.

Sketching benefits all types of designers (*e.g.*, see [Buxton 2007]). While this work studied architects, related research demonstrates the need for designers to work with paper-based information graphics of all sizes, spread across walls and tables [Bellotti and Rogers

1997, Klemmer, *et al.* 2001, Newman, *et al.* 2003]. More recent work has shown that it is possible to use digital pens and notebooks to support sharing between design students [Maldonado, *et al.* 2007].

We have also presented ideas for augmented paper to six attorneys across three law firms. Like Carol, lawyers can benefit from support for paper-based co-located collaboration. Many still do take notes on note pads; however, several of the lawyers confirm that junior associates are more comfortable working directly on laptops during meetings. It may be that the perceived cost of using paper tools is currently greater than the value of flexible input.

3.6 Summary

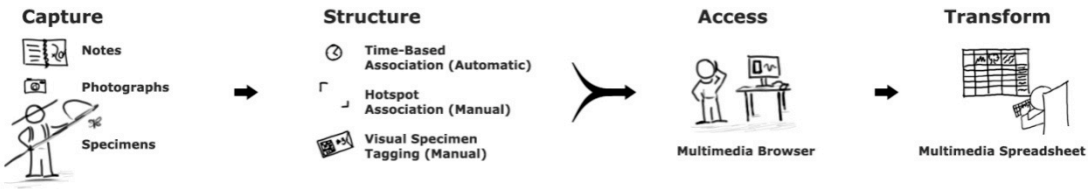
This chapter presented the existing practices of field biologists, and identified areas where technology could speed up or otherwise improve the workflow. For example, field tools need to be robust, as biologists work in sites that vary widely in temperature and humidity. Computer displays must also be readable in the sunlight. Current displays are low contrast, but paper displays work well outdoors.

We found that digital pens are suitable in their current form for field use. Further decreases in price, additional robustness and battery life, and better software will increase the value of this tool. Finally, we explained how to generalize the findings to other groups of users, such as designers.

For this dissertation work, we concentrated on the idea of mobile interfaces for capturing, structuring, and sharing. The next chapter explores one such interface, a smart notebook for rich capture and access.

4

Smart Notebooks



for Mobile Capture and Access

This chapter presents ButterflyNet, a capture and access system designed to meet the needs of field biologists outlined in CHAPTER 3. This system includes three components to enhance the workflow: a smart paper notebook and digital camera, a browser, and a multimedia spreadsheet. The interactions stem from the notebook and camera ensemble, which helps users capture and structure field content, such as photographs and notes. When the scientist returns to the lab, he can access his research content on a browser, and transcribe it using the multimedia spreadsheet. The ButterflyNet system was evaluated with 14 field biologists in a first-use study, conducted at the Jasper Ridge Biological Preserve (JRBP). The study found that users could integrate the automatic association and spreadsheet into their workflow. However, the manual association techniques show

promise for only some users, as they slow down field practice. The next section chronicles the design. We then present ButterflyNet's components, the evaluation, and lessons learned. The chapter concludes with the architecture and implications for the design of PaperToolkit.

4.1 Evolution of Designs

The field study revealed opportunities to augment the biologist's workflow with technology, including faster data entry, better data analysis, and sharing. We focused on three goals:

- Enable biologists to capture more data while in the field.
- Provide association and browsing techniques to enhance the retrieval of data back at the lab. Since biologists are now capturing more data, efficient access is vital.
- Support the transcription and transformation of content into analyzable forms.

Additionally, the interactions should build on existing practices, to increase the likelihood that a biologist would adopt the system. Thus, we decided to focus on the paper notebook. By recognizing the *centrality of notebooks* in current practice, the software system could allow users to be immediately familiar with its main interactions.

We brainstormed, sketched, and presented ideas to biologists. The ideas ranged from hybrid notebooks that combined paper with PDAs, through spreadsheets that tracked data

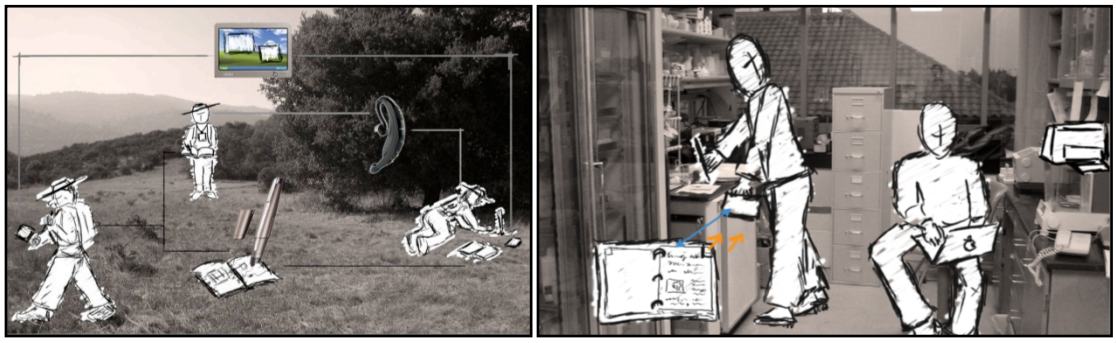


FIGURE 4.1. Two early concept sketches. *Left*) MultiMOBILE provides multiple means of interacting with data while in the field, including digital pens and notebooks, handhelds, and Bluetooth ear pieces. *Right*) Pen gestures allow biologists to rapidly send content from a paper notebook to printers and laptops in the vicinity.

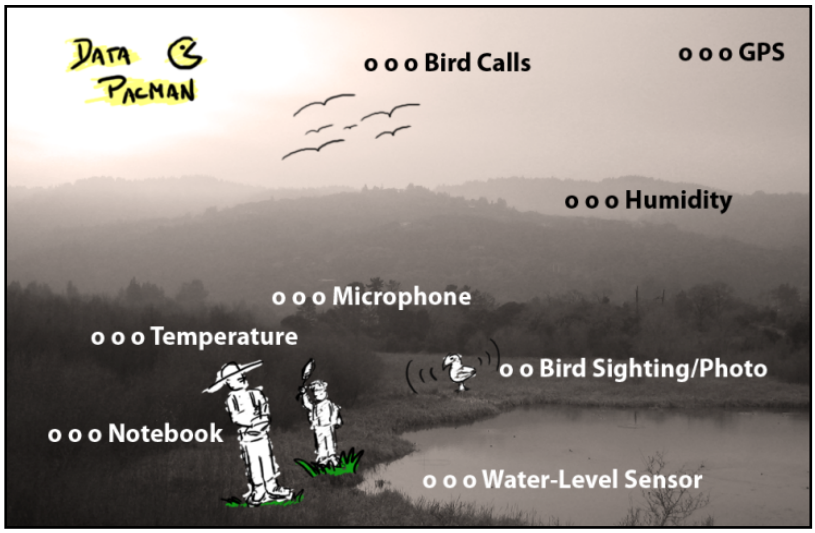


FIGURE 4.2. The early sketch that inspired ButterflyNet. The Data PACMAN (Personal Automated Capture Mechanisms for Augmented Notebooks) captures and correlates multiple streams of data for field biologists, including handwritten notes, bird calls, and GPS logs.

lineage, to audio interfaces that would record and process voice commands (see Figure 4.1). MULTIMOBILE and GESTURES were two concepts that explored the idea of device ensembles—how to manage content and interactions across multiple devices, including paper notebooks, handheld computers, audio earpieces, laptops, and printers. Data PACMAN was a third idea, which targeted automatic correlation of multiple media streams (see Figure 4.2). These ideas were presented to our collaborators in the biology department. From the feedback, the paper notebook with automatic correlation seemed most promising. We fleshed out this idea, and created a working digital pen and notebook prototype which we deployed to the Los Tuxtlas research class (detailed in SECTION 3.3).

The Los Tuxtlas deployment found that the smart notebook had the potential to improve how biologists captured and accessed research content, without drastically modifying existing practices. Following the iterative design process, we created additional prototypes, including sketches, screenshot mockups, and working programs. The Los Tuxtlas working prototype demonstrated the laboratory retrieval of notes. Building on this idea, a subsequent screenshot prototype explored the idea of a rich multimedia interface for accessing field notes (see Figure 4.3, right). These prototypes were presented at meetings

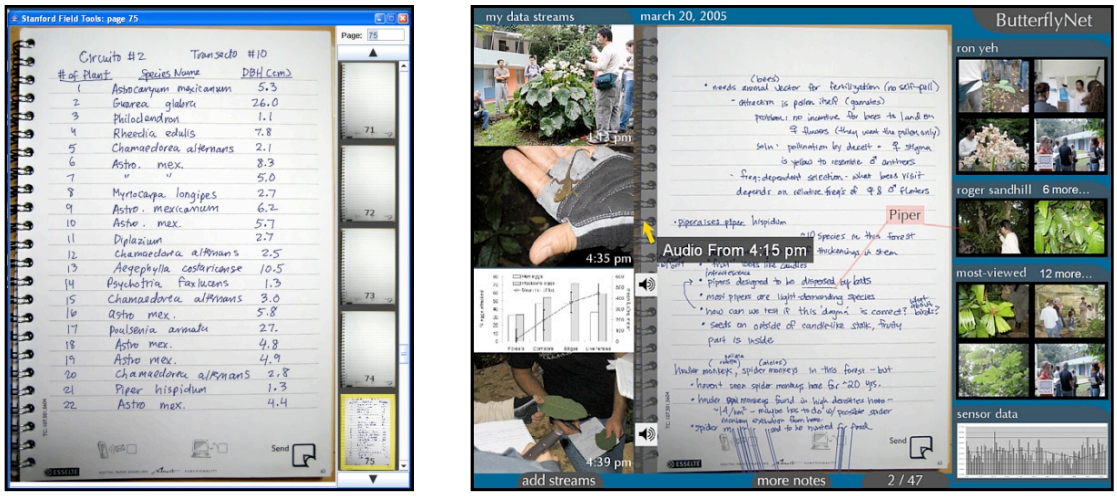


FIGURE 4.3. *Left*) The first working prototype allowed users to browse notes captured in the field. This was tested in Los Tuxtlas. *Right*) A subsequent screenshot mockup helped us communicate ideas to biologists and our colleagues in HCI research.

with biologists. We also discussed them at length with a doctoral student in biology (our primary liaison with the Biology department). After multiple iterations, we had fleshed out the basic features of the smart notebook. We also renamed Data PACMAN to ButterflyNet.

The prototypes enabled rapid exploration of many ideas, including enhanced data transcription, data lineage, querying sensor data, and sharing. For example, one Adobe Flash prototype explored how multiple users might log in to view notebook pages shared to the research group's website (see Figure 4.4). We implemented two of the ideas—the smart

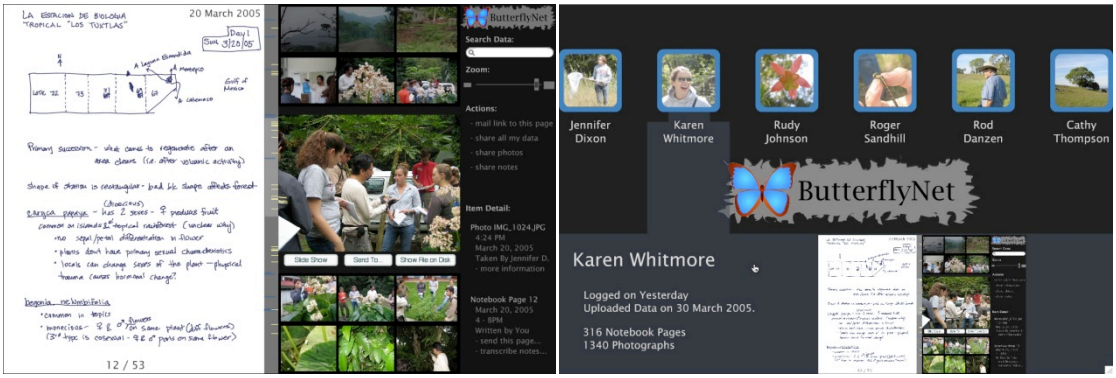


FIGURE 4.4. An interactive prototype of ButterflyNet. *Left*) The main view shows a page of notes with temporally-related photographs in a side panel. As the user navigates the digital notebook using the scrollbar, photographs will appear or disappear depending on how closely they relate to the visible page. *Right*) Biologists can view each others' notebooks.

notebook for correlating media, and the spreadsheet for structuring research content.

4.2 Notebook for Capture and Access

A central ubiquitous computing theme has been techniques for capturing and accessing information [Abowd and Mynatt 2000]. Filochat [Whittaker, *et al.* 1994] does this for personal and shared notes. PARC has contributed tools for capture and access of group meetings [Minneman, *et al.* 1995, Moran, *et al.* 1997]. eClass showed that capture and access can be effective in classroom lectures, where the professor's actions are captured and made available to students over the web [Truong, *et al.* 1999]. Integration of free-form ink with other streams of input such as audio and video has been central to these systems (see *e.g.*, [Kam, *et al.* 2005, Lamming and Flynn 1994]). This inspired the automatic media correlation feature of ButterflyNet (*e.g.*, by time).

ButterflyNet extends previous work by providing association by other metadata facets

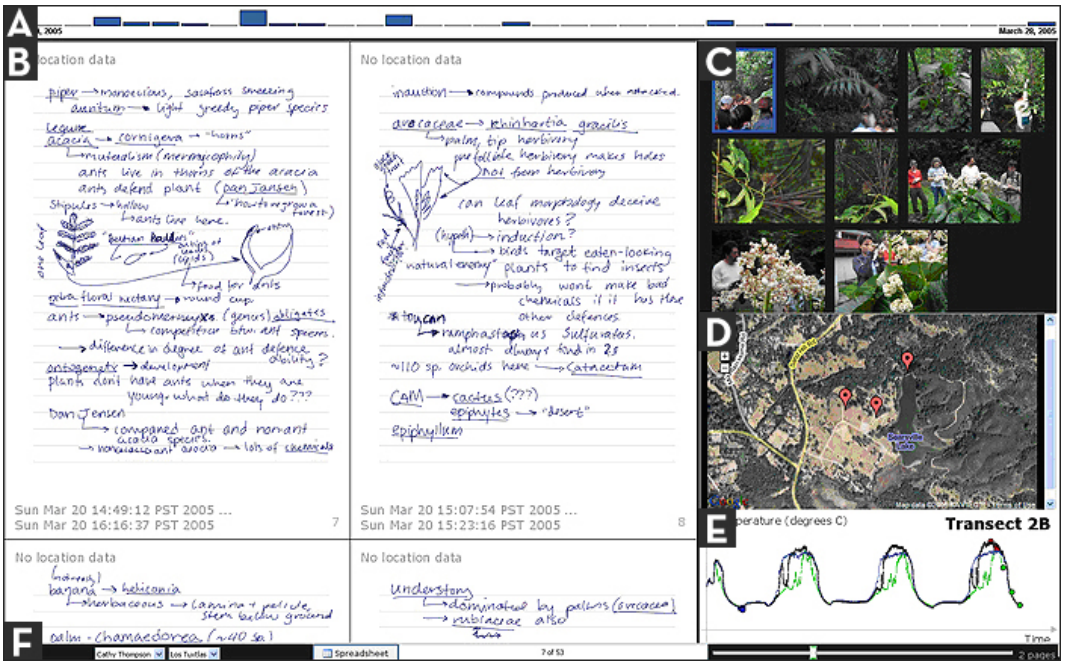


FIGURE 4.5. The ButterflyNet browser enables users to efficiently access field notes and related content. The notebook provides automatic and manual techniques to associate media. The browser provides A) a timeline visualization of captured notes, B) a main panel presenting the digitized handwriting, C) related photographs in the context panel, D) uploaded GPS data, and E) related data logged from a sensor net.

(*e.g.*, location). It supports the capture of handwritten notes, digital media (*e.g.*, photographs), and physical specimens. The capture interface consists of the digital pen, paper notebook, and digital camera. The access interface comprises the ButterflyNet browser software (see Figure 4.5). In the field, a biologist writes observations and measurements into his notebook. The pen saves the location and timestamp of each stroke to its memory. To capture photographs, the biologist uses a digital camera. This smart camera timestamps captured photos, and communicates with the pen wirelessly over Bluetooth.

When the biologist returns to the field station, he connects the pen and camera to a PC to upload his notes and photographs. The ButterflyNet browser then allows him to browse this collection. As he navigates his notes, the context panel automatically presents the photos most closely related (in time) to the visible page (Figure 4.5C). For example, if the user views notes from 3:23PM on March 23, 2005, he will see photographs taken on or near that time in the context panel. The timeline visualization (Figure 4.5A) enables a user to jump to content by date/time. The height of each bar represents the number of note pages at that time interval. The navigation bar (Figure 4.5F) lists notebooks from collaborators; the user can view them by selecting from a dropdown menu. The slider controls the zoom level; the browser can show one page at a time or scale up to four across.

The algorithm that correlates photographs with note pages was chosen for its simplicity. It selects the earliest timestamp and the latest timestamp from the strokes contained on the page, and returns the first twenty photographs that fall into that range (as determined by a database query). For details on alternative algorithms we considered, see Appendix B.

The key component of this system is the digital pen, which captures notes both physically (with ink) and digitally. If the pen's digitizer were to fail, users would still be able to record observations, as the paper and inking pen provide redundancy. Conversely, if a physical notebook were lost or otherwise unavailable, the electronic version can be used. ButterflyNet offers graceful degradation: it is never worse than current practice. The smart notebook metaphor extends ideas in previous work, such as AudioNotebook [Stifelman, *et*

al. 2001]. This class of interfaces augments existing practice by preserving informal note capture, and enhances data retrieval through automatic and manual association of content.

Manual Association by Pen Gesture

While automatic association provides benefits without any extra work, there are scenarios when a scientist might rather associate a photograph to a particular location in his notes. For example, one biologist reported that he always captures an overview photograph of his field sites on each visit. If this photo were more tightly coupled with his notes, he could view it to help him recall field conditions when he reviews his handwritten notes. Currently, the photo sits in a folder on his computer, separate from his notes.

ButterflyNet addresses this problem through a manual association technique (see Figure 4.6). If the biologist wants to designate a photo of interest, he can navigate to it and then mark a gesture in his notebook to provide a location for that photograph. The scientist first selects the digital content by using his camera to capture or browse to a photo. Then, he draws a bracket gesture (*e.g.*, ⌜ ⌟) into his notebook. The chosen content is now linked to that location in his notebook. The smart camera emits a beep to confirm that the pen gesture has been recognized. The audio feedback is an important feature, as users may not be able to look at the camera while working in the field.

Back at the lab, the browser renders the photograph within the bounds of the bracket gesture. This interaction extends prior work (*e.g.*, [Dymetman and Copperman 1998, Klemmer, *et al.* 2003]) by enabling end users to *author associations* on the fly. To provide the rich interaction, we prototyped the smart camera using a handheld PC with a web camera

FIGURE 4.6. To associate a photograph to a page in the notebook, the user A) captures or browses to a photo, then B) draws a bracket gesture ⌜ ⌟ into the notebook with the pen. C) The smart camera provides real-time visual and audio feedback to confirm the association. D) The browser renders the photo within the bracketed, in line with the digitized notes.

affixed to the back. The camera communicates wirelessly with the pen, offering the real-time feedback for in-the-field interactions. Given current technology trends, we anticipate that a production implementation would use a camera phone with software that talks to the pen.

Manual Association of Physical Specimens

A second technique enables the user to associate multiple photographs to a *physical specimen*. The scientist places his specimen in a special envelope augmented with a 2D barcode and patterned paper (see Figure 4.7). To associate a photograph with this specimen (*e.g.*, "I found lichen near this pond") he takes a picture with the barcode in the view of the camera. When the biologist uploads the photographs to his PC, ButterflyNet will recognize the barcode and establish the association. Thus, photos containing this barcode are *linked* to notes written on the envelope and the specimen contained within. The augmented envelope can later be used with a retrieval system (*e.g.*, barcode scanner) to find the related photos.

This technique aligns with the existing practice of using envelopes to store physical specimens. Additionally, the augmented envelopes can be created by end users with printers; the barcodes can be read with off-the-shelf scanners or web cameras. And finally, the envelope includes a human-readable ID. If the barcode recognition fails, a human can establish the associations visually.

FIGURE 4.7. Augmented specimen envelopes enable a biologist to associate photographs to a physical specimen and its annotations. The biologist uses an envelope enhanced with a 2D barcode. He A) takes a photo with the barcode in view, B) writes an annotation on the envelope, and C) places the specimen in the envelope. D) ButterflyNet detects the barcode and establishes the association. Photos can be retrieved in the browser either by typing in the human-readable ID, or by scanning the barcode.

Tangible Interactions for Content Retrieval

ButterflyNet also enables users to access *digital* research data by using their *physical* notebook. With this technique, a user taps a notebook page with his pen. The browser responds by presenting the digital version of that page and all associated media; when used with the smart camera, only the closest photos are displayed. This device ensemble approach (using the notebook to control the PC or handheld) enables users to work with the best interface for each task. Physical notebooks are easier to browse, whereas digital displays enable zooming for graphics. It is especially valuable for in-the-field retrieval, because screen real estate is intrinsically limited for the mobile PCs.

4.3 Spreadsheet for Transcribing and Structuring Data

Finally, ButterflyNet enhances data transformation through the multimedia spreadsheet, which contributes novel organization and visualization techniques (see Figure 4.8). The multimedia spreadsheet was inspired by earlier use of rich visualizations in spreadsheets (*e.g.*, [Chi, *et al.* 1998, Levoy 1994]), and helps a biologist transcribe tabular data. In the browser, the biologist first selects a region of notes to transcribe (by dragging the mouse cursor). The selected handwriting is imported into a spreadsheet as an image, scaled so that the rows of data align with the rows of the spreadsheet. As the biologist types, a placeholder moves down the page to help him visually keep track of the row he is transcribing, eliminating the need to look back and forth between a physical notebook and the computer display.

ButterflyNet also provides a physical interface to import handwriting into the spreadsheet. While the spreadsheet is in focus, the user can draw a bracket gesture ⌜ ⌟ around handwritten data in his notebook. ButterflyNet recognizes this pen gesture, extracts the selection from the corresponding digital notes, and exports it to the spreadsheet.

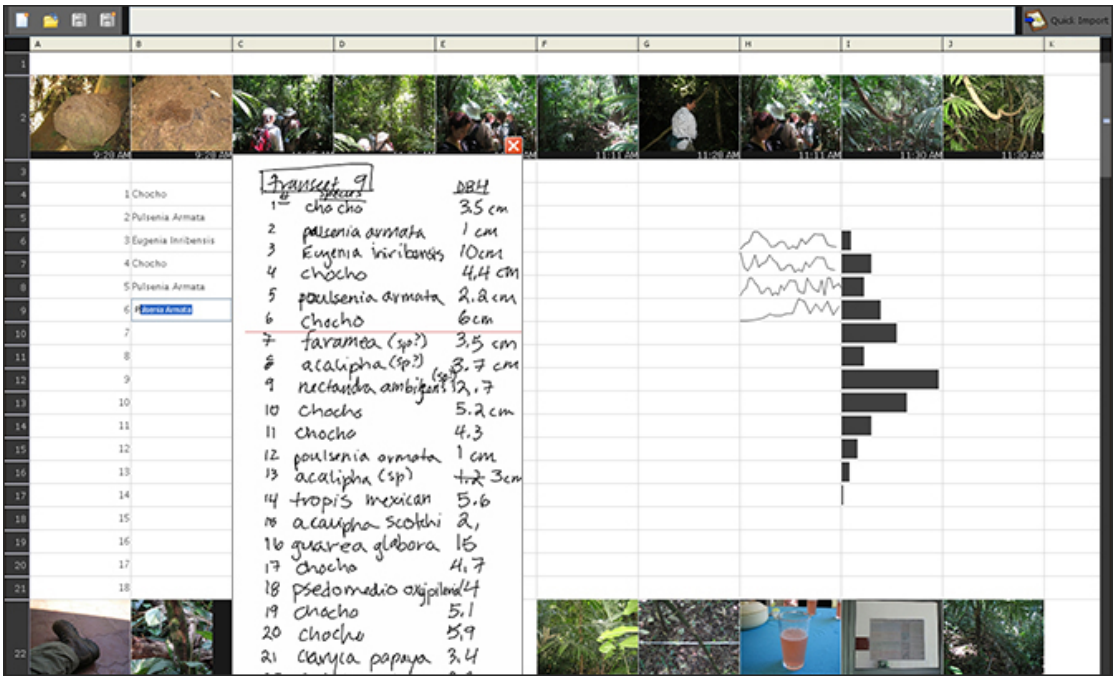


FIGURE 4.8. The multimedia spreadsheet assists with the transformation of field data. A floating window displays digital ink while a red line marker visually tracks the row a user is currently transcribing. To provide context to the data, one can assign images, time series line graphs, and percentage bars to individual cells. Photographs can be chosen from a drop-down menu, which displays photographs from the browser's context panel.

In Excel, images and charts cannot be placed in cells; they float over the sheet. ButterflyNet's multimedia spreadsheet enables users to embed these visual objects into individual cells. As visualizations are now first-class objects, they can be used to annotate rows or columns of data. For example, a biologist who records measurements of specimens can benefit from embedding a photo of the specimen into each row of measurements. To import an image, the biologist right-clicks a cell, which opens the quick-import menu. This menu displays thumbnails of the photographs displayed in the ButterflyNet browser, and is automatically updated whenever the context panel changes.

4.4 Evaluation

We conducted a first-use study of ButterflyNet, focusing on interactions with three data types: photos, notes, and specimens. The study attempted to shed light on three hypotheses:

H1 The field capture techniques (digital pen, automatic and manual association) enable structuring of content with *minimal overhead*.

H2 The browser presents an *efficient and rich* information view by presenting photographs in a context panel and in line with the notes.

H3 The multimedia spreadsheet *speeds up* the transformation of field data.

Since the ButterflyNet interactions were inspired by field biology tasks, we recruited field biologists to use the interface. The 14 participants (six male; eight female) included JRBP docents, Ph.D. students in biology, and professional researchers. Field experience ranged from none (for one docent), to several years (for Ph.D. students), up to a maximum of 18 years (for one professional). Five of the 14 had at least 10 years of experience in the field. Each session lasted 2.5 hours, and was held in and around the field station at Jasper Ridge. We compensated each participant $45 cash for their time.

First, participants used the ButterflyNet capture devices to work in the field (see Figure 4.9) and collect photos, notes, and specimens. Second, the biologist used that day's data to create a spreadsheet. Specifically, the task was modeled after a field research method where biologists capture measurements of specimens along line transects, as we witnessed in Los Tuxtlas (see SECTION 3.3). In the field, biologists used three techniques:

- For each oak gall they found along a 2×40-meter line transect, they recorded the

FIGURE 4.9. *Left*) A participant uses the smart camera to take a photograph of an oak gall. She carries a pen and notebook to capture measurements along the line transect. *Right*) The task was informed by our observations of existing practice in Los Tuxtlas.

distance of the gall along the transect, its size, its color, and a photograph. The measurements were written into the notebook.

- At three points along the transect, they photographed the habitat using the smart camera, and manually associated it with the bracket gesture into their notebook.
- At three other points along the transect, they photographed, annotated, and collected a physical specimen of their choice using the barcoded envelopes.

These three tasks mirror everyday field tasks—collecting measurements, photographs, and specimens. The participant carried a backpack (with water and equipment), a field notebook and digital pen, a digital camera (Canon SD300), the smart camera, and barcode-tagged envelopes. Participants carried two cameras because at the time of the study, the bracket gesture required the smart camera's programmability, while the consumer digital camera yielded better photographs, which also allowed for recognition of the barcodes. We envision that future cameras (or camera phones) will provide the smart camera's programmability. After about an hour of field work, the biologist returned to the field station. The first half of the experiment concluded with a 15-question survey of the biologist's background and opinions on the field capture methods.

Next, the participant engaged in a lab task (see Figure 4.10). The user was asked to browse the photographs and digitized notes, and create a spreadsheet with transcribed

FIGURE 4.10. *Left*) In the lab portion of the study, participants browsed the correlated media, and transcribed data to the multimedia spreadsheet. *Right*) This was inspired by the data entry sessions we observed at the end of each field day in Los Tuxtlas.

measurements and imported photographs (to share with collaborators). The participants were shown the export-to-spreadsheet method for aided transcription, but were not required to use it. The lab task ended with a second 15-question survey to assess the lab tools and ButterflyNet in general. Finally, we conducted a debriefing interview. Other than this interview and the tutorials of the ButterflyNet tools, participants completed tasks on their own, while experimenters captured observations in photos, video, and handwritten notes.

4.5 Results

This section presents outcomes from the user study, organized around ButterflyNet's key features. We refer to specific observations, questionnaire results, free-form responses, and hypotheses where appropriate.

Media Association

Participants readily understood the concept of time-based automatic association. At the time of the study, ButterflyNet associated media at a fixed (and coarse) granularity—the context panel showed photographs captured within the time span of the current page. Unfortunately, many users recorded multiple measurements and photographs per page of notes, and sometimes desired finer associations than ButterflyNet provided. For example, there might be several photos of galls all at 1:24PM. To determine an exact association for the spreadsheet task, some participants would find anchor images in the browser (*e.g.*, the dark and small outlier gall at 1:24PM), and then interpolate the rest. For example, the subsequent photos must be associated with the measurements immediately after the small and dark gall's. We conclude that capture and access systems need to provide the user a way to adjust the granularity of automatic associations, and tools to visualize the associations.

Participants were excited about the possibilities presented by the bracket gesture for manually associating photographs to parts of a page. People mastered the gesture quickly. One participant found it efficient enough to draw in every row of measurements, to achieve a one-to-one association between data and photographs. During the debrief interview, another user mentioned that ButterflyNet, with its manual and automatic association techniques, would work perfectly as her travel journal; this comment points toward general applicability of these techniques. Sample data from the study can be seen in Figure 4.11.

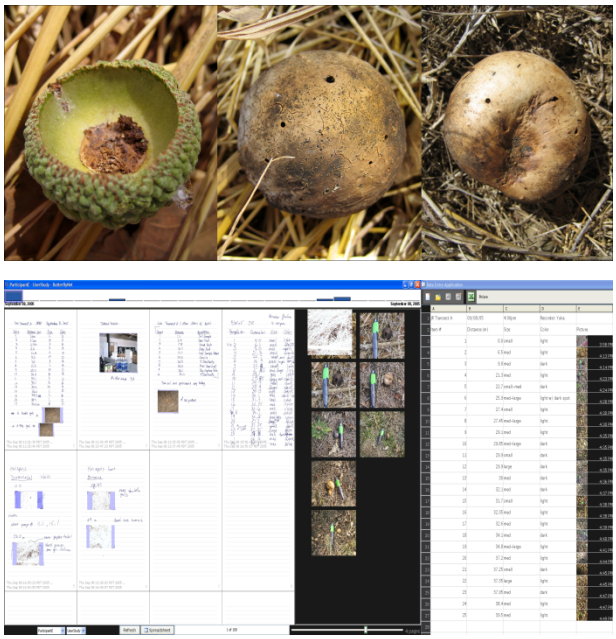


FIGURE 4.11. *Top*) Three photographs taken by one participant. *Bottom*) An overview of the browser and spreadsheet from a second participant, who has zoomed out to view

Study participants were also able to quickly learn how to use the barcoded envelopes for collecting specimens. We did notice, however, that the visual tag would occasionally not be recognized (*e.g.*, due to tall grasses occluding parts of the barcode). Fortunately, the biologist would still be able to associate the photographs to his specimens, since the visual tag includes a human-readable number (see Figure 4.7D).

Biologists Favor Automatic Association

Figure 4.12 shows the median participant response to the association technique questions. These results contribute to supporting H1, that the capture techniques can be useful while adding *minimal overhead*. Users felt that automatic association would not increase field time, and were positive toward the technique's potential value. Through automatic

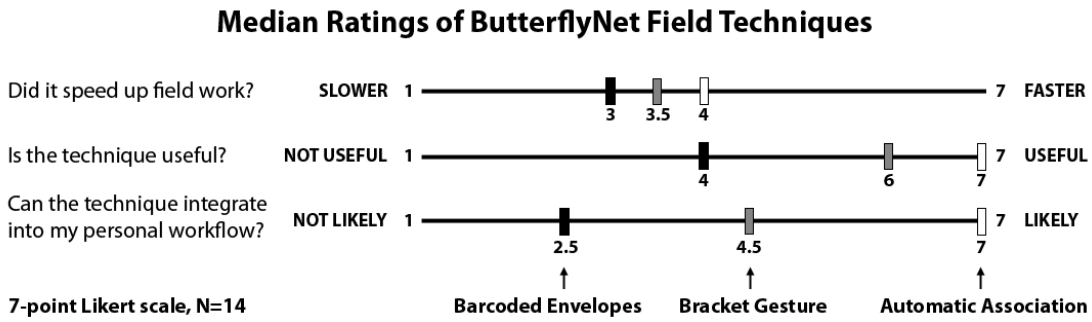


FIGURE 4.12. Participants found the automatic association technique to provide great value with little overhead. The bracket gesture showed promise, as it is useful but increases field time slightly. The specimen envelopes may suit only some biologists.

association, ButterflyNet offers an informal UI, such that the in-the-field focus—when time is precious—is on *content capture*, rather than interface manipulation. And though flexibility over the time window for correlation would improve the experience, the system was already providing value beyond today's makeshift solutions.

However, the data also show that participants felt the manual association techniques slightly increased field time, and felt that the specimen envelopes would have to improve before they would be integrated into current work practices. This response may have a few explanations. First, biologists may be reluctant to use tools that increase field time by *any* amount, even if the technique promises to provide benefits later on. Second, not all of our subjects collect specimens (or even photos) in their work, and thus have no use for the envelopes or bracket gesture.

Since, the version in the study addressed only photo-centric tasks, we partitioned the users by how much they value photos. In this case, opinions about the association techniques diverged significantly. In all cases, the likelihood that the subject would use a technique ranked higher for those who valued photos, showing that biologists who use photos daily are excited about ButterflyNet's potential. For instance, when asked if she would use the field tools in her work, a veteran of more than 10 years responded with straight 7's (the highest ranking). She takes 10-20 photos per field day, and views photographs as extremely important (7 out of 7). She stated that the ability to find and associate photos with spreadsheet cells was perfect for her work with measurements and photos of animal teeth.

Information Access and Transformation

Participants readily understood the browser's presentation and access interface. In the questionnaire, we determine the perceived usefulness of the interface. Figure 4.13 summarizes the responses for 14 aspects of the system, measured on a 1-to-7 Likert scale (1 for *very unlikely*; 7 for *very likely*). The top advantages users reported were that ButterflyNet would aid capture and transcription of more data (*speeds transcription* and *increases capture volume*). Additionally, it would help them recall experiments better. These two support H2, that the browser provides rich information access.

The data indicates that ButterflyNet helps data transformation (H3) and integrates with biologists' current practices. All users completed the lab tasks, and felt overall that the transcription aid would speed up data entry. The tools integrate well with current practice: 12 of 14 users reported regular spreadsheet use (7 out of 7 on the Likert scale). In the free-form responses, eight mentioned that they liked the association of photos with notes. Six liked the tools for exporting data. Six wrote that digital backup of notes would be invaluable.

Graceful Degradation

Paper-based tools are resilient to catastrophic failures. Very occasionally, the digital pen

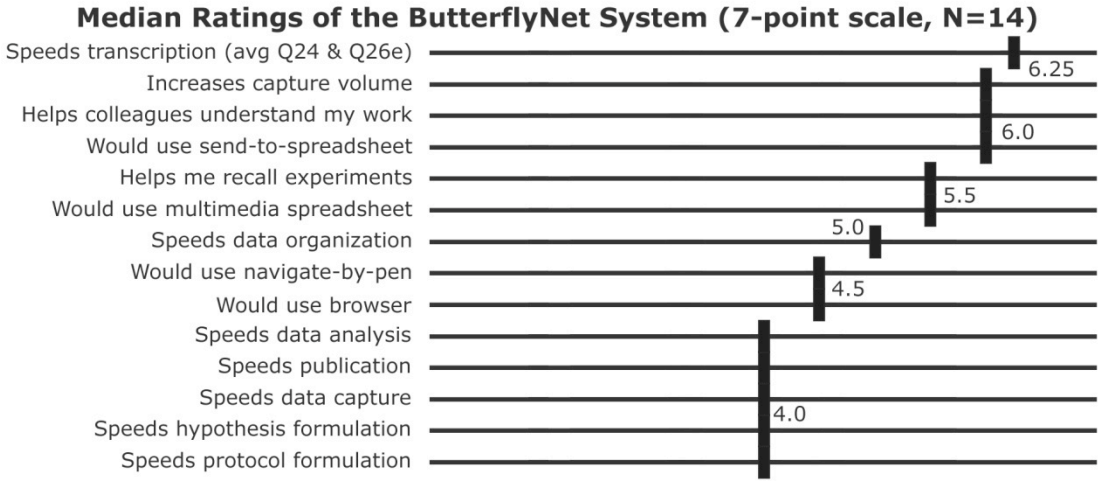


FIGURE 4.13. Biologists felt that ButterflyNet would help them transcribe data faster. Additionally, it could help them capture heterogeneous information (*e.g.*, photos and notes).

would miss a few letters or numbers in participants' handwriting—due to dirt on the page, or the printer we used to generate the dot pattern. The few users who encountered missing data in their digitized notes quickly switched to their physical notebook, where the data was faithfully captured with actual ink. These users seemed comfortable switching to the physical notebook when the digital representation was incomplete.

Gesture Recognition

Since our fieldwork found that biologists' value field time, we designed the pen gesture to be as lightweight as possible. The gesture engine we used, PapierCraft [Liao*, et al.* 2005], normally requires a pigtail loop ( ) at the end of all gestures to enhance recognition rates. We removed this to achieve lighter-weight gestures. Additionally, ButterflyNet does not include a way to switch into *gesture mode*, so the system must inspect all strokes. This design achieves simpler field interaction, but results in more recognition errors (false positives, because users may draw boxes that look similar to the bracket gesture). One solution would be to allow users to delete false positives or create manual associations after the fact (back at the lab).

The recognition of bracket gestures worked reasonably well. Of the 69 brackets drawn, 54 were correctly recognized (recall = 78.3%). Of the 61 brackets found by the system, 7 were false positives (precision = 88.5%). However, most errors arose from a single participant's data, whose brackets were smaller than the defined threshold (a heuristic to prune the list of strokes sent to the recognizer). Without this user's data, the recall and precision rates were 93.3% and 91.3%, respectively.

Limitations

The questionnaire feedback pointed to limitations. First, participants felt that the transcription aid was valuable, but wanted a handwriting recognition engine to automatically transcribe all measurements. Second, a few participants voiced concerns about the need to use a special pen; they were worried they might lose it in the field.

We found that experts *who use photos* find the system useful, but many of the experts did not currently use photographs in their work. For example, one expert who gave low ratings studies bat calls and takes zero pictures per day. When we described in the debriefing that future versions could support audio, he agreed that then, ButterflyNet would prove extremely valuable to him.

Finally, as always, there are limitations with self report data. While the ratings generally support ButterflyNet's lightweight interactions (H1), rich information view (H2), and efficient transformation of data (H3), we must remember that each session took no more than 2.5 hours. Longitudinal evaluations would provide additional insight.

4.6 Architecture

The ButterflyNet software architecture can teach us lessons for PaperToolkit's design. Handwritten input comes from the digital pen (used in batched mode). When a biologist plugs his pen into his computer through the USB cradle, all notes written since the last synchronization are saved as XML files (one per page) in the user's data directory. This is handled by the *Pen Synchronization* module (see Figure 4.14).

When the user loads the browser, ButterflyNet reads in the newest note pages (XML) and photographs (JPEG) from the user's data directory. A local database is populated with the items and metadata (*e.g.*, timestamps of photographs). This *Database* supports search queries from the *Browser* module, which contains the GUI classes (*e.g.*, for rendering photographs in the context panel). As the biologist navigates the GUI of the *Browser*, pages are loaded from the *Notebook* module. ButterflyNet detects the timestamps and sends a query to the *Database*. The retrieved content (*e.g.*, photos) are displayed in a context panel. When the user wants to transcribe data, he interacts with the *Spreadsheet*, which receives photographs and ink from the browser. These interactions are summarized in Figure 4.14.

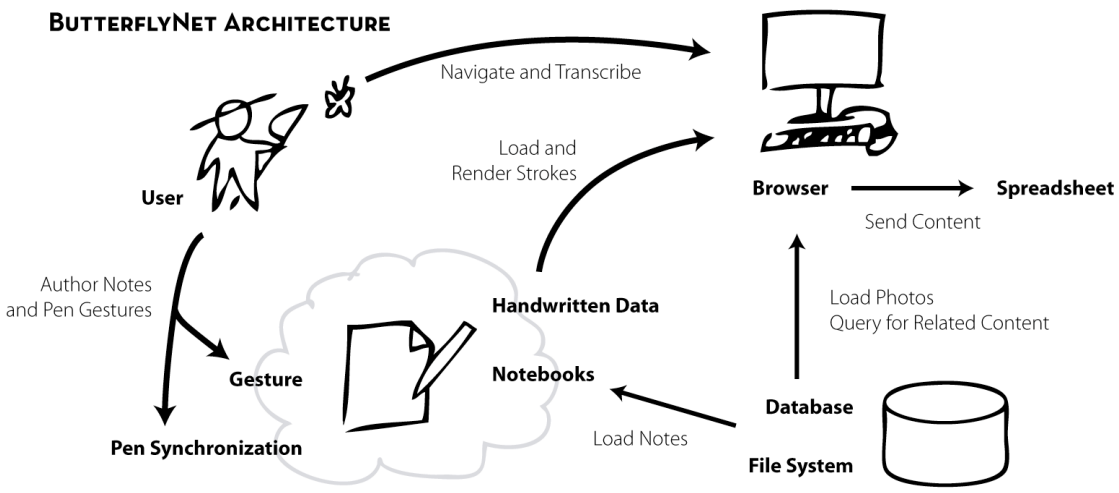


FIGURE 4.14. The ButterflyNet software architecture. The modules that handle pen input, gesture recognition, and modeling/rendering of ink strokes and notebook pages comprise 30 Java classes. The pen-centric functionality can be reused in other paper + digital applications.

To allow users to page back and forth in the notes, the system must be able to model and render pages of ink strokes. The *HandwrittenData* module contains ink strokes (arrays of X and Y coordinates with timestamps) and groups them into notebook pages. It supports the rendering of handwriting to GUI components in the browser, and allows export to image files (*e.g.*, JPEG). It also supports import/export of handwriting from/to XML files.

To display the handwriting correctly, ButterflyNet needs information about individual notebooks (since each Anoto notebook has unique coordinates and page offsets). The *Notebooks* module provides information about each notebook (*e.g.*, coordinates to help the system interpret incoming batched or real-time ink strokes). For example, the notebook used in the study was A5-sized, and contained 27 horizontal rulings per page. The first page was on 21.3.23.40 (an Anoto page address), and is offset by –114 pattern dots in X and Y. Thus, if an incoming ink stroke begins at location (X=120, Y=125) on page 21.3.23.41, it is rendered at location (6, 11) on page 1 in the browser.

The final module handles real-time gestures. The *Gesture* module is deployed on the PC and the smart camera. In the field, if the biologist draws the bracket gesture, this module recognizes the strokes and logs an event. The log is later uploaded to the browser, which renders the manually associated photo. This module also handles the real-time pen gestures

for controlling the ButterflyNet browser. For example, tapping on a physical page will open the browser to the correct digital page.

4.7 Implementation

ButterflyNet was written in Java (J2SE 5.0). It includes a small Microsoft .NET component that handles the pen synchronization; this monitor communicates with the Anoto SDK (version 3.1). ButterflyNet was tested with the Nokia SU-1B and Logitech 102 pens. The field notebook used in the study was designed in Microsoft Word and rendered to PDF. The resulting document was augmented with dot pattern with the Anoto Forms (FDK) plug-in for Acrobat Professional 7.0. The extra step of using the FDK is cumbersome, so we later designed PaperToolkit to automatically overlay dot pattern onto the paper interface.

The browser and spreadsheet are Swing applications. Handwritten notes are rendered with Piccolo [Bederson, *et al.* 2004], which allows the browser to animate and zoom pages. Photos are rendered with Java Advanced Imaging (JAI) and Piccolo. Pen data and photographs are stored on the file system (a user-specified data directory), and indexed by an HSQLDB database [HSQLDB 2005].

We encountered performance and memory-usage issues when loading and rendering pages of ink strokes (the pen captures at 75Hz, resulting in many samples per page). Performance was also a problem when loading large numbers of image files. To address these issues, we implemented pre-fetching and caching for note pages and photographs. Additionally, photographs were automatically processed into thumbnails on import.

The smart camera was prototyped with an OQO handheld PC [OQO 2006] running Windows XP. Real-time pen interactions were captured with the Zoom 4320 Bluetooth dongle and the Widcomm stack (version 5.0.1.801). From testing different combinations of Bluetooth software stacks and dongles, we found this pair to be the most stable for working with the Nokia SU-1B pens. Papier-Mâché [Klemmer 2004] was used to acquire images through a webcam attached to the OQO. The bracket gesture was recognized with the PapierCraft command infrastructure [Liao, *et al.* 2005]. For the augmented specimen

envelopes, we used the ClearImage SDK to recognize the Data Matrix 2D barcodes. ButterflyNet's map view was built using Google Maps, and sensor data was captured from Crossbow motes [Crossbow Technology 2005] running TinyOS [Levis and Culler 2002].

4.8 Extensions and Opportunities for Future Work

ButterflyNet was architected with extensibility in mind; it provides database tables for multiple types of content (*e.g.*, documents, video, audio), and also supports different metadata facets (*e.g.*, location, tags). In particular, we have experimented with display of GPS logs and sensor data (temperature and light levels). If notes contain location information, the map view will show visual markers depicting the relevant locations. If sensor readings were logged at the same time as the captured notes, they will also appear in the context panel.

The results from the ButterflyNet project point toward some exciting opportunities. One important step will be to study how biologists can use automatic and manual correlation for data outside of photographs and notes. The version deployed for the first-use study did not include the GPS or sensor data features. The freeform responses revealed that while participants liked the integration of photos, GPS integration would also prove extremely valuable. GPS support would allow us to associate media based on location information. Future interfaces should also explore interfaces for visualizing and accessing continuous-time media such as audio and video.

One aspect that ButterflyNet did not explore in-depth was handwriting recognition. From our tests with the Microsoft Tablet PC recognizer, we decided that handwriting recognition was not yet good enough to automatically translate field notes. However, it can provide value by automatically tagging notes to support text search. The recognition results can be stored in a hidden information layer; the notes would still look handwritten, even though they represent text strings (*e.g.*, see [Phelps and Wilensky 2001]). Future work can explore methods to present recognition results, or enhance retrieval and correlation by automatically creating hypertext links between pages.

While the bracket gesture only works for photographs, there is no reason why it cannot be generalized. As long as a device can record the timestamp of captured or browsed-to data, it can leverage this manual association technique. In the future, a field biologist may be able to associate field video or sensor data using simple pen gestures.

4.9 Dissemination and Broader Impact

This research was presented as a live demonstration at IBM Almaden, a poster at the ACM Symposium on User Interface Software and Technology (UIST), and as a full paper talk at the ACM Conference on Human Factors in Computing Systems (CHI) [Yeh, Liao, *et al.* 2006]. To motivate further work in this area, the author has given public talks about ButterflyNet to computer science researchers (Yahoo! Research Berkeley, Intel Portland, and Intel Research Berkeley) and the scientific community at large (Jasper Ridge, Stanford Biological Sciences, and NASA Ames).

As part of the dissemination efforts, we have deployed versions of ButterflyNet to two instances of BIO 175 at Stanford University (the Los Tuxtlas class), anthropologists at Intel Portland, a scientist with NASA, researchers at Jasper Ridge, and computer scientists (Intel Berkeley) exploring wireless infrastructures in Ghana [TIER 2007]. For example, the Los Tuxtlas and NASA deployments digitized 2133 pages, containing 628,288 ink strokes.

ButterflyNet has also been used as a springboard for other research in our laboratory. Using the browser as a foundation, Brian Lee and collaborators created the iDeas project, which supports designers with novel sharing and information retrieval techniques [Lee 2007, Maldonado, *et al.* 2007]. The rich spreadsheet metaphor has also inspired PhotoSpread, a system for tagging and querying photos [Kandel, *et al.* 2007] (the author advised this team early in the design phase).

This project was a collaborative effort, and provided research experience for multiple students. Chunyuan Liao (a visiting Ph.D. student) implemented the bracket gesture, and Brian Lee (a Ph.D. student in our lab) implemented parts of the browser's GUI. Boyko Kakaradov (an undergraduate research assistant) and Chunyuan Liao helped with the study.

4.10 Informing Toolkit Design

The experience in designing, building, and evaluating ButterflyNet helped to inform the design of PaperToolkit. First, the study demonstrated that automatic time-based correlation was highly successful. A toolkit might provide correlation of handwritten notes as a core abstraction. The correlation algorithm might process ink strokes and cluster them based on time or location. These abstractions could save the developer time and mental effort, and enable rapid design of ButterflyNet-like applications.

Participants mastered the bracket gesture quickly. Pen gestures can be a fluid way to specify commands to an application. A toolkit should support recognition of gestures, and allow the developer to experiment with different recognition approaches and algorithms.

In reflecting on ButterflyNet's development, much of the time and effort was spent in displaying, scaling, and aligning digital ink on the screen. PaperToolkit should provide abstractions to store and manipulate digital ink, and render it at different scales in the GUI. Second, ButterflyNet supported batched and real-time interactions, and the code looked and acted differently in the two cases. Handling batched data entailed processing and storing ink strokes into files, for later display. Real-time data required code that turned the coordinates into higher-level events, such as a pen tap, drag, or gesture. A key goal of PaperToolkit (see CHAPTER 6) was to unify these two programming models.

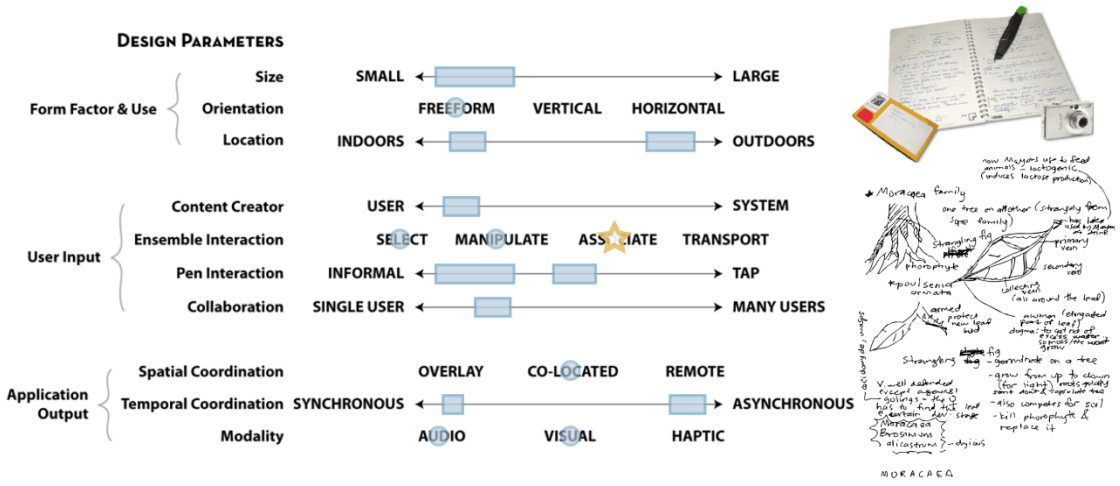


FIGURE 4.15. ButterflyNet presents a small freeform paper interface that is mainly used outdoors by a single field scientist. The main benefit is that ButterflyNet associates handwritten media with digital content, such as photos. The user generates most of the content through informal inking interactions. In the field, audio and graphics output is presented on a nearby handheld device.

4.11 Summary

This chapter presented ButterflyNet, a smart notebook and camera ensemble that helps field scientists collect, organize, and retrieve research content. ButterflyNet is primarily a single-user interface (see Figure 4.15). Since it is intended for outdoors use, the notebook is small and lightweight. The next chapter examines a different part of this design space, larger paper interfaces that support co-located collaboration between two or more users.

5

Large Printed Interfaces

The previous chapter described ButterflyNet, a smart notebook interface designed for field scientists. This chapter explores a different part of the design space: interfaces that support co-located and remote collaboration. Interactive Gigapixel Prints (GIGAprints) are large augmented paper interfaces that can be placed onto a table or wall to support multiple simultaneous users.

Large surfaces such as tables and walls support collaborative activities in many disciplines, including design and engineering. We introduce GIGAprints, a class of applications comprising large paper prints augmented with digital pens and computer displays. With this system, one or more users work around a large poster or map, adding handwritten annotations, drawing pen gestures, or tapping on items printed onto the sheet. To augment the sheets, the GIGAprint system overlays Anoto dot pattern onto imagery such as

photographs or map data. The digital pen communicates coordinates to a nearby PC in real-time (through Bluetooth wireless). The handheld or desktop computer interprets the pen actions and provides visual or audio feedback. A user study with 12 participants showed that GIGAprints can aid collaborative work and support visual search of information graphics.

5.1 Background

For centuries, people have used large paper information graphics as cognitive aids in tasks such as navigation, design, and architecture. This investigation was inspired by the rich information environments present in large prints, such as the high resolution photographs from the Gigapxl Project [Gigapxl Project 2006], large posters that facilitate debates [Horn 2003], Kerouac's *On the Road* [Kerouac 1991] (a manuscript typed on one 120-foot-long roll of teletype paper), Imhof's cartography [Imhof 1982], Minard's flow maps [Tufte 2001].

By including large prints into the ensemble of computing devices, the GIGAprints project provides a *pragmatic* solution to the inability of paper maps and posters to communicate with our computers. The project is also an *epistemic* solution, providing a window into the user experience of future interactive wall and table technology. A concept sketch depicting an idealized view of GIGAprints can be seen in Figure 5.1. This world

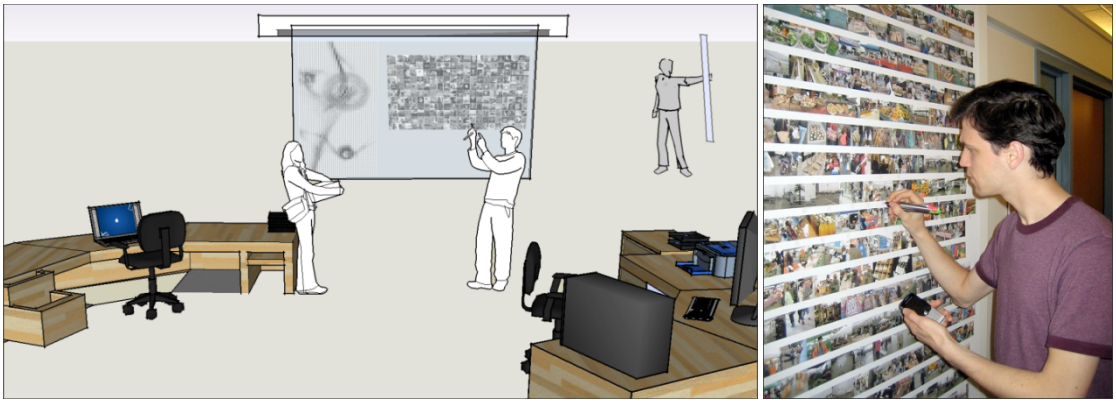


FIGURE 5.1. *Left*) An idealized view of GIGAprints. It supports flexible input, as users can interact with the wall-sized display using their hands and pens. They are large and high-resolution, but allow for mobility, as a user can detach a print, roll it up, and take it elsewhere. *Right*) The current prototype couples a large paper print with a digital pen and handheld device. Graphical feedback is presented on the mobile device, or overlaid onto the print.

includes high resolution digital displays that update at interactive rates (*e.g.*, to show video). Multiple users can gather around these displays, and interact with them using their hands or pens. These displays are portable. Unlike today's large flat-panel monitors, future GIGAprints can be removed from the wall, rolled or folded, and moved to another location (*e.g.*, onto a table).

The current implementation displays static imagery on large paper prints generated by a 44-inch Epson inkjet printer, provides input through digital pens, and presents output on handheld devices or nearby LCD monitors. A video of several of the GIGAprints presented in this chapter is available on the web at http://hci.stanford.edu/gigaprints.

Benefits and Drawbacks of Current Practice

There are tradeoffs in the current practice of using large table and wall surfaces for collaboration. For example, designers may use affinity diagramming (*e.g.*, see [Beyer and Holtzblatt 1997]) to analyze the results of a brainstorm on a whiteboard. Ideas are recorded onto note cards, which are then sorted into clusters on a table or wall. The benefit of this interaction is that the work can happen in parallel (participants can break into groups to work on their own set of cards). A drawback is that these cards must be transcribed into an electronic document to be shared. (An alternative practice is to capture a digital photograph of the whiteboard or table).

Walls can also support collaboration by displaying large amounts of content at the same time. In a study of editors at publishing companies, Bellotti and Rogers found that teams depended on walls to support group discussions. At a newspaper office, "a review meeting is held where the managing editor tacks the pages of yesterday's paper on the wall of the meeting room and everyone sits around and critiques the stories, graphics, and layout" [Bellotti and Rogers 1997]. They found similar practices at a magazine company, where pages are posted on a wall in one of the rooms, "where anyone can come and discuss the content and the all-important design and overall flow from page to page. Sticky notes are

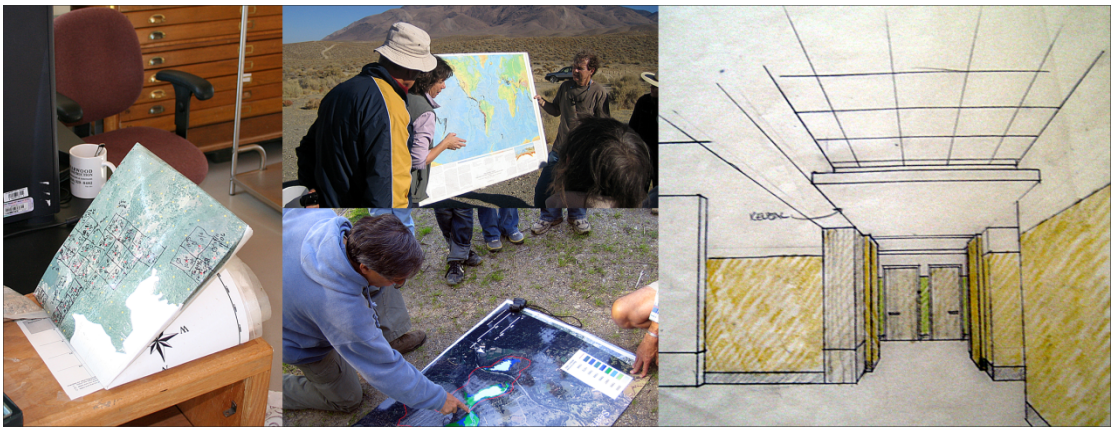


FIGURE 5.2. *Left*) an ant biologist's 4x3-foot map is folded up before it is taken out to the field. *Middle*) Maps are used by biologists for teaching and communicating with students and new researchers. *Right*) architects use sketches and prints, such as this perspective view, to explain design ideas to clients and colleagues.

also attached to the pages as comments for others to pick up." Thus, walls not only support synchronous collaboration (meetings), but asynchronous tasks (comments for passers-by).

To observe the utility of large wall and table spaces, we continued our observations on field biologists (see Figure 5.2). We observed and worked with six biologists, and found that large-format prints were used in many tasks. In particular, large maps were used in research planning, data capture, and communication. For planning, the Jasper Ridge camera trapping project used maps to determine where to place cameras to automatically capture photos of passing mammals. For data capture, one biologist used a 4×3-foot map to in the field to capture her ant survey data. She would write observations on the map, effectively geo-tagging her research data as she traveled between sites. Finally, we have seen biologists use large maps in the field to teach and communicate with incoming researchers (see Figure 5.2, middle).

5.2 Interactive Walls

Hanging a GIGAprint up on a wall simulates an interactive wall-sized display. The wall prints can act as an ambient display, presenting large amounts of static high-resolution content, such as photographs, data visualizations, and text. Passersby can glance at the content, or

walk closer to read the details. Because the print is large, several people can use this display simultaneously, providing input with their pens. The following examples demonstrate the range of wall-oriented interactions we explored.

Network Monitoring

Large public visualizations can support input from multiple users. Additionally, the print medium can be used to display high resolution graphical content. To explore these avenues, we created a 3×6-foot print that hangs in the hallway, to help our research group monitor the health of computers in the lab (see Figure 5.3). This print comprises four areas of interactive content. First, it displays a grid of 225 graphs arranged in a small multiples pattern (*e.g.*, [Tufte 1990]); each shows the network activity of a single machine. Second, the print displays 135 charts representing external servers (*i.e.*, ISPs) that our machines communicate with. These two areas comprise 561,000 data points. Third, the print includes buttons to invoke commands. Finally, it contains a map of the world to allow users to invoke location-based queries.

To use the system, a user taps on button regions representing a computer, ISP, or system command. The user can also drag on the map to select a region (see Figure 5.3, right).

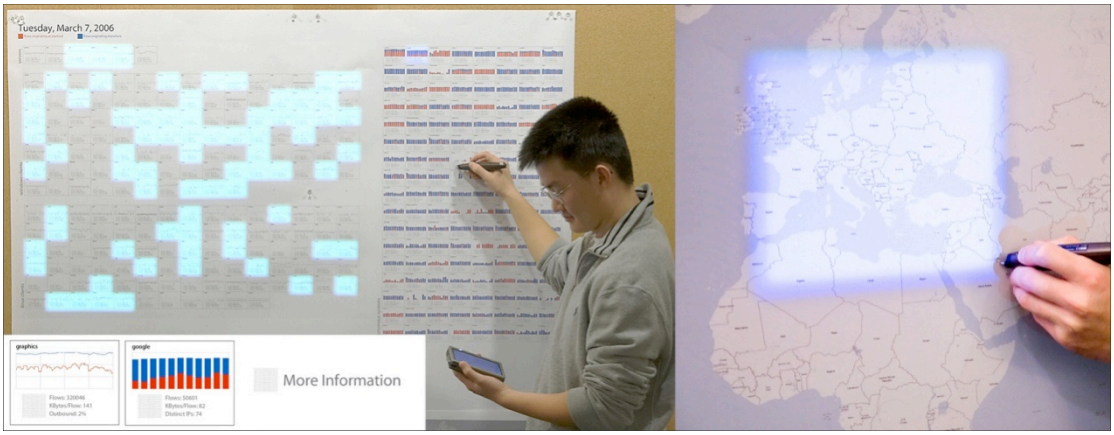


FIGURE 5.3. The network monitoring interface displays real-time and historical information about the computer network. The line graphs and bar charts display 561,000 data points. Here, the user selects an external server to see its details on his handheld display. He can also query a map by dragging the tip of the pen (with a non-inking barrel) across the paper map. The results of his queries are projected onto the printed display.

Textual feedback is presented on a handheld, and coarse-grained feedback is presented via overlaid projection.

Users can alert a system administrator about suspicious activity by tapping EMAIL and then tapping a particular PC. They can also annotate the print with ink. These two interactions facilitate asynchronous collaboration.

The next set of interactions includes the mobile device. While computer names have been anonymized on the printed public visualization, a network administrator can approach the print and select any computer; real-time private data is then sent to his handheld.

Finally, richer interactions are available through integration of the printed display with projected output. The projector acts as a targeted spotlight to provide real-time information. In the ambient mode (default), the projector continuously highlights machines as they engage in activity with remote ISPs. When a user approaches and taps on paper buttons, the projector highlights the results of queries. For example, the user can select the European continent on the map to highlight the machines in our lab that are communicating with ISPs located in Europe.

This application, in particular, is related to a prior system called Ariel, which used overlaid projection and vision-based tracking of light pens to support the review of engineering drawings [Mackay, *et al.* 1995]. However, GIGAprint applications do not depend on overhead camera-based tracking, so mobility is preserved. A worker can fold up a GIGAprints and use it outside (albeit without the projector feedback).

Photo Wall

Inspired by the value provided by high-visibility artifacts in publishing houses (*e.g.*, [Bellotti and Rogers 1997]) and design studios (*e.g.*, [Klemmer, *et al.* 2001]), we developed the Photo Wall. This display enables users to visually scan a large set of images, and send a photo to their mobile device by tapping the pen on a region beneath it (see Figure 5.4, left). For organizations such as news agencies (as surveyed by Bellotti and Rogers), a large, high-resolution display that can help users find the right asset for a layout can prove invaluable.

This type of display provides several benefits. In a design studio, it can act as an ambient display to enhance the visibility of artifacts (see Figure 5.4, middle), or allow designers to visually compare multiple designs. Because the display is always on, it supports asynchronous collaboration; one user can write a comment on one photo, it can be seen by a collaborator later in the day. It can also serve as wall decoration (Figure 5.4, right).

For our collaborators in biology, a photo wall could help with the camera trapping work, where cameras automatically capture photographs of mammals in the field. The biologists commented that they could find the photo print useful for sorting through research content, and comparing different photographs of bobcats.

FIGURE 5.4. *Left*) A user retrieves a photo to his mobile device by tapping on a paper button underneath the photograph. *Middle*) This photo wall served as an ambient display for passers-by during the project fair for the CS247 design class. *Right*) It can also serve as a decorative poster in an office.

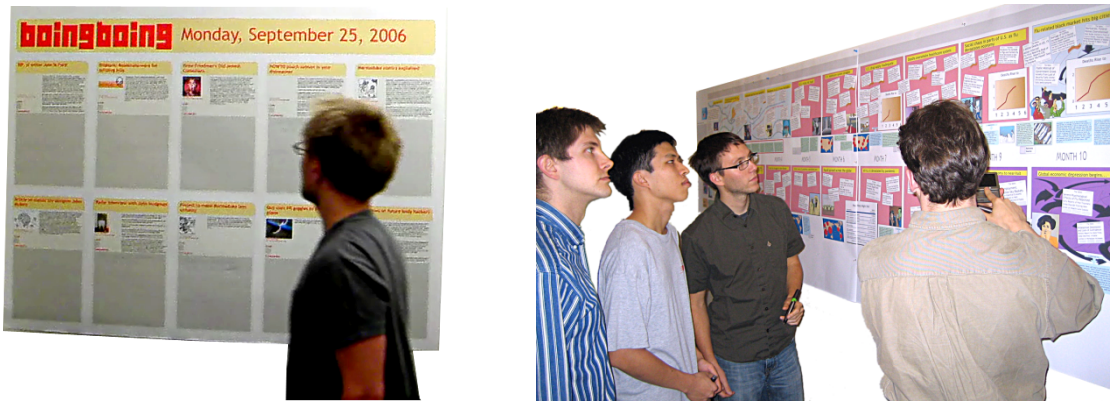


FIGURE 5.5. *Left*) The Blog Reader displays online articles. A passerby can read the articles, or provide comments in the gray inking areas. These comments are uploaded to a website. *Right*) An interactive timeline can facilitate group discussion [Horn 2003].

Intermittent Updates with a Blog Reader

The blog reader prototype pictured in Figure 5.5 displays ten articles from a Really Simple Syndication (RSS) feed in a public place, and provides areas for handwritten comments. Individuals who pass the display are able to casually scan it for new stories, or new comments on the stories that interest them. The comments are captured digitally, and published as images to the Web. This application provides asynchronous collaboration, both locally (subsequent passers-by) and remotely (on the web). The paper display serves as a prototype of a future world where digital walls can display articles ambiently.

Posters

Research posters are used in many disciplines to communicate new findings at meetings and conferences. For the International Conference on Ubiquitous Computing (UbiComp) in 2006, we augmented a number of our laboratory's research posters with dot-patterned regions. The regions supported simple tap interactions that invoked actions, such as starting a video. For example, tapping on an author's name would create a new email document, with the author's email address filled in.

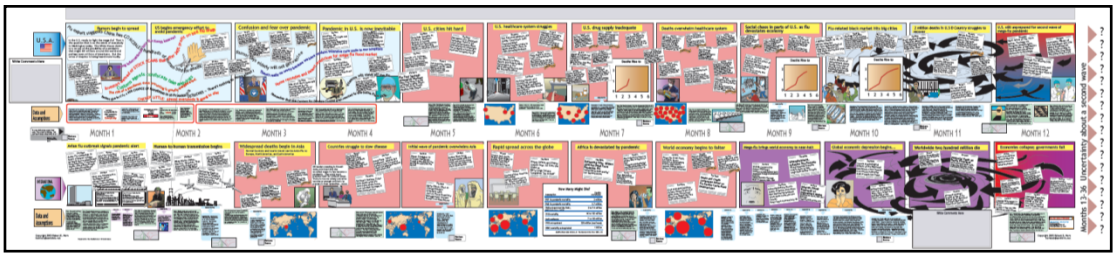


FIGURE 5.6. This 14-foot print is an interactive timeline (content by [Horn 2003]) containing three types of regions. Large comment regions allow users to write in comments that are shared to the web. Voting regions are paper buttons that allow a user to agree or disagree with an issue. Link regions allow a user to retrieve a news article to his handheld display.

Interactive Timelines

Interactive timelines are the largest prints explored through the GIGAprints project (see Figure 5.6). We created a prototype using Horn's 14-foot-wide Avian Flu timeline as the background content [Horn 2003]. The size of the print supports more than four simultaneous users. Users can read the display and discuss issues with each other. The timeline provides three types of augmented regions—retrieval, command, and inking. Users vote up or down on issues by marking the corresponding areas with a digital pen. They can retrieve digital articles to a mobile device by tapping on a retrieval button. They can also write comments in special inking areas, which will be uploaded to the Web. One limitation is that it took more than four hours to print this timeline using the Epson 9800; for more discussion of tradeoffs, see SECTION 5.8.

5.3 Augmented Tables

Walls and tables have distinct affordances. Interactive wall posters are normally ambient displays, but can also support small groups where a facilitator drives the interaction. Tables, on the other hand, more easily support group discussions where each participant plays an equal part. This section describes several GIGAprints that support table-based activities.

Co-located Collaboration and Competition

We designed an augmented tabletop print based on Horn's mess maps [Horn 2003]. A mess map is a large print used to facilitate debates and discussions in small groups (*e.g.*, four people). We augmented a 4×2-foot Avian Flu mess map, which supports discussion of the potential effects of an outbreak on the economy. Each person has a digital pen, and may make annotations on the print during the meeting. These annotations are recorded for later retrieval. A user can also access a digital article by tapping any excerpt (this loads the URL onto a nearby monitor). The prototype was presented to Horn, who provided positive feedback and ideas to extend the design.

Twistr is also a tabletop print, but instead of supporting collaboration, it targets competition. In this game, a large sheet of photographs is placed on a table. Two players each carry two pens, one per hand. A nearby monitor shows four photos per turn, one per player per hand. The object is to tap your own photos before the other player taps his. This game demonstrates two aspects of GIGAprints. First, the application can support more than one user through multiple pens (the pens are identified through an ID). Second, this application shows that the prints allow for visual search of large amounts of graphical content. Users do not need to scroll through low resolution thumbnails.

Remote Collaboration with BuddySketch

We developed a GIGAprint for remote users to discuss printed content. In this application, participants at each remote location print out identical sheets containing content they want to discuss. For example, Candace prints out bobcat photographs from her research project. Roger and John, Candace's remote collaborators, print out the same content at the other site. They then communicate over the phone or video conference. When Candace wants to refer to a photo, she can select it with a tap of her pen. The photo is displayed on the monitor at both locations. If Roger now selects a photo, it will be displayed next to

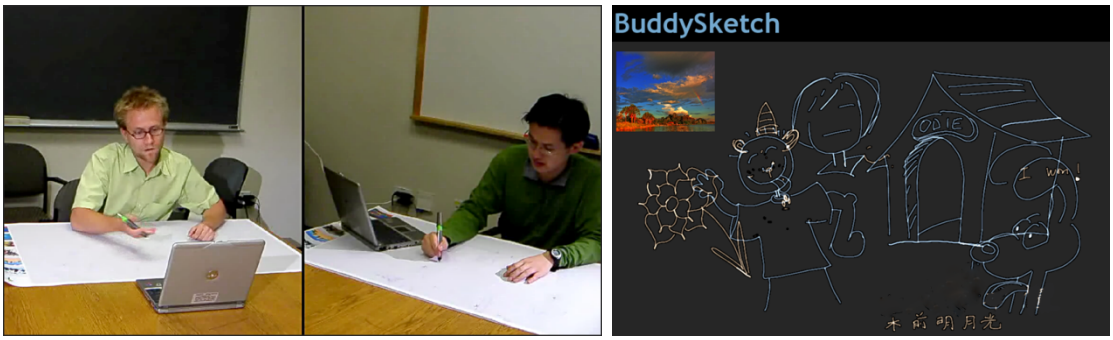


FIGURE 5.7. BuddySketch supports video conferencing by transporting sketches and photographs between computers. *Left*) Handwriting and sketches are sent in real-time to the remote computer. The laptop displays both sets of drawings. *Right*) A screenshot of sketches made by two users during one of our user study sessions.

Candace's. They can also trade ink annotations written on their sheets. Drawings are sent immediately to the remote computer (see Figure 5.7).

This application supports mobility and flexibility, as the participants do not need to remain seated, unlike with solutions where a camera is pointed at the sheet. The wireless pen enables a user to sit at a table, or stand next to a wall, or work out in the field while sharing his sketches. Shared sketching is a well-researched area (*e.g.*, ClearBoard [Ishii, *et al.* 1994], VideoDraw [Tang and Minneman 1991], and Distributed Designers' Outpost [Everitt, *et al.* 2003]). BuddySketch has the advantage that participants may reconfigure their drawing surface during the meeting, and that drawings are permanently captured on paper. However, unlike the ClearBoard, VideoDraw, or Outpost, the BuddySketch application does not help users coordinate their drawings (*e.g.*, through a video feed).

5.4 Mobile Prints and Data Capture

Maps have long been used for their resolution, size, and portability. To explore the benefits of augmenting maps, we prototyped several map-centric interactions. The first enables users to select geo-tagged photographs from a database by tapping on a location on the printed map; matching photographs are presented on the mobile display (see Figure 5.8).

However, in some situations, users may not have access to a graphical display. We explored this scenario by building AudioGuide, an augmented tourist map. This application

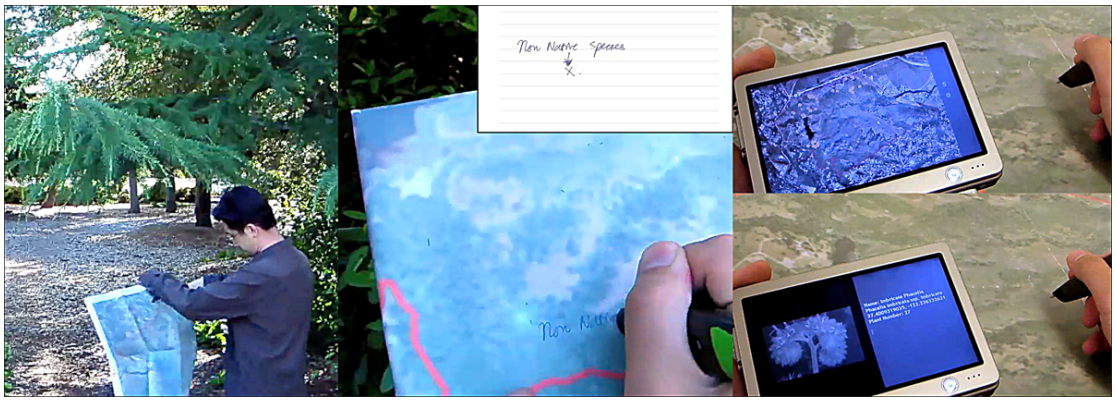


FIGURE 5.8. The augmented map provides interactions for the field and lab. Annotations are captured in the field and presented in a browser. In the lab, an ensemble interaction retrieves photos by GPS location. The user presses his pen down on the map, manipulates the handheld's scroll wheel to change the search radius, and releases his pen to start the query.

can be shared between two users, and includes the pen, a Bluetooth audio earpiece, and a handheld display (see Figure 5.8).

Instead of returning feedback on a webpage, AudioGuide broadcasts information to an audio headset (over Bluetooth). When a user selects a region on the map, an audio description of that region plays into the earpiece. The AudioGuide studies the possibility for using GIGAprints out in the field, with no digital graphics display. A biologist might use this to retrieve information about her experimental site, without having to read it from a screen.

Finally, we designed an augmented map for ant surveys, and demonstrated its interactions to five biologists at Jasper Ridge. In addition to the features from AudioGuide, the map enables a user to add ink annotations. When the user finishes writing, she hears a voice confirming that input has been captured for that particular site (*e.g.*, "data saved for site 34" generated through text-to-speech). This print supports the mobility of field biologists, and requires only two hands (map in one; pen in the other). Future designs can integrate a handheld display and GPS, which may be useful if the biologist has a collaborator to hold the extra devices. An added benefit is that the map can be cut into pieces if the biologist decides to split up work with teammates. It can also be rolled up or folded. This flexibility supports the biologist's requirements for mobility (see SECTION 3.2.1).

5.5 Exploratory Study

We evaluated the GIGAprint interactions through an exploratory laboratory study. These interactions were targeted toward a general audience, with emphasis on collaboration. We recruited six pairs of users from the Stanford community (six male; six female) for a total of 12 participants. Eight of the users were biologists affiliated with Jasper Ridge.

We selected three of the prints to test with users—Twistr, BuddySketch, and AudioGuide. This study examined three hypotheses:

- H1 Large prints provide rich visual context by presenting large quantities of graphics side-by-side. This aids in search and comparison.
- H2 The prints aid collaboration through their large size, support for multiple pens, and output to multiple devices.
- H3 The robustness of pen and paper and the ability to use alternative feedback modalities (*e.g.*, audio) supports mobility.

Each session lasted 1.5 hours. In the session, the pair of users were first introduced to the technology, and then completed three tasks—one per print. They then filled out a questionnaire evaluating the applications and the overall concept of GIGAprints. Each participant received a $25 Amazon gift certificate.

For the first task, the two users engaged in three rounds of Twistr, each lasting 15 turns. The pair listened to a brief tutorial of the game, and then began. A single laptop presented photos for each competitor's left and right hand. A participant would then search the large print for his target photos, and tap on the left photo with the pen in his left hand, and the right photo with the pen in his right. Whoever scored the most points at the end was declared the winner.

The second task simulated a video conferencing session, supported by BuddySketch. We seated the users across the room from each other, such that each person could talk and hear (perfect fidelity video conferencing) but could not see what the other was writing. Each had a laptop computer and a large paper sketching interface. Users were asked to discuss a

topic of their choice and to transport drawings and photos to the other person's laptop computer using BuddySketch.

For the final task, the users were asked to act as tourists of the Stanford University campus. As they walked around outside, they tapped on marked sites to hear a description of that building through a Bluetooth earpiece.

5.6 Results

The study revealed much about how GIGAprints can support visual search, collaboration, and mobile/flexible interactions. This section details the issues, highlighting questionnaire results and our observations during the study.

Visual Search

The most obvious benefit was the fact that users could view lots of content at once. This factored into Twistr, where users remarked that they started to remember, after only a few rounds, where certain photos were located. In particular, some commented that photos with faces and sharp colors would especially jump out. In the questionnaire data (see Figure 5.9), Twistr was rated highly for its ability to support visual search.

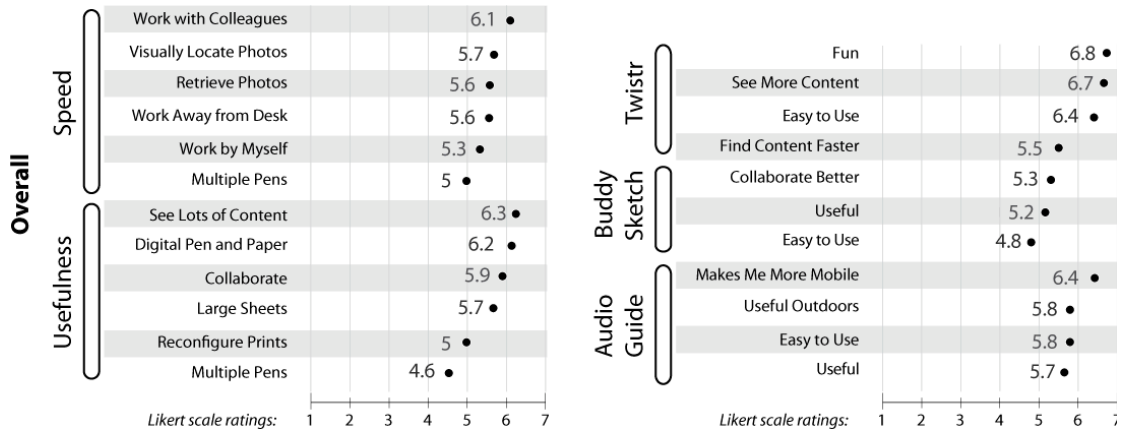


FIGURE 5.9. *Left*) The overall evaluation of GIGAprints (N=12) revealed that it could support collaboration, and enable users to see and locate lots of visual content. *Right*) User ratings of individual applications show that GIGAprints can provide an fun and engaging experience, and enhance mobility. The BuddySketch application needs further iteration to enhance usability.

Collaboration

GIGAprints have the potential to enhance remote and co-located collaboration. With BuddySketch, users were able to share their drawings in real-time. The questionnaire data show that users felt it could help with collaborations. During the study, we observed users collaborate on a variety of topics. One group used the shared drawing space to analyze different football plays.

Players were willing to reach around each other to acquire their photographs (Edward Hall refers to this as the *intimate distance*, in his work on proxemics [Hall 1990]). The Twistr game illustrated how large prints can support simultaneous users around a table. With the AudioGuide map, users worked with each other to determine which areas of the map to listen to. In the common configuration, USER A would carry the pen and map, while USER B would hold the handheld and wear the earpiece. The driver (A) would then tap on parts of the map, while the listener (B) would explain what the map was saying.

Mobility and Flexibility

Electronic displays of comparable size usually trade off mobility for high resolution graphics. GIGAprints instead provides some mobility and flexibility, but does not allow for real-time graphics (except through a handheld display).

During the BuddySketch task, we observed users moving their laptops and prints around to work on different areas of their drawings. With the AudioGuide, participants walked around outside while interacting with the print and receiving audio feedback. The print worked well in the sun, where the handheld's display (which we did not use) was rendered unreadable. The questionnaire data show that users felt AudioGuide could be integrated into a mobile workflow (6.4 out of 7). One biologist exclaimed to her collaborator, "this feels very much like we are doing our ant survey."

One pair of users tested the strength of the wireless connection between the handheld and the pen. The user with the handheld and earpiece walked away while the user with the map repeatedly tapped on audio regions. Eventually, the connection broke, and the

application degraded to a regular paper map. Ink could still be written on it, and it still provided static location information. However, audio descriptions were not available. In ensemble interfaces such as the AudioGuide, software should be able to withstand disconnections in communications, and ideally support automatic reconnections.

Limitations of the Applications

The study revealed limitations in the applications, and with GIGAprints in general. One limitation was due to a tradeoff between flexibility and features. With BuddySketch, we strove for an interface that was easy to set-up and move around. Thus, it did not provide overlaid projection so that a user could see the other's sketches georeferenced with his own. Users reported this as the main limitation with BuddySketch, because they wanted to work on the same drawing, instead of sharing distinct drawings with each other. Ideally, BuddySketch would use paper that could provide graphical updates, such as electronic paper [E Ink 2007]. Electronic paper will be necessary for interfaces that need to georeference graphical feedback with the user's pen tip (*e.g.*, in shared whiteboard applications).

One AudioGuide user pointed out that integrating with GPS could help tourists (and map users in general) find their bearings on the map. Thus, a user could tap on the paper map to see if his expected location agreed or disagreed with the GPS system's information. The audio could also be automatically tailored to the person's current orientation (*e.g.*, "behind you is the Stanford Oval").

For many GIGAprints, the pen interaction not only provides a method to capture handwriting and sketches, but also acts as a pointing tool. However, the pen provides the application only one input location, and is not yet rich enough to replicate the pointing and gesturing we do with our fingers. In the future, interactive paper might support touch input. This would enable natural bimanual control of prints. For example, one hand might position a photograph while the second annotates it with the pen. A prototype approximating this future interface can be seen in SECTION 7.5.

5.7 A Design Space for Interactions and Prints

As part of this work, we began to formulate our design space for paper + digital applications and interactions. This section describes how we developed parts of the our paper + digital design space, which is shown in full in SECTION 2.2 and referred to in CHAPTERS 6 and 7.

Pen-based interactions with GIGAprints fall into four categories: select, manipulate, associate, and transport (see Figure 5.10). *Select* is an operation by which the user specifies some content of interest. For example, a user can circle a map to select photographs tagged with a particular location. *Manipulate* includes actions that modify the specified content. One might drag a pen over a floor plan to manipulate a 3D view of an architectural design. *Associate* enables the creation of symbolic links between physical and digital elements. ButterflyNet (see CHAPTER 4) enables users to associate a photograph to a location in a notebook. Finally, the sending of data to the receiving device is accomplished through *transport* operations. This taxonomy for interactions was developed for GIGAprints, but has been included in our paper + digital design space (SECTION 2.2). It was created by examining prior work in defining graphical input primitives [Foley and Wallace 1974] and graphical input architectures [Myers 1990]. This space subsumes Foley and Wallace's *pick*, *button*, *locate*, and *valuate* into the first two ensemble interactions (*select* and *manipulate*). This space contributes *associate* and *transport* operators for reasoning about interactions that involve multiple devices (especially paper).

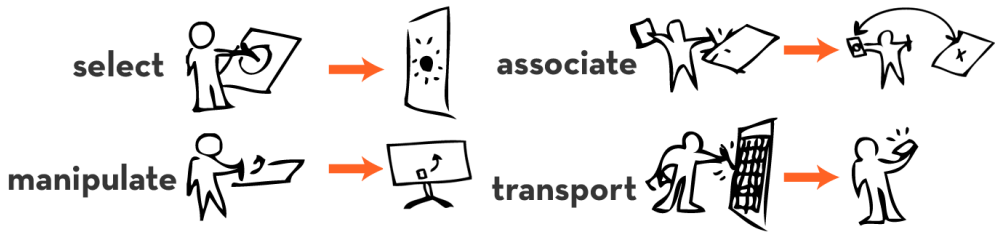


FIGURE 5.10. There are four types of paper + digital interactions. SELECT designates items of interest on the receiving device. A MANIPULATE interaction modifies content on the receiver. ASSOCIATE creates a link between information in the paper and digital worlds. TRANSPORT describes the movement of content between the print and the device. The underlying communication happens between the computer hosting the GIGAprint application and the device that displays the feedback (they may be the same computer).

The design of the interactive prints can also vary along different axes, including *scale*, *spatial* and *temporal coordination* of content, and *update frequency*. Most GIGAprints fall onto the larger end of the scale spectrum. The smallest GIGAprint we designed was 3.5-feet wide; the largest (Horn's timeline) was 14-feet wide. Smaller prints support mobility, whereas larger prints support co-located collaboration (around walls and tables). GIGAprints fall into the *yard* category of Weiser's "computing by the inch, foot, and yard" [Weiser 1991]. Example systems on the inch, foot, and yard scales, include (respectively) WebStickers [Holmquist, *et al.* 1999], Books with Voices [Klemmer, *et al.* 2003], and DigitalDesk [Wellner 1993].

Spatial coordination refers to how graphics (printed and digital) are presented in relation to each other. Klemmer's taxonomy for tangible computing includes four categories: georeferenced, collocated, non-collocated, and non-visual [Klemmer 2004]. Our spatial coordination axis includes only three: overlaid, co-located, and remote. Non-visual output is moved into the modality axis. Our three categories cover the GIGAprints we have developed. When digital and paper content are overlaid, the graphical feedback is georeferenced with the print (*e.g.*, the network monitoring application's projected overlay). With co-located systems, the paper and digital content occupy adjacent physical spaces (*e.g.*, the Photo Wall). In remote systems, the physical and digital content may be far from each other. In the Blog Reader, the published RSS feed lives on the web, apart from the print.

Temporal coordination represents the measure of time it takes for a paper-based action to invoke a change in the digital content. This coordination can be synchronous (*e.g.*, real-time audio feedback in the AudioGuide) or asynchronous (*e.g.*, handwriting captured on a map, and later uploaded to ButterflyNet).

Update frequency is unique to paper + digital applications, because paper does not present real-time graphical updates. GIGAprints can only approximate large digital displays by refreshes of the print. A wall-sized poster might be static (never updated); the Blog Reader can be updated intermittently (daily); a streaming display of incoming photos

update in real-time (printed continuously). The idea of update frequency for GIGAprints was later folded into the *temporal coordination* axis for the final paper + digital design space.

5.8 Tradeoffs with using GIGAprints

This section reviews key insights from our analysis of the benefits and drawbacks of using GIGAprints. We first look through two theoretical lenses: Klemmer *et al.*'s ubiquitous computing themes and Green *et al.*'s cognitive dimensions. We then discuss the real-world issues of deploying GIGAprints as end-user applications and as prototyping tools.

Ubiquitous Computing Themes

Klemmer *et al.* described five themes to consider when designing for physical activity in ubiquitous computing systems [Klemmer, *et al.* 2006]. One theme is the *visibility* of artifacts. When used in locations such as a design studio, GIGAprints can promote visibility: the display's permanence can make design practices apparent, and also support co-located collaboration. A second theme is that of supporting *thick practice*, because "interfaces that *are the real world* can obviate many of the difficulties" of modeling work processes in software. GIGAprints fit well into the many disciplines that currently use large paper maps and posters. The added benefit is that if the technology fails, the interface degrades more gracefully than a comparable digital-only solution. A non-working GIGAprint is still no worse than the paper map a biologist uses today. However, a possible scenario is that the user has come to depend on the print as an interface to some software; in that case, if the pen hardware fails, the print is rendered useless. A third theme that applies is *performance*. For text entry, a keyboard is still much more efficient than writing with a pen. However, pens are high performance tools for sketching; this can be leveraged by any GIGAprint that takes informal sketching as a primary input (*e.g.*, BuddySketch). In fact, fluid handwriting and sketching is a core benefit of pen computing in general.

Analysis through Cognitive Dimensions

We also applied the cognitive dimensions framework to help us analyze the interactions [Green and Petre 1996]. This framework was developed for programming language design, and has since been adopted to inspect the complexity of interactions. Here, we pick ten of the 14 axes to help us discuss GIGAprints:

Visibility & Juxtaposability: For activities such as exploration and design, GIGAprints can be effective in increasing the visibility of graphical content. The print can permanently reside on a wall or table. When a user looks for information, it will be there, without the need to navigate folders on a file system. However, the tradeoff is that there is a hard limit to the amount of visible information. On the other hand, a graphical interface might present a tiny window, but a user can scroll the view to effectively navigate an infinite canvas. Similarly, GIGAprints have limitations in juxtaposability, since a large print cannot be easily torn apart and reconfigured. Small sticky notes afford better juxtaposability. GUIs are also good, because users can easily reposition digital content.

Mapping & Consistency: These interfaces map closely to existing tools in domains that value paper artifacts (*e.g.*, biology and architecture). Because of this close mapping, and because of the parallels between the pen interactions and mouse-based GUI interactions, the interfaces can be consistent with a user's expectations. For example, pen taps (like mouse clicks) signify the selection of a user interface element. However, GIGAprints do not map well to some domains (*e.g.*, mobile journalists would benefit more from small notebooks).

Secondary Notation & Terseness: A benefit of pen and paper is that adding secondary notation (*e.g.* handwritten annotations) is easy. When a user adds sketches or writing to a print, he is adding supplementary information, which may mean nothing to the computer, but everything to a teammate. However, secondary notation is often better conveyed by voice for co-located collaboration. Terseness describes the ability of the interaction to be short, yet specify sufficient information. As pen interactions map to real-world tasks, input can be expressed as tersely as a check mark in a box, or a squiggle to designate a delete of a paragraph (as long as the computer can recognize it). But certain interactions are better

when working in a GUI. For example, text input is more exact and efficient with a keyboard, and can be a terse form of interaction, especially with auto completion of text fields.

Progressive Evaluation & Role Expressiveness: Since GIGAprints are static, they are not as good as GUIs at supporting progressive evaluation. With a static interface, users do not know if they have provided the correct sequence of interactions to invoke the desired actions. Even with a handheld for feedback, users will need to look away from the print to evaluate the system's state. Additionally, paper widgets may not have obvious affordances (low role expressiveness). Computer users know the language of GUIs (checkboxes allow multiple selections, while radio buttons allow selection of only one item from the group). However, since paper interfaces are not common, there is not yet a standardized language of interaction—users might not know how to interact with the print.

Premature Commitment & Viscosity: As the prints can be quite large, the design of GIGAprints currently requires some premature commitment. Printing a poster-sized interface with today's hardware takes approximately half-an-hour, and changes to the look-and-feel of the interface always require a reprint. Ideally, the toolkit can address this issue, and make the process of designing paper UIs less viscous (adverse to rapid changes).

Practical Concerns of Deploying GIGAprints

Many GIGAprints are only prototypes of future interactive environments. For example, applications like the network monitor allow us to experiment with interactive wall posters. With current technology, GIGAprints take long to print (at least 30-minutes). Ink and paper can also be expensive. Thus, applications that require a refresh of the printed material may be impractical. These applications depend on technology improvements, such as increased availability and lowered cost of large-sheets of electronic paper [E Ink 2007]. However, some of the GIGAprints are practical enough for real-world use. For example, maps are already used by biologists, so adding interactive features only adds value to the application.

5.9 Architecture

We now describe the building blocks that compose a GIGAprint, using the network monitoring application as an example. The main component is the large physical print. The network monitoring application contains two large prints pinned onto the wall. In software, the *LargePrint* module lays out the background graphics and supports rendering to PDF and PostScript (to communicate with printers).

The *ActiveAreas* module maintains the location and sizes of areas on paper that respond to real-time input from the digital pen. These areas will be overlaid with dot pattern when the print is rendered. In the network monitoring application, there are several active areas, including paper buttons to select computers and ISPs, buttons to invoke commands, and a large active area to allow queries on the map.

The user interacts provides input to the GIGAprint by tapping, dragging, or writing with a digital pen. Output is provided through auxiliary devices, such as LCDs, projectors, handhelds, audio headsets, and more. In the network monitoring application, the *PenInput* module handles real-time input by reading coordinates from the Bluetooth serial port. It

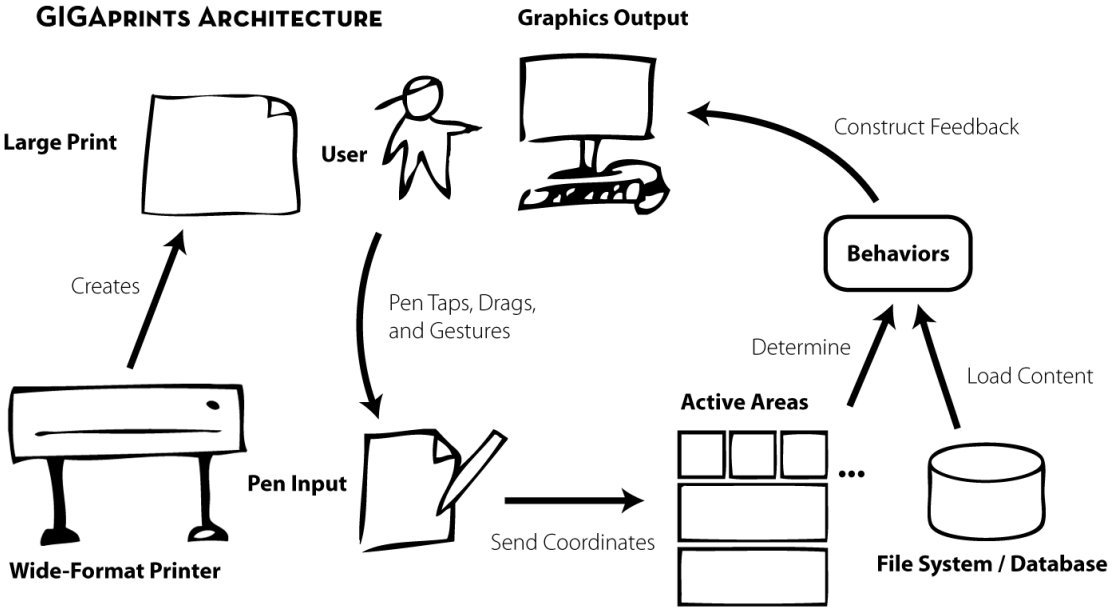


FIGURE 5.11. The software architecture of an Interactive Gigapixel Print (GIGAprint). Graphics output can be projected onto the large print to simulate future electronic paper.

maps the coordinates to an active area to determine what behavior to invoke. The *GraphicsOutput* module provides feedback to the projector or handheld display. In particular, this application's output module maintains calibration information to align the projector output to the printed display. Additionally, the output module sends commands to the handheld through socket-based message passing.

These components, the prints, pens, and auxiliary devices, form the core of every application. The designer must 1) generate appropriate graphic content for the printed display, 2) decide upon pen interactions that suit the task, and 3) connect the behaviors and devices so that appropriate output is triggered for every input. The main components are summarized in Figure 5.11.

5.10 Implementation

The GIGAprints demonstrated in this chapter were written in the Java programming language (J2SE 6.0). While ButterflyNet's interactions were mostly batch processed, the GIGAprints rely on real-time input from the digital pen. The application connects in real-time to a pen over Bluetooth wireless. Pen coordinates are matched against a list of active areas to determine what action the user has invoked.

The *PenServer* broadcasts hardware coordinates over a socket connection. This allows the pen hardware to be hosted on a single machine, and the coordinates to be processed by one or more machines (*e.g.*, a handheld device). A listening application reads in the coordinates and invokes an appropriate response for the end user. Each application needs to map incoming pen coordinates to actions like pen taps, drags, *etc.* The process is analogous to programming a GUI by reading mouse coordinates, and requires effort by the developer to keep track of coordinate-to-behavior mappings.

Another advantage of the *PenServer* design is that a developer can *telnet* into the server to debug the pen. This is useful because in this new genre of applications, it is sometimes difficult to identify if the failure stems from the software or from the pen hardware itself.

Most applications require a single PC or handheld, which runs the software that communicates with the pen and provides user interface feedback. However, a few applications require communications between multiple devices (*e.g.*, BuddySketch and NetworkMonitor). For these, an *ActionReceiver* (inspired by *PenServer*) simplifies message passing: it provides convenience methods for sending Java objects (*e.g.*, message strings) between two devices, or for invoking behaviors such as loading a URL.

To produce the prints, we generate PostScript files, which include the imagery and dot pattern. The resulting PS file is sent to a 44-inch wide inkjet printer (Epson Stylus Pro 9800). As the Anoto pattern comes in letter-sized (8.5×11-inch) chunks, large prints require smart tiling of the pattern space (see Appendix C.3). This complicates event handling, as some module needs to maintain the tiling configuration, and use it to transform raw pen coordinates into relative coordinates on the large physical print.

5.11 Implications for Toolkit Design

This chapter presented Interactive Gigapixel Prints (GIGAprints), large paper interfaces that support co-located collaboration, visual search of graphical content, and mobility. Its features were based on our observations of how biologists use high resolution maps both in the lab and out in the field.

These GIGAprints supported ensemble interaction techniques that enable users to leverage both the print and related digital media. The techniques were illustrated through multiple prototypes that sampled different points in the paper + digital design space (see Figure 5.12). For example, AudioGuide provides mobility despite its size, because it can be folded. BuddySketch sends remote feedback (sketches) to the other conference participant.

The prints were created using an Epson 9800 wide-format inkjet printer and the ColorBurst RIP. We assembled PDF or PS files with Anoto dot pattern overlaid on imagery. The digital pen sent these coordinates in real-time to the individual application, which would detect pen taps or drags. Output would be displayed through a graphical interface, either projected onto the print, or displayed on a mobile device.

The user feedback and observations from our first-use study of three GIGAprints showed that the large display surface supports rapid visual search. Because the prints can be placed on a table or on a wall, multiple people can collaborate either synchronously or asynchronously. Even though the prints are large, they still provide a measure of mobility—a user can roll up a print and transport it to a different location.

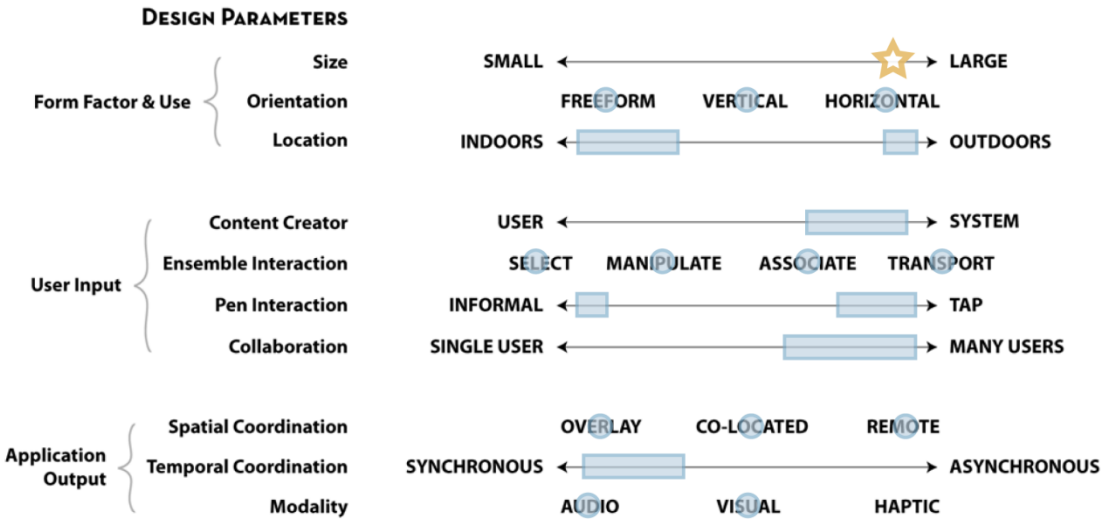


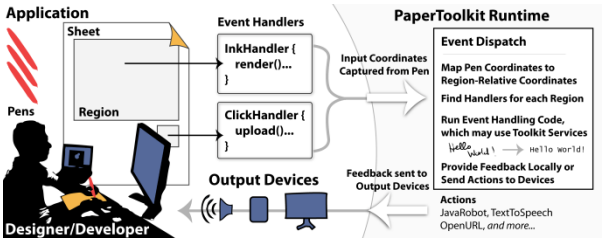
FIGURE 5.12. GIGAprints are large printed interfaces that support multiple users. However, they do not completely sacrifice flexibility and mobility, as BuddySketch AudioGuide demonstrate. The Network Monitor demonstrates overlaid graphics via a projector.

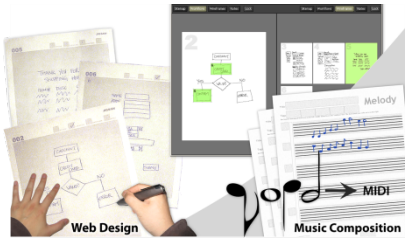
Creating GIGAprints was a difficult and time-consuming process, and the experiences gained provided the following motivational insights for PaperToolkit:

- Paper UIs can come in a wide range of sizes, from small notes to a wall-sized posters. They can contain one or more pages. The toolkit should support *flexible specification* of the paper interface.
- Collaborative scenarios involve multiple users. The toolkit should accept input from multiple pens, and output to one or more devices.
- Support the basic pen interactions, but allow for extensibility. The developer should be able to invent new interactions combining pen, paper, and computer.
- Printing interfaces is slow. We must include methods to speed up interface design, rendering, and testing for our developers.

This research was presented as a video demonstration at UbiComp [Yeh, Brandt*, et al.* 2006], and has been presented in talks at NASA, Intel, and at law firms in Palo Alto, CA.

6





The PaperToolkit Architecture

This chapter presents PaperToolkit, a software architecture and suite of tools that enable programmers to design, create, and test paper + digital applications. First, we present two scenarios to motivate the design of the toolkit, and show how previous pen-and-paper toolkits would have trouble supporting them. The core architecture and abstractions are presented in SECTION 6.2. The section demonstrates how PaperToolkit builds on the programming model for graphical user interfaces, introducing abstractions for handling and transforming ink strokes, providing feedback through remote devices, and coordinating pen input in real-time and batched modes. SECTIONS 6.3 through 6.6 zoom in on particular features of the toolkit. The implementation details are covered in SECTION 6.7. The chapter concludes by discussing how to support different programming models with PaperToolkit's libraries; these alternative architectures would present distinct tradeoffs. Following this

chapter, CHAPTER 7 describes the evaluation of PaperToolkit, while CHAPTER 8 reviews our designs created in response to the evaluation.

6.1 Scenarios

Chapters 4 and 5 detailed two classes of paper + digital applications and the range of interactions surrounding those applications. This section presents two scenarios to motivate how a developer will create programs using PaperToolkit. It then highlights how programmers might use existing toolkits to accomplish these tasks.

6.1.1 Augmented Notebook for Mobile Professionals

Karen, a software developer, is designing the PicFinder Notebook, which automatically recognizes the handwriting in a user's notes and retrieves related news photos from web searches. When the user taps his pen to a region in the notebook, the digitized ink and the related photos are uploaded to a LiveJournal blog. Karen will need abstractions to deal with ink input, handwriting recognition, and exporting ink to image files.

6.1.2 High Resolution Maps for Collaboration

Jack works at a startup that produces augmented paper maps ranging from letter-sized sheets to engineering blueprints. These maps capture ink annotations and allow association and retrieval of media through taps and gestures. Feedback is sent to a nearby mobile phone. Jack will benefit from support for multiple users, high level input handling, and communication with external devices. To facilitate processing, he also needs abstractions for clustering ink, and displaying and manipulating those clusters in a GUI browser.

6.1.3 Exploring an Alternative Approach

As seen in CHAPTER 2, there are three existing approaches for working with pen-and-paper input. The first is to think of the pen as a batched input device for capturing ink strokes on notebook pages. The Anoto SDK does this, presenting abstractions for retrieving and iterating through strokes written on a page [Anoto AB 2007]. The developer can ask for all

ink written on a particular page coordinate (*e.g.*, 33.0.16.24), or on a named area within that page. Second, PADD builds on Anoto, and tracks the originating digital document [Guimbretière 2003]. Thus, incoming ink strokes can be overlaid onto a PDF as annotations. These approaches do not support the handling of *events*, so Karen could not bind the upload action to a real-time pen tap on the notebook page.

Third, iPaper can retrieve ink strokes in batched mode or in real-time [Signer 2006]. It primarily models pen strokes on a region as queries to an underlying database (either local or remote). The database can return associated media (*e.g.*, a sound file) or code snippets called "active components," which model real-time events. This is a powerful and flexible platform for building paper applications. However, the database presents tradeoffs. While it provides flexibility (database entries can be updated at runtime) it also presents a second authoring step for the developer. In contrast, GUI programmers are accustomed to specifying program behavior directly in source code. For example, Swing and Flex programs encode behavior in event handling code written in Java or ActionScript, respectively.

PaperToolkit adopts this GUI approach (see Figure 6.1), because even if it is not an optimal solution, it is a *familiar* one to programmers. The toolkit is modeled after graphical

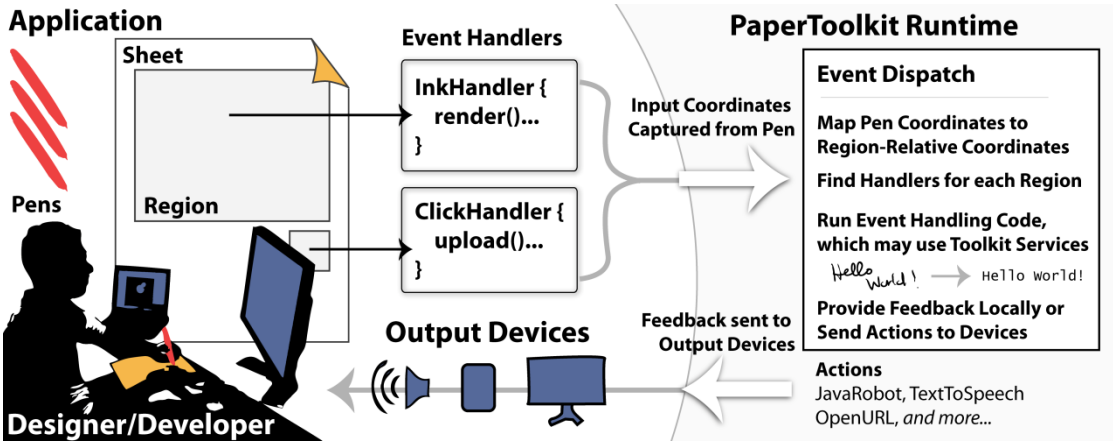


FIGURE 6.1. In PaperToolkit, input arrives from pens (*left*) and is dispatched to event handling code (*middle*). Output is displayed on the local machine or routed to devices (*bottom*). The architecture unifies real-time and batched input by injecting synched data into the event stream. Event dispatch and UI construction are modeled after GUI architectures, to help programmers create paper + digital applications faster.

interface architectures. It supports Karen's and Jack's projects by including event dispatch, layout, and a familiar programming model (see Figure 6.1). It is the first open source toolkit for augmented paper applications, and also allows architects to experiment with different programming models, by providing flexible low-level services (*e.g.*, access to the raw pen coordinates). Additionally, PaperToolkit provides Jack support for communicating with handheld devices, and the ability to rapidly experiment with algorithms to cluster ink marks on the map. As a whole, the toolkit presents a departure from previous work.

6.2 Core Abstractions

Since PaperToolkit offers a GUI-like programming model, the resulting programs look similar to Java Swing programs (see Figure 6.2). The programmer writes in Java to build up a paper *Application* that contains *Sheets*, *Regions*, *Handlers* and outputs to *Devices*. The runtime receives input from a digital pen, and locates and invokes event handlers corresponding to the active regions. Output is displayed on either the local computer hosting the application, or on a remote *Device*.

```
1 public class SimplePaperApp {
2    public static void main(String[] args) {
3       Application app = new Application();
4       Sheet sheet = app.createSheet(8.5, 11);          // in inches
5       Region region = sheet.createRegion(1, 1, 4, 2); // x, y, w, h in inches
6       Device remote = app.createRemoteDevice();
7       region.addEventHandler(
8          new ClickAdapter() {                          // detects pen taps
9             public void clicked(PenEvent e) {
10               remote.displayMessage("Pen Tap");
11            }
12         });
13      app.run();
14   }
15 }
```

FIGURE 6.2. This complete PaperToolkit program contains one active region on a sheet of paper. When a user taps this button, the app notifies a remote device (*e.g.*, a mobile phone). A GUI programmer would find this approach familiar. Additionally, the *Device* abstraction handles message passing, without requiring explicit code for socket communications (lines 6 & 10).

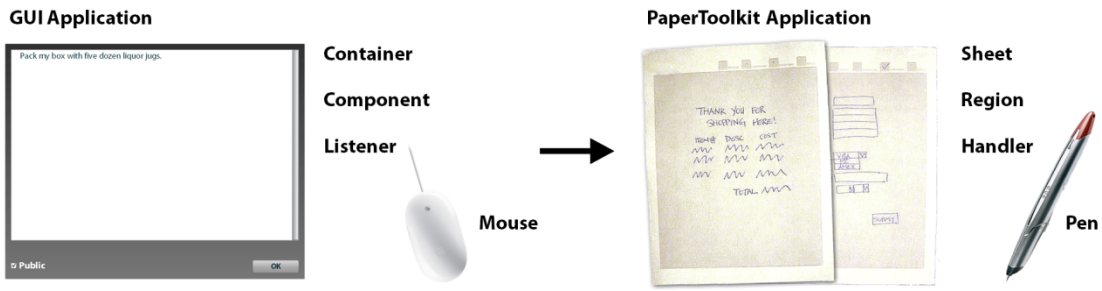


FIGURE 6.3. The basic interface abstractions are borrowed directly from the GUI world. PaperToolkit treats pen input as if it were a mouse (detecting pen down, drag, and up).

6.2.1 Interfaces and Handlers

PaperToolkit's basic abstractions map one-to-one to the most common GUI concepts: windows, components, and listeners become *Sheets*, *Regions*, and *EventHandlers* (see Figure 6.3). With PaperToolkit, the programmer thus has the option to work with two distinct interfaces. The graphical interface allows for mouse and keyboard input, and presents real-time output. The paper interface supports input from pens, but can only be updated in batch, through print operations. At runtime, the paper interface is input-only. Programmers can leverage this second interface for any application that requires the benefits of paper.

Event handlers treat pen input similarly to how GUI listeners work with mouse input. PaperToolkit provides handlers to cover the common interactions, including pen taps, single strokes, gestures, and handwritten input. These handlers are extensible. For example, the *MarkingGesture* and *Gesture* handlers extend from the *Stroke* handler, because it fires an event and provides information for each incoming pen stroke.

Since the paper display does not provide real-time graphical updates, PaperToolkit includes a *Device* abstraction to route feedback to local or remote devices. Devices can invoke *Actions*, including the common tasks of opening files, playing sounds, and loading web pages. With these abstractions, programmers can make applications that support input on paper, and output on a computer. These abstractions are listed on the top of Table 6.1.

	Concept	Examples	Useful in Other Apps
Paper UI	Application Runtime and Event Handling	PaperToolkit, ClickHandler	○
	Paper UI Construction and Printing	Sheet, Region, Printer	◒
	Coordinating Ensemble Interactions	Device, OpenURL, LiveWhiteboard	●
Pen	Pen Data Capture, Manipulation, Rendering	Ink, InkRenderer, InkXMLParser	◒
	Ink Stroke Metrics and Recognition	InkUtils, HandwritingRecognizer	●
Tools	Development and Debugging Tools	SaveAndReplay, PaperUIDesigner	○
	Units, Coordinate Conversion, Utilities	Inches, CoordinateConverter	●

TABLE 6.1. The seven concepts presented by PaperToolkit. Many of these services can be used separately from the toolkit. In particular, the abstractions for digital ink rendering and manipulation can be used by applications that receive input from digitizing tablets.

PaperToolkit is composed of a number of independent services that can be used separately from the toolkit. For example, one abstraction receives input from the hardware pen. A developer could use just this component if she wanted to work with the raw coordinates. PaperToolkit combines these services into a familiar architecture and programming model, providing methods to lay out paper UIs, add behavior to regions, and receive real-time events. Adopting the GUI model provides a basic infrastructure for programmers to quickly create applications. The research lies beyond basic event dispatch, in the abstractions and interactions that distinguish paper applications from purely graphical ones.

6.2.2 Interface Builder

PaperToolkit includes an interface builder for generating paper UIs. For example, to add a look-and-feel to her augmented notebook, Karen first uses a design tool (*e.g.*, Adobe Illustrator or Microsoft Word) to specify the look of the paper interface, producing a background PDF. The interface builder reads in this PDF and enables

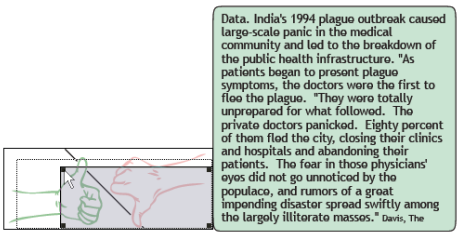


FIGURE 6.4. The UI builder allows developers to add active regions to an existing PDF (created by a design tool). Here, the designer is placing a region over a vote up/down paper button.

Karen to augment it with active regions (see Figure 6.4). With the builder, the developer places, sizes, and names regions through direct manipulation, producing an XML specification of the interface. The program code reads in this specification and attaches event handlers to the regions. This reduces the need to encode the UI components in Java, and allows non-programmers to produce paper UIs.

6.2.3 Binding of Paper Regions to Handlers

When Jack creates his augmented paper map, he might start by adding one large *ImageRegion* to his main *Sheet*. This region will display the map, and allow users to tap on locations to invoke GPS queries over a set of photographs. Jack might also add a second region that acts as a paper button; whenever a user taps on it, the software adds a bookmark—all notes and photographs taken near that time are "starred" for later retrieval.

To support these interactions, the toolkit maintains a mapping from incoming hardware coordinates (streamed over Bluetooth) to region names. The *EventDispatcher* can then locate the correct region, find handlers attached to that region, convert the raw coordinates to region-relative coordinates, and call *handleEvent*(...) in each handler.

Mapping raw coordinates to regions is a necessary step for any toolkit supporting pen input on paper. PaperToolkit saves the mappings in an XML file on the file system, but these mappings could alternatively be stored in a local or remote database (as in iPaper). GUI applications also perform this mapping, but in a slightly different manner. The GUI window manager determines which active window receives the mouse input, and then computes at runtime which components' boundaries intersect with the input coordinates. The listeners on those widgets are then invoked through the component hierarchy. A paper application has the benefit that the interface does not resize at runtime, so the mappings from hardware coordinates to region coordinates can be stable throughout the application's life cycle.

Of course, PaperToolkit does allow the binding to be modified at runtime. This enables a powerful debugging technique, where the developer can test components by drawing them onto arbitrary areas of patterned paper (detailed in SECTION 8.5).

6.2.4 Architecture Diagram

The main parts of PaperToolkit are summarized in Figure 6.5. Each module contains software abstractions that the developer can use in her program. For example, the *Sheet*, *Region*, *Handler*, and *Pen* are available in the *Pen Input* and *Interface* modules. Input is captured and sent to *Event Handling*, which contains the different handlers that the developer has selected for the application. The code in the handlers can work with *Digital Ink*, which can be sent to different *Services* for processing. In Karen's notebook, the digital ink can be rendered into JPEG images for upload to the web. In Jack's map, ink can be clustered and correlated with input photographs.

Event handling code can then send feedback to the local computer or mobile devices, through the *Device* API. The *Device* abstraction and other parts of the architecture are fleshed out in greater detail in the remainder of this chapter.

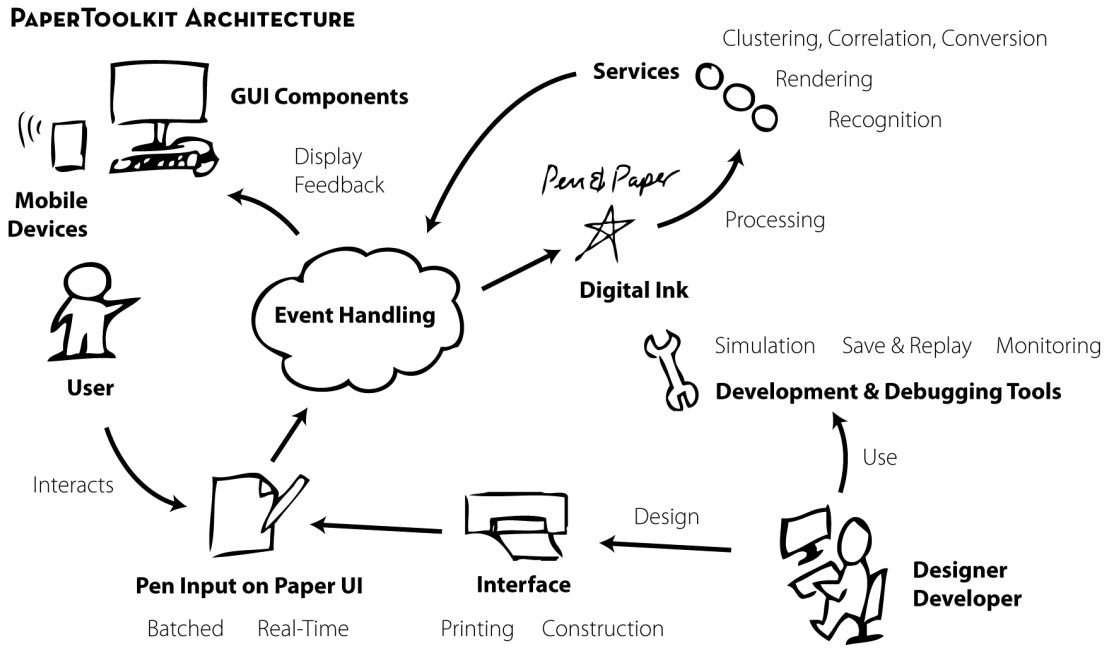


FIGURE 6.5. The PaperToolkit architecture was informed by the designs of ButterflyNet and GIGAprints. The developer can use software abstractions to work with these main modules (*e.g.*, *Ink*, *Sheet*, *Pen*). Visual tools help the developer use the abstractions more efficiently.

6.3 Feedback in Mobile Environments

PaperToolkit differs from previous work by providing abstractions to deal with application output in mobile environments. As seen in CHAPTER 3, paper tools lend themselves to mobile scenarios, as they tend to be lightweight and robust. These situations may require developers to send feedback to multiple users and devices. To integrate paper into this ensemble, PaperToolkit helps developers handle three issues: input from multiple users, graphical feedback, and output to handheld devices.

6.3.1 Multiple Users and Pens

First, programmers need the ability to distinguish between multiple users. PaperToolkit does this by supporting multiple pens, providing a *penID* in the *PenEvent* object. This unique identifier is retrieved from the synchronization data (an Anoto provided identifier), or is automatically generated at runtime when the Anoto-provided ID is not available. Identifiers were also used to detect multiple mice in Hourcade's MID [Hourcade and Bederson 1999] and were used to distinguish between users in Mitsubishi's DiamondTouch table [Dietz and Leigh 2001]. As the toolkit only distinguishes between pens, it does not model situations when users swap their pens. To allow the exchange of pens between users, the programmer will need to support explicit user identification (*e.g.*, a log-in mechanism).

6.3.2 Graphical Feedback

Programmers must rely on a computer to provide feedback to the end user. For paper applications, a common task is to present the digitized ink graphically. PaperToolkit provides GUI components to display *Ink* objects, which contain *InkStrokes*. For example, PaperToolkit includes components such as *InkPanel* (a Java Swing component), *InkPPanel* (a Piccolo component [Bederson, *et al.* 2004] supporting animation), and an *InkDisplay* (an Adobe Flash component). The components use Catmull-Rom splines to display ink strokes (for alternatives, see APPENDIX B.3). Additionally, rendering can use the force information

in the strokes to vary ink thickness. These components also support background imagery, so that ink can be overlaid on top of graphical content, such as Jack's maps.

6.3.3 Interactions with Devices

Applications may need to support communication between multiple devices. For example, BuddySketch provides shared paper sketching during video conferences (see SECTION 5.3). PaperToolkit supports this scenario through the *Device* API, which simplifies communications by abstracting socket communications from the developer. In BuddySketch, each computer participating in the conference is a *Device*. When a user draws on the sheet, the local computer asks its remote peer to update its ink display by invoking an *Action* on the remote computer. This is done through a mobile code approach (*e.g.*, [Thorn 1997]), where action objects such as *OpenURL* or *PlaySound* are serialized to XML and sent to the remote device over TCP. The listening device (also running a full PaperToolkit application, or a smaller *ActionReceiver*) reconstitutes the object and calls *invoke*() on it.

The *Device* API provides the most common actions, and the option to pass arbitrary message objects. The key is that the network details and underlying communication are hidden from the developer, who uses high-level methods such as *displayMessage*, *playSound*, *openURL*, and *readText* (*e.g.*, see Figure 6.2, line 10). The API does not handle the automatic connection between devices, as the first time the *Device* is accessed, a dialog box asks the user to specify a host name for the device. Once connected, the host name is saved for future sessions. Future versions of the *Device* API can leverage ideas for creating ad-hoc connections between computers, as seen in ubiquitous computing platforms such as iStuff [Ballagas, *et al.* 2003] and SpeakEasy [Edwards, *et al.* 2002]. Even without this capability, the device abstraction layer speeds up the prototyping of ensemble interactions.

6.4 Batched versus Real-time Input

Digital pens can operate in two modes (with the Nokia pen, the user specifies the mode by tapping on a paper button). In batched mode, the user works with the pen away from a

computer. The pen saves all input, and allows the user to upload it when he returns to his PC. In streaming mode, the user works near a handheld or desktop computer. The pen sends data to the computer in real-time. A program responds to the pen input by displaying graphics or playing sounds. While PaperToolkit concentrates on real-time interactions, it does support batched input, dispatching events as it would with real-time input.

Batched Input as Ink

When a user plugs in a pen, PaperToolkit reads the ink strokes and saves them to an XML file. Applications can operate on this data, represented as *PenSynch* objects. This subsumes the existing Anoto approach, which makes the ink strokes on pages available to the developer. However, that approach also requires the developer to register her program in the Windows registry, so that it can be notified on every pen synch. To use Anoto's SDK, the developer needs to implement a method to receive the data:

```
// Anoto makes pen data available through an object d which needs
// to be passed to PenRequest's Initialize(data) method
void Anoto.Notification.IDataReceiver.Notify(string c, object d);
```

PaperToolkit improves this process by enabling developers to manually check for data when the application is loaded (instead of having Windows invoke the application through the registry binding). Additionally, the pen data is strongly typed (as opposed to `object d`). This helps programmers (and their IDEs) find the correct methods to use:

```
// PaperToolkit offers strongly-typed methods to ask for pen data
PenSynchManager synchManager = new PenSynchManager();
PenSynch penSynch = synchManager.getMostRecentPenSynch();
List<Ink> importedInk = penSynch.getImportedInk();
```

The example above shows how program code can retrieve digital ink objects from the XML files saved by PaperToolkit. Additionally, running applications can listen for *PenSynch-Events*, which are fired every time a user drops his pen into the USB cradle.

Batched Event Handling

As seen in CHAPTER 4, paper applications can be used both near and away from PCs. Thus, programmers must handle both real-time and batched interactions. Prior tool support allowed developers to receive events in real-time (*e.g.*, [Signer 2006]) or ink data in batched mode (*e.g.*, [Anoto AB 2007, Guimbretière 2003, Signer 2006]). Our experience shows that users can benefit from *event handling from batched input*.

For example, a biologist using the ButterflyNet notebook is frequently away from a PC. Here, a batched architecture enables the software to process handwritten notes when the user docks his pen to a computer at the field station. However, while working in the field, the biologist can use a pen gesture to link a photograph to a place in his notebook. When his smart camera recognizes this action, it provides *immediate* audio and visual feedback to acknowledge the linking gesture. In the ButterflyNet notebook, this was accomplished as two separate applications. The smart camera's software recognized gestures in real-time, logging events to a file. The software that ran on the PC would take these events and align them to photographs. While functional, this fractured implementation makes code difficult to read and more difficult to maintain.

PaperToolkit helps developers collocate event handling code for batched and real-time events. To do this, it provides a flag in *PenEvents* so that data received through the wireless connection can be distinguished from data received via the dock. In event handling code, the developer checks the *isRealTime*() flag to provide appropriate feedback. To support this, when a pen is docked, PaperToolkit reads the data and injects events into the application's event stream.

Besides the creation timestamp and the flag, batched and real-time events behave equivalently. This architectural feature enables developers to present feedback appropriate to each situation, facilitates clearer code organization, and provides developers a way to test in real time applications that will be deployed for batched usage.

Calibrating Batched and Real-time Coordinates

Ideally, the pen hardware reports identical coordinates in batched and streaming modes. Unfortunately, the particular streaming pen we use (Nokia SU-1B) reports streamed data in physical Anoto coordinates (absolute X and Y values), but reports batched data as logical Anoto coordinates (relative to a page). Since the transformation function has not been published, PaperToolkit requires a one-time calibration for developers who wish to coordinate real-time and batched interactions. This information is saved for future sessions. For details on the calibration procedure and the coordinate conversion needed to unify batched and real-time input, see Appendix B.2.

6.5 Working with Digital Ink

The core architecture allows a program to capture real-time or batched input from a digital pen. This input is sent through the corresponding event handlers, and can be stored internally as *Ink* objects, which contain a list of *InkStrokes*. The strokes in turn comprise *Samples* (X, Y, time, and force). SECTION 6.3.2 discussed ways to render digital ink to GUIs. However, many applications depend on higher level operations on digital ink. PaperToolkit provides abstractions for recognition, correlation, clustering, and coordinate conversion.

6.5.1 Recognition

To provide value beyond the GUI-like interactions (taps, drags, *etc.*), PaperToolkit supports handwriting and gesture recognition. The *GestureHandler*, *MarkingGestureHandler*, and *HandwritingHandler* can be added to any region on a sheet. *GestureHandler* provides single-stroke template-based recognition, and is an implementation of the $1 recognition algorithm [Wobbrock, *et al.* 2007]. The marking gesture recognizer calculates the direction of the stroke, and fires an event reporting one of eight compass directions. The *HandwritingHandler* communicates with a *HandwritingRecognitionService* is built on Microsoft's Tablet PC handwriting recognizer.

The recognizers can be used separately from the toolkit (without the event architecture and handlers). Any application can pass an *InkStroke* to the *DollarRecognizer* to receive a recognition result. The *HandwritingRecognitionService* accepts *Ink* objects as input, and returns the top result as a string.

With any recognition algorithm, ambiguity is a core concern. The program must determine what the user intended, and it will be incorrect some percentage of the time. PaperToolkit does not explicitly model ambiguity in the handlers, and this is a current limitation. The underlying recognizers do understand ambiguity, however. The dollar recognizer returns the single best template with a confidence score, and it can be modified to return a sorted list of the TOP N matches (with confidence scores). Likewise, the handwriting recognizer can return the ten best results through the `getAlternatives()` method. For more techniques on how an interface might model and mediate ambiguity, see Mankoff's dissertation [Mankoff 2001]. A future version of PaperToolkit could incorporate such techniques. For example, if the dollar recognizer's top result does not meet a minimum confidence threshold, a GUI widget might ask the user to choose from the potential matches.

6.5.2 Correlation

Developers often wish to correlate incoming ink strokes with other media (*e.g.*, photos, audio, GPS logs). ButterflyNet does this by reading all the timestamps of media and placing entries in a database. When the user flips a page, the system processes the page of ink strokes to determine the beginning and ending timestamps, which are used to construct a SQL query. The database returns a list of related content, which is then displayed by the browser.

PaperToolkit simplifies this with abstractions for media correlation. Instead of asking the developer to inspect and coordinate timestamps directly, it allows direct queries for ink strokes and files based on input timestamps (avoiding the need to set up a database for simple programs). For example, assume Karen is working with an *Ink* object from a page of notes. To find the handwriting closest to the time a particular photo was taken, she can use

the `getStrokesClosestToTime(ink, targetTimestamps)`. This method returns the single stroke with a timestamp nearest to `targetTimestamps`. To query in the other direction (*i.e.*, to find *photos* with timestamps near a chunk of ink) the developer instead uses `getObjects-ClosestToTime(objectsWithTimestamps, targetInkTimestamps)`.

The same can be done with other metadata facets, so long as a *distance metric* is defined for the metadata. PaperToolkit supports correlation by timestamps and GPS location, but allows developers to specify arbitrary metadata through `ink.addMetaData(…)`. Currently, the toolkit does not provide support for calibrating clocks between different devices (*e.g.*, pen, camera, and computer). This step would be necessary for fine-grained associations.

6.5.3 Clustering

To support correlation and to enable end-user direct manipulation of groups of ink, PaperToolkit contains abstractions to cluster digital ink in time and space. Different clustering algorithms (*e.g.*, K-MEANS and MEAN SHIFT) and approaches (*e.g.*, hierarchical and lattice), offer distinct tradeoffs. For example, K-MEANS requires the developer to specify K, the number of clusters to look for. In practice, simple clustering based on a developer-specified time or space window works well. *SpatialClustering* and *TemporalClustering* implement this simple clustering, and both accept *Ink* objects and return *InkClusters*.

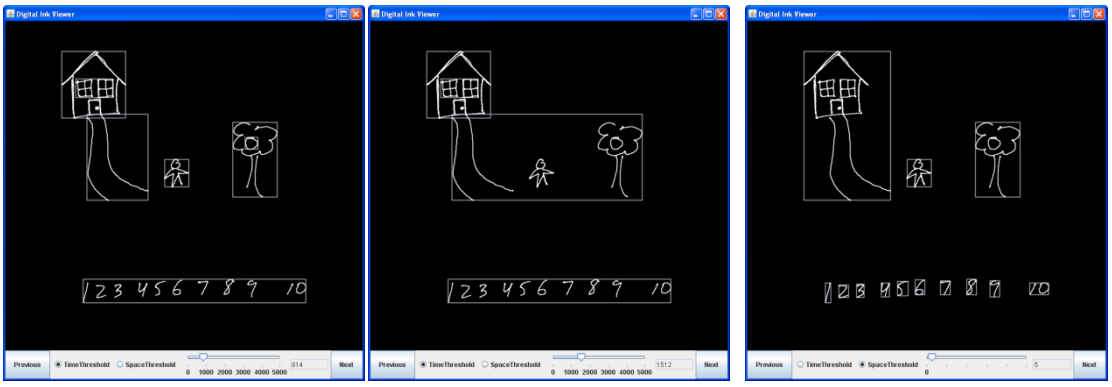


FIGURE 6.6. A visual tool allows developers to compare different clustering approaches. *Left*) Temporal clustering with an 800-millisecond window. *Middle*) Temporal clustering with a window of 1.5-seconds. *Right*) Spatial clustering with a 5-pixel window.

One contribution beyond prior work is the ability to visualize these clusters in the *InkClustersPanel*, a GUI component that allows a user to see and reposition groups of digital ink. The toolkit also provides a development tool based on the *InkClustersPanel* (see Figure 6.6). With it, developers can tweak the time and space thresholds interactively to see the results of different clustering approaches. This helps the developer choose between the available options and tunable parameters. This tool reflects our larger approach of providing visual tools to help programmers understand and work with the software abstractions.

6.5.4 Coordinate Conversion

When handling input, program code may have to convert incoming ink strokes to other coordinate systems. Mapping applications deal with paper coordinates (inches), the device's screen coordinates (pixels), and world coordinates (GPS latitude and longitude). This differs from desktop programming, as GUIs handle all coordinates in screen coordinates (pixels). Since most interactions are handled through GUI event listeners, coordinate conversions are rarely needed in GUI code (unless the application is a WYSIWYG editor that translates screen graphics to printer graphics).

To facilitate conversions, PaperToolkit includes abstractions for different units, and simple conversions between those units. For example, paper UI components can be laid out

using *Inches*, *Centimeters*, or *Points* objects. When displayed on screen, those units can be converted to *Pixel* values through a `unitsObject.toPixels()` method call.

However, some transformations depend on further context. For example, GPS transformations depend on the area of the map that is shown on the page. PaperToolkit supports these generic mappings with a *CoordinateConverter*, which maintains a mapping between input and output coordinates. The program can populate the mappings with examples. Then, the application can convert between the two coordinate systems by interpolating and extrapolating from the internal mappings.

6.5.5 Extending Event Handlers

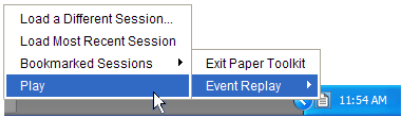
To provide custom handling for pen interactions, the developer can extend the abstract *EventHandler* class, which provides a notification to its subclass on every incoming pen sample (from 30 to 75Hz, depending on the model of the pen). For example, to make a handler that invokes different behavior based on the force of a pen tap, the *ForceHandler* subclass can inspect the force value of incoming *PenEvents*. (No system in this dissertation uses the force information outside of ink rendering, as in practice, the force of the pen tip is difficult for users to control.)

FIGURE 6.7. The developer's workflow contains steps that are often repeated. For example, a bug will result in further development and testing. Tools that speed up this loop can make developers more productive (*e.g.*, rapid testing allows developers to try multiple designs).

6.6 Replay of Pen Input

At its core, PaperToolkit is a set of software abstractions for creating paper + digital applications. However, we can improve the workflow (see Figure 6.7) with techniques to address aspects of developing paper applications (*e.g.*, printing). This broadens the toolkit by adopting ideas from traditional programming environments.

For instance, we observed limiting factors to iterative testing. Most notably, physically providing pen input on each test run presents problems in efficiency and the ability to replicate bugs. PaperToolkit addresses this by allowing replay of pen input (inspired by replay of Java programs [Lewis 2007, Steven, *et al.* 2000]). This offers several advantages. First, the input is consistent between trials and between testers. This helps when testing recognition algorithms, and enhances the ability to reproduce bugs. Second, replay makes it more efficient to reproduce long input sequences for testing or demonstration. Without replay, it would be frustrating to write a paragraph of text as test input, only to encounter a bug that requires a restart. For demonstrations, the developer can replay a log file to show the various features of an application. Finally, replay enhances collaboration. If pen hardware is limited, one developer can generate all the test input, and share log files with teammates.



PaperToolkit logs pen input from running applications, and saves each session to a file in the *EventData* folder. The developer can replay sessions through the system tray menu of the running application. She can also bookmark sessions, to return to them in the future. To share a session, she emails the associated log file to a teammate.

Save and Replay Architecture

During a test session, the *InputDevice* abstraction communicates with the underlying hardware to receive raw coordinates, and sends the live data to the *EventDispatcher* (see Figure 6.8). In the background, the *InputDevice* logs all incoming data to the file system. When the developer selects replay, the most recent log data is fed back to the *InputDevice* abstraction. The *EventDispatcher* dispatches the incoming data to the *Handlers*, as if it were coming from actual hardware. When the developer initiates replay, the system waits several seconds before beginning (to allow the developer to position any GUI elements). Long gaps in the replay log are compressed to two seconds, to speed up replay sessions.

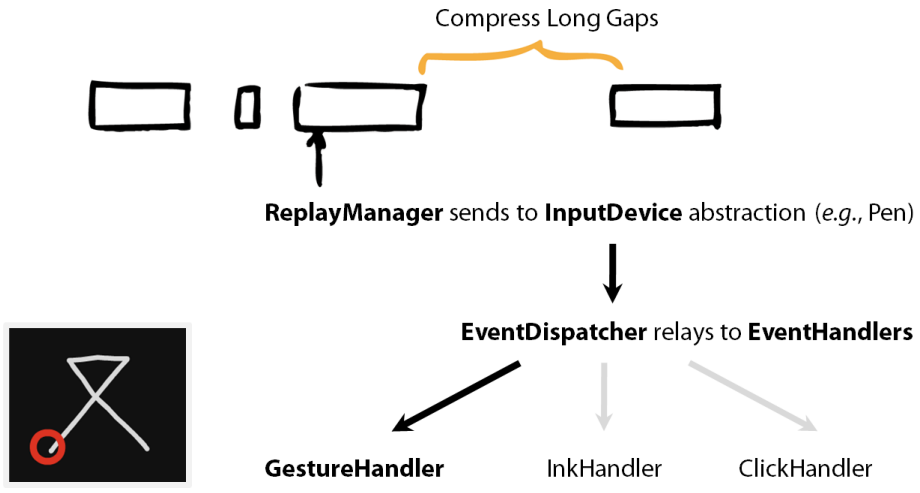


FIGURE 6.8. The *ReplayManager* reads the log file and sends it to the *InputDevice* (software abstraction for the pen). This data is reported to the *EventDispatcher* as if it had come from the hardware. Long gaps (> 2 seconds) within the replay data are compressed to two seconds.

Sending Events to the Dispatcher

A runtime-controllable event dispatcher enables the unified handling of batched and real-time input (SECTION 6.4) and the replaying of pen input (SECTION 6.6). This feature is similar to what the Java Robot API uses to send mouse input to the operating system [Sun Microsystems 2007]. But rather than treat it as an API for programmers, we mainly use it to support the two toolkit features.

For example, when a replay log file is loaded into memory, it is represented as a list of *PenSamples*. When the programmer invokes a replay, the list is read and sent to the available pen *InputDevices*. These objects communicate with the *EventDispatcher* as they normally would, creating *PenEvents* that the dispatcher sends to *EventHandlers*.

Building the Application around User Input

An important benefit of replay is that it allows developers to capture user input before much of the application is built (similar in spirit to test-driven development). Recall the scenario in SECTION 6.1.1, where Karen is building an augmented notebook application. Karen first designs a bracket gesture for excerpting handwritten notes to send to a blog. Then, with very little code, she creates a skeleton application (with empty handlers) that captures notes and pen gestures. She then provides sample input, including the pen gesture. Finally, as she fills in the handlers, she can replay the test case multiple times to watch the application develop.

Lessons Learned

Because of the time savings it offers, we use save and replay for all of our development; external researchers have also commented that it has made testing more convenient. However, we have learned some lessons that may help other toolkit designers.

Currently, replay only works for real-time pen input. The replay menu should also make batched data available for replay. This would be a small extension to the current tools, and would not require changes to the architecture (the *ReplayManager* can treat the batched data as a different type of event log). Additionally, the tool does not allow developers to

choose portions of the test session to replay. This would have impact on sessions that depend on external variables, such as network latencies, or if the user switches between pen and GUI input. Currently, the toolkit assumes the log can be replayed from start to end.

Also, the replay logs are named by the time and date of the session. The developer can add more descriptive names, but a better solution would be a quick preview tool. The developer could browse through a small multiples visualization of the event logs, to help her select from the collection.

6.7 Implementation Details

The core of the toolkit is built on Java SE 6.0. However, a number of the services are built on other platforms, to make use of hardware, different GUI environments, printer output, and recognition engines. This section provides an overview of the major implementation decisions, to inform future toolkit designers.

Pens and Devices

Batched pen input is handled through a Microsoft .NET 2.0 component, as Anoto synchronization is provided through native Windows code. The pen monitor uses Anoto's SDK to detect when the user drops his pen into the USB cradle. Data is saved to an XML file representing the synchronization. The toolkit dispatches this data to running applications.

For flexible real-time pen input, each digital pen (we use Nokia SU-1B) is handled by a *PenServer*, which receives coordinates from the serial port. The coordinates are sent to a corresponding *PenClient*, which lives on the computer running the paper application (most commonly, the same computer hosting the *PenServer*). The client sends coordinates in real-time to the application runtime, which finds and invokes the corresponding event handlers.

Having a server that abstracts the underlying pen hardware enhances visibility during debugging. The developer can *telnet* into this server to view raw data; or, a tool can monitor and log the data. Additionally, this opens up support for remote collaboration, because a digital pen can be hosted on a computer at Stanford, but send data to a computer in New

York. The toolkit hides these details by automatically starting and stopping servers and clients. The average developer does not need to know that *Pens* comprise a server/client pair.

The toolkit models *Devices* in a similar fashion. The *Device* abstraction comprises an *ActionReceiver* and *ActionSender* that is normally hidden from the developer. However, as the design is flexible, the developer can use this server/client pair separately from the toolkit. The *ActionReceiver* waits for messages, and is usually deployed on a handheld or other remote computer. Whenever a local computer wishes to communicate with that handheld, it connects using an *ActionSender*, and dispatches an *Action* object (which can represent actions from opening URLs to playing sounds to reading text out loud using text-to-speech).

Handwriting and Gesture Recognition

The recognition services are accessed through event handlers (*HandwritingHandler* and *GestureHandler*), but their underlying recognizers can be replaced. Currently, the handwriting handler uses Microsoft's Tablet PC recognizer, and the gesture handler uses Wobbrock's $1 recognizer [Wobbrock, *et al.* 2007].

Microsoft's recognizer works well for most research prototypes. However, when it is used to recognize ink written on *paper*, the results are not nearly as good as when it is used to recognize ink written on a Tablet PC. We hypothesize that users modify their writing behavior on a tablet: because applications show handwriting results in real-time, they can adjust their handwriting to perform better. On paper, there is no real-time feedback from the recognizer: the user writes more naturally, but at the expense of good recognition. We could perhaps apply ink normalization (*e.g.*, [Simard, *et al.* 2005]) to preprocess the strokes.

The $1 recognizer works well as a single stroke gesture recognizer. However, developers still need to use heuristics to determine which ink strokes to send to the recognizer. The naïve approach is to send an entire page of notes to recognizer. However, this would result in poor performance and a large number of false positives (as the recognizer always provides a result). A better approach is to designate an area for gestures (*e.g.*, the margin of a notebook) or assume that gestures strokes are larger than handwriting (*e.g.*, a big bracket

amidst a page of words). An entire page of ink strokes can thus be pruned to a few potential gestures, which can be sent to the recognizer.

Printing

For rendering paper interfaces, PaperToolkit uses Postscript and PDF. The sheets and dot patterns are rendered using the Java EPS [Mutton 2007] and iText PDF libraries [Lowagie and Soares 2007]. To communicate with printers programmatically, the toolkit's printing API builds on the Java print service, hiding some of its complexities.

Flash Integration

Most PaperToolkit applications provide graphical feedback by including a Java Swing GUI in the ensemble. However, it is important to support Adobe Flash as an alternative, because many modern web applications are built on the Flash platform (and thus many programmers are familiar with it). As a result, PaperToolkit provides integration with Flash GUIs through Adobe's Flex platform, which renders interfaces to the SWF format. The *Ink* classes are mirrored in Flash, through ActionScript 3 classes. The Flash UI communicates with the toolkit through the *ExternalCommunicationsServer*. Simple messages are passed as strings over the socket connection. Complex objects are serialized to XML and reconstituted as ActionScript or Java objects.

Replay and Monitoring

The toolkit provides multiple entry points to support advanced services, such as replay and monitoring. For example, the *EventDispatcher* allows an application to programmatically dispatch pen events. Upon replay, the toolkit reads the log file, constructs *PenEvents*, and sends them to the *EventDispatcher*.

The *ToolkitMonitor* allows programs to watch the toolkit during runtime. The SideCar monitoring interface described in CHAPTER 8 uses this feature to track toolkit actions.

Using Independent Services

Many of the toolkit's components (*e.g.*, Pens, Devices, Flash integration) are implemented as client-server pairs. This provides flexibility, as these services can be used separately from the main toolkit. For example, a developer can use the handwriting server to recognize input from a Wacom tablet. However, the added flexibility comes at a cost in performance and code complexity, as the toolkit must hide the components from the developer. When a paper application exits, the toolkit must make sure that these services exit properly (*e.g.*, by sending an *exit* message to the server). This is most apparent when a developer is debugging, and is using the *HandwritingRecognitionServer*. Since it is implemented as a .NET service, killing the Java process (developers often do this when using the Eclipse IDE) will prevent the toolkit from sending the *exit* message. In this case, the server must be closed separately (through the system tray icon the handwriting server provides).

Alternate Architectures for Paper + Digital Applications

At the lowest level, PaperToolkit is a convenient way to communicate with the pen hardware in batched and real-time modes. We used these low-level services to experiment with a GUI-like architecture for paper applications. However, other architectures and programming models are possible. PaperToolkit's services can be used to experiment with alternative programming models for augmented paper. In other toolkits, abstractions such as the classes for digital ink, correlation, and clustering would still be applicable.

For example, one could design a toolkit around the idea of *media correlation*. This toolkit might not use event handlers, and would treat incoming media (*e.g.*, ink and photos) as distinct streams with facets of metadata (*e.g.*, time and space). The core abstractions would allow fast and flexible queries between data streams.

Applications like ButterflyNet would benefit from specialized support for searching for related content. However, this hypothetical toolkit would not support real-time pen gestures as well (*e.g.*, ButterflyNet's bracket gesture).

6.8 Summary

The PaperToolkit architecture provides a low threshold and high ceiling for developing pen-and-paper applications. The architecture dispatches high-level events for digital pen input (both in batched and real-time modes), and supports application feedback through output to devices (the *Actions* API). PaperToolkit addresses a wide range of applications in the paper + digital design space (see Figure 6.9). It enables collaboration by supporting input from multiple pens, and maintains flexibility by taking a modular approach for implementation. For example, the handwriting recognition server (one of multiple servers in the toolkit) can be used separate from the *HandwritingHandler*. The toolkit does not provide explicit support for overlaying projector output on top of prints, and does not deal with haptic feedback. However, it does provide special support for output to nearby devices, through the *Device* API. The next chapter presents our mixed-methods toolkit evaluation.

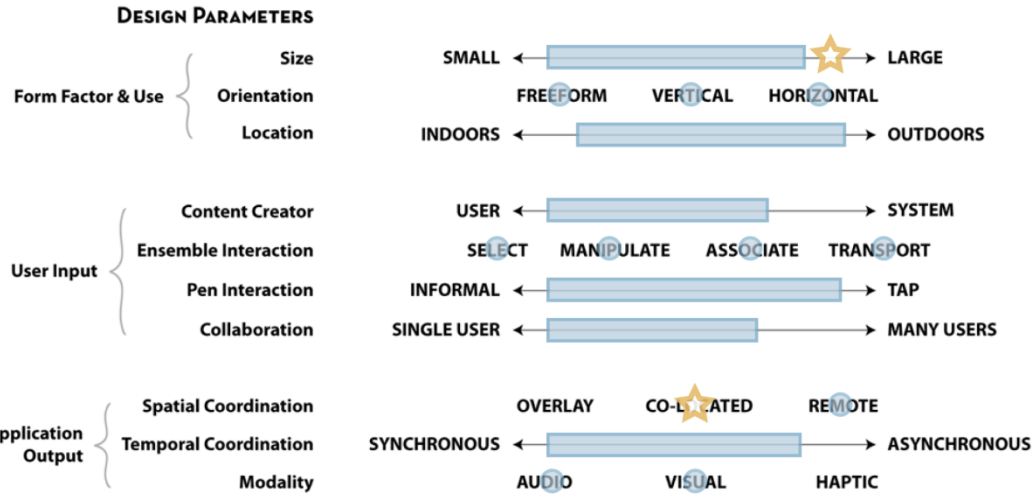


FIGURE 6.9. PaperToolkit supports a wide range of paper + digital applications, and provides special support for large interfaces and application on nearby devices.

7

PaperToolkit Evaluation

This chapter presents a mixed-methods evaluation of the core abstractions in PaperToolkit. The evaluation included three main parts: most notably, the toolkit was made available (open-source) to an HCI class at UC Berkeley. Over six weeks, the 17 teams created projects ranging from maps for location-based search to notepads for blogging. The deployment revealed that PaperToolkit provided a low-enough threshold for novices to create a wide variety of working applications.

Second, we analyzed the source code of resulting projects to determine which aspects of the toolkit helped most, and which features it lacked. The analysis revealed that developers needed extra debugging support and better abstractions for adding gesture recognition into their projects.

The third part of the evaluation was long-term use in our lab, to understand how *experts* could use the toolkit as a research platform explore the paper + digital design space.

This deployment (to research assistants in our lab) suggests that the toolkit is an enabler for novel augmented paper research. Three of the paper + digital projects have been presented as research posters at the UIST and UBICOMP conferences [Bastéa-Forte, *et al.* 2007, Bernstein, *et al.* 2006, Jiang, *et al.* 2007].

7.1 Overview of Methods

This section overviews the evaluation methods, and provides a roadmap for the rest of the chapter. The core architecture of the toolkit was inspired by the event-based architectures found in GUI toolkits, such as Java Swing, Windows Forms, and Adobe Flex. Thus, one main question to answer was whether a *novice to the toolkit* could easily pick up the abstractions (especially if they had to use a GUI toolkit at the same time). To do this, we deployed the toolkit to the CS160 class at UC Berkeley (Fall 2006). The 17 teams (69 students) used the toolkit for the last six weeks of their semester to create working paper + digital projects. Our observations are detailed in SECTION 7.3. We then collected and analyzed the source code of the 17 projects. The goal of this analysis was to determine the parts of the toolkit that benefited students most, and to identify any important missing features. This analysis is detailed in SECTION 7.4.

It is also important to understand how experts can benefit from the toolkit abstractions. We report on usage by research assistants in our laboratory, who used it to create novel research integrating pen and paper with digital tables (*e.g.*, [Dietz and Leigh 2001]). The toolkit has also been used by external research labs. This use by experts is described in SECTION 7.5.

SECTION 7.6 discusses the lessons we learned, and summarizes design implications for future augmented paper toolkits. Several of these implications are addressed in a subsequent design response iteration, which is presented in CHAPTER 8.

7.2 Background

While user-centered design of software tools is still a nascent field, there is some precedent to draw upon. Many of these projects begin by conducting a field study of the target population of software developers. For example, Landay surveyed professional designers of graphical user interfaces to discover that they used paper to sketch storyboards during the early stages of design [Landay 1996]. Similarly, Ko began his work on WhyLine for Alice by conducting an observational study of seven Alice programmers [Ko and Myers 2003]. He found that programmers asked why/why-not questions about unexpected/expected program behavior. Klemmer also began his work on Papier-Mâché by interviewing researchers in the tangible input space [Klemmer 2004].

An inspection technique for assessing programming tools and API usability is the cognitive dimensions framework [Green and Petre 1996]. We use this method (which does not require users) to complement the long-term deployment in our toolkit analysis.

7.3 Deployment to an Undergraduate HCI Class

To understand how PaperToolkit could support developers over a longer term, we made it available to students in the CS160 class at UC Berkeley during the fall semester of 2006 (the author did not teach the class). The 69 students (17 teams) developed interactive paper applications with PaperToolkit; we provided the stable core of the toolkit, which consisted of UI construction and event handling (SECTION 6.2.1) and handwriting recognition. Their development began in the eighth week of a 14-week class, after they had designed and tested paper prototypes [Rettig 1994]. The author held in-person sessions to answer questions and receive feedback, folding these observations into later iterations of the toolkit.

7.3.1 Projects Built

It is notable that 17 teams of students with no prior experience in building paper interfaces were able to build working projects in less than six weeks. PaperToolkit's event-based approach offered a low threshold for students who had programmed GUIs before. However, many students learned GUI programming as part of this introductory course (*e.g.*, a team said that "none of us had developed event-driven programs prior to this project"). For those students, the similarity to Java Swing meant that they did not have to learn two vastly different architectures.

Project topics varied, including paper-based web design, personal organizers, and sharing tools for news and blogs (see Figure 7.1). We now describe a few of these projects, and illustrate how each used PaperToolkit (see Figures 7.2-4).

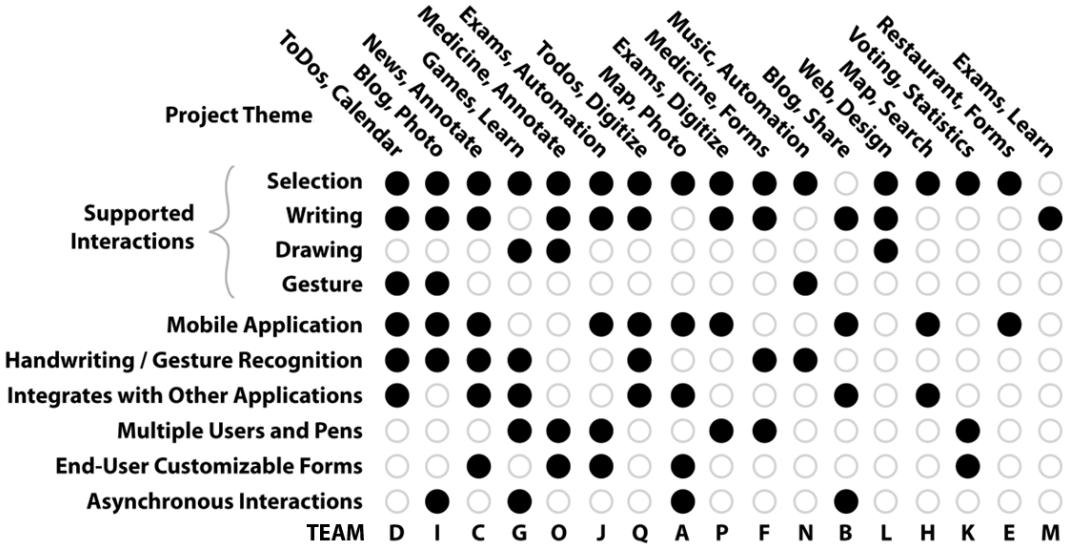
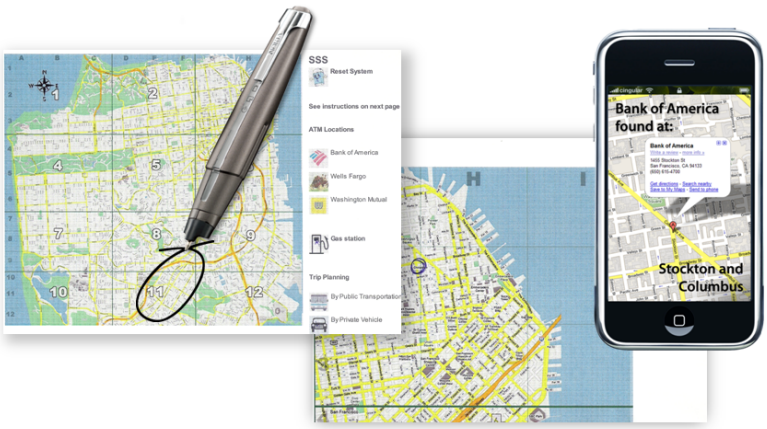


FIGURE 7.1. The 17 projects (named A-Q) covered many themes. The deployed PaperToolkit supported selection (*e.g.*, check a box) and writing operations well. However, informal interactions requiring recognition were notably less common (*e.g.*, draw a musical note). We have since improved support for gesture recognition. Ten projects were mobile, and seven integrated with existing applications in mash-up fashion. Four supported batched input, where ink is processed after the user returns to his PC.

Seven teams created applications by connecting interactive paper with web services or existing desktop applications. This mash-up approach works well because it leverages pre-built functionality. Six teams integrated web applications into their projects, either by scraping HTML, or using established APIs (*e.g.*, Flickr and Google Calendar). One group created a Firefox plug-in. From this, we learned that modern toolkits should facilitate data transfer with web services, and provide support for connecting to existing desktop applications. PaperToolkit supports this goal by making Ink objects available in web-friendly formats (*e.g.*, XML, JPEG, PNG), and by allowing GUI feedback through Adobe Flash.



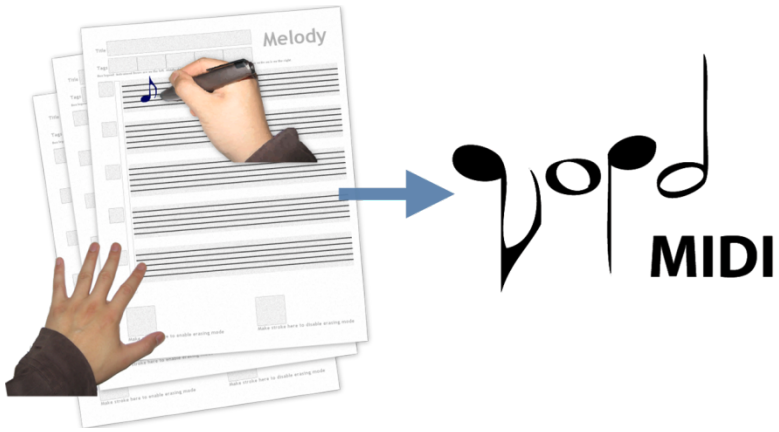
Team H's location-based search sends text messages to the user's mobile phone.

Team I's paper blogger lets a user compose entries away from his computer.

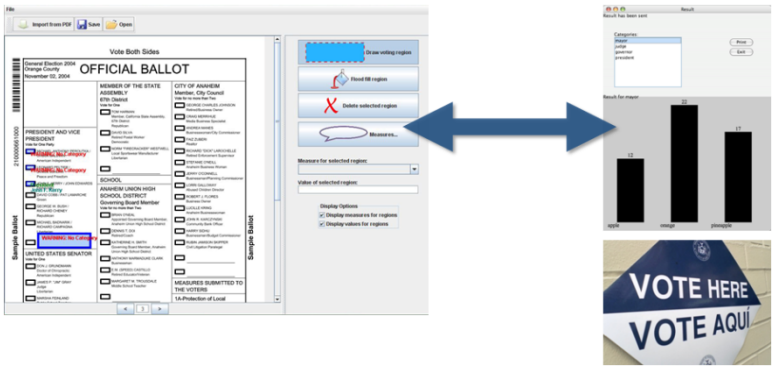
FIGURE 7.2. Two of the 17 class projects: map-based search and paper blogging.

Second, we observed a need to equate the programming models for real-time and batched input. Our initial goal for the toolkit was to study real-time event dispatch for paper applications, since prior work had concentrated on batched input. Thus, the deployed toolkit only supported event handlers for real-time input. Batched input was made available through *Ink* objects (to maintain consistency with the underlying Anoto model).

Operating in batched mode—where data resides on the pen until it is uploaded—eases the deployment of mobile applications by eliminating the need for a nearby PC. However, though 10 teams created mobile applications, only three of these teams took advantage of the batched support. This may be because 1) real-time input is better for repeated test sessions, and 2) event handling is easier to understand (compared with iterating over ink

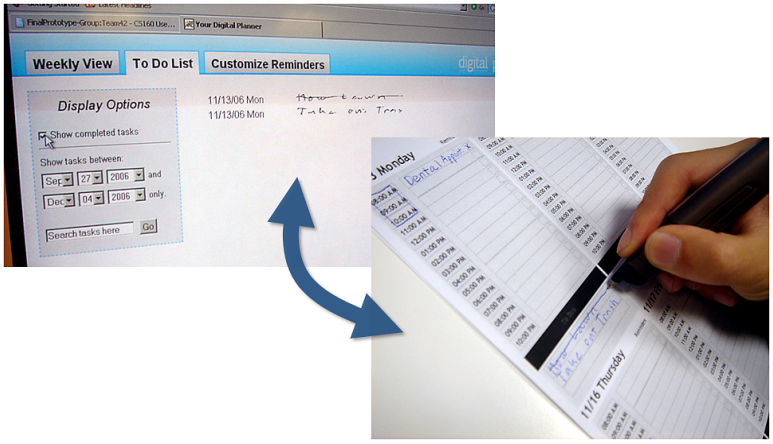


Team N's music project translated handwritten notes into MIDI sound files.

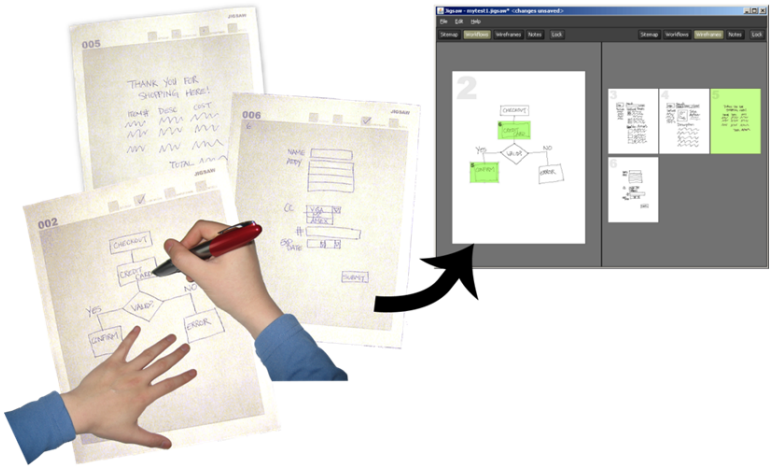


Team K's augmented ballots tally votes automatically while maintaining a paper backup.

FIGURE 7.3. Two of the 17 projects: music recognition and automatic vote tallying.



Team D's paper planner synchronizes with a web calendar.



Team L's web design environment supports real-time capture and control of the digital view.

FIGURE 7.4. Two of the 17 class projects: a synchronized paper/web calendar and a web design environment.

strokes). We have since unified support; real-time and batched inputs now dispatch events identically.

Team D created a paper planner that would synchronize with a web calendar (Figure 7.4, top), part of the class of applications where the paper interface mirrors the digital interface (*e.g.*, PADD [Guimbretière 2003]). *Team N*'s music project automatically converted handwritten music into MIDI sound files, part of the class of applications where informal input is recognized to automate tedious tasks.

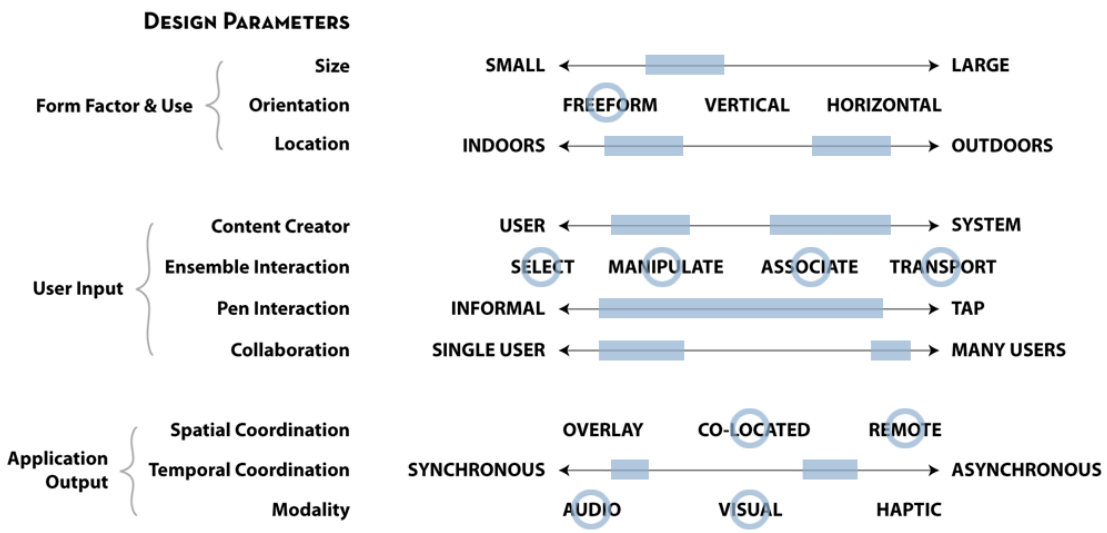


FIGURE 7.5. The student projects covered a wide part of the design space. The map search allowed selection interactions, while the paper blogger provided manual association of photographs. Input ranged from circle-gestures through musical notes to freeform blog entries. While most projects supported single users, some were designed for multiple users (*e.g.*, the augmented ballots). Students did not provide large prints, or overlaid graphics, because they did not have access to wide-format printers or projectors.

7.3.2 Coverage of the Design Space

The 17 projects covered a wide spectrum of the design space. For example, the six applications in Figures 7.2-4 support a range of interactions and use cases (see Figure 7.5). However, none of the applications include large prints, as the students did not have access to a wide-format printer. While four of the applications were designed to support asynchronous processing (music, planner, blogger, ballots) all but the blogging team tested with real-time input, because at the time, the toolkit only supported synchronous event handling.

7.4 Analysis of Source Code Produced by Students

This section reviews the insights we gained by using source-code analysis to inform toolkit design. Examining code produced by developers offers an empirical account of usage patterns and provides information about the usability of the architecture and API. We reviewed the students' 304 source files (~35K source statements / ~51K lines of code with

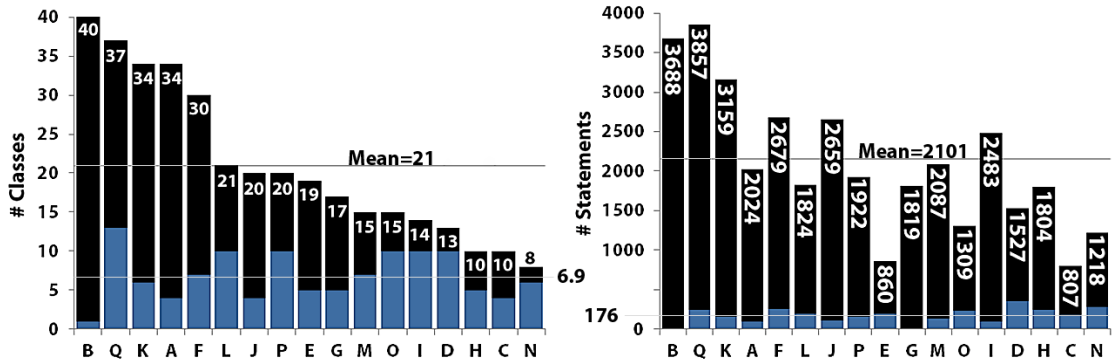


FIGURE 7.6. With PaperToolkit, student teams created substantial projects in six weeks. The average project comprised 21 Java classes and 2012 source statements. The size and complexity of projects lends weight to the external validity of our insights. The light blue bars show the portion of projects directly instantiating objects or calling methods in PaperToolkit. While a substantial portion of *classes* accessed the toolkit, only a small number of *lines of code* were needed to effectively use the paper + digital capabilities.

comments) through several methods. First, we searched through and examined files by hand. Second, we calculated statistics about the code by writing scripts to process the files. We recorded observations for each file, and grouped recurring themes (concerning how students created and debugged their applications). The next sections detail these patterns.

7.4.1 Toolkit Usage

We examined usage of the toolkit to understand the impact it had on each project, and to see which parts of the API were most important. In Figure 7.6, we see that the toolkit abstractions enabled students to create interesting paper + digital interactions with very few lines of code: the light blue bars show the fraction of each project that used PaperToolkit classes and methods. We see that with about 200 lines of code, students can receive input from the digital pen, process it, display it to the screen, and more.

We also examined the distribution of classes that projects used (see Figure 7.7). The most used classes (left side) are for UI construction (*Region* and its subclasses, *Inches*, *Sheet*). If we concentrate on making this part of the API more clear and concise, we can reduce the lines of code needed, and possibly reduce the bugs developers will encounter.

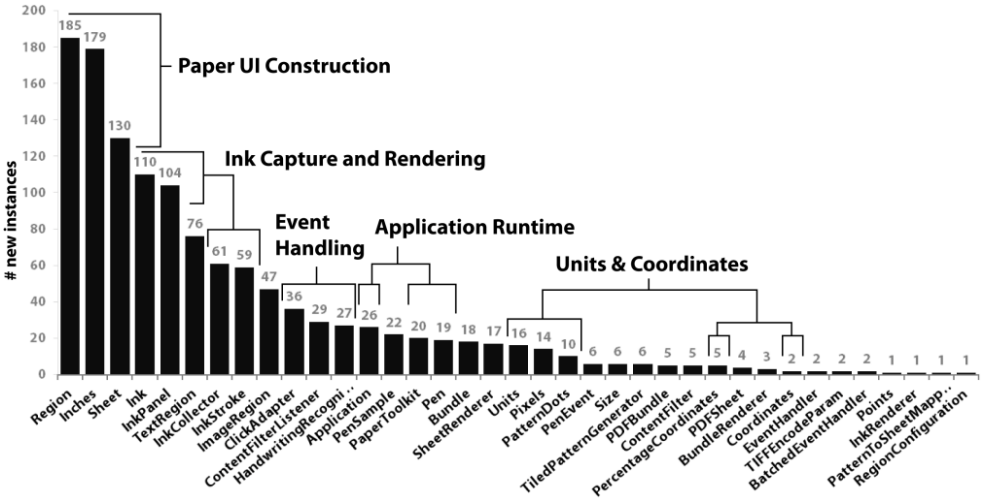


FIGURE 7.7. This shows how often teams used the toolkit classes (# instances declared). The heavily used classes are for constructing paper UIs (*Region*, *Inches*, *Sheet*). Classes for manipulating ink were also important (*Ink*, *InkPanel*, *InkCollector*, *InkStroke*). Seldom used classes were newer features that were less well-documented. We can have a large impact if we improve the API of classes on the left (*e.g.*, by making names easier to remember). To raise the tail, we might add documentation and examples.

Examining the tail of the distribution may reveal classes that were too difficult to use, or hidden by lack of documentation and examples. For example, *BatchedEventHandler* was only used twice, as the deployed toolkit focused on real-time handling (in documentation, examples, and implementation). From this result, we improved batched handling support.

We suggest that toolkit designers can gain insight by examining usage and reflecting on why certain methods and classes are or are not used. However, by itself, this data may not be sufficient, as there can be multiple explanations for a class being on the left side of the distribution (verbose vs. valuable) or the right (useful yet specialized vs. hard-to-use). The toolkit designer may wish to talk to developers or cross-check the results with additional observations before making API revisions. For example, from using the toolkit, we found that the prevalence of *Inches* was due to verbose constructors; we streamlined UI construction by adding constructors that assumed a default unit of measure (see SECTION 8.7).

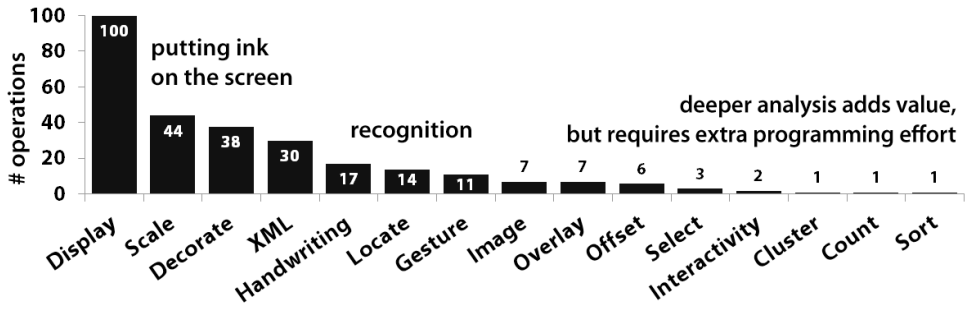


FIGURE 7.8. This shows the different operations on digital ink discovered in the project source code. These *operations* may consist of multiple Java statements. While display and scale were common, it took extra effort to compose solutions for recognizing gestures and clustering strokes. Making it easier to explore these operations will raise the tail of this curve for future projects. Applications will benefit from the more powerful and flexible interactions.

7.4.2 Composing Heuristics for Gesture Recognition

Several of the projects included some form of pen-gesture recognition. As deployed, PaperToolkit only included a marking handler and a handwriting recognizer. Examining how developers approached recognition might point toward opportunities to improve the toolkit beyond its current ceiling. As digital ink is a core concern of paper applications, we gathered data on all *operations* on digital ink. For the purposes of this analysis, an operation is a group of code that strives toward a particular goal, such as the recognition of a gesture.

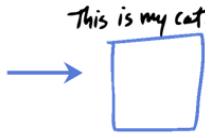
The distribution of ink operations (see Figure 7.8) largely reflected the functions available in the toolkit. However, it is clear that students desired more support for gestures, as they designed their own methods to cluster, sort, and recognize ink strokes. In the figure, we see that the projects displayed, scaled, and decorated ink, but seldom clustered or sorted ink strokes, perhaps because the deployed toolkit did not include many abstractions to analyze digital ink strokes.

Only four teams implemented recognition of pen gestures (via heuristics). *Team D* detected when users crossed out handwritten text, and updated a web planner to reflect the completed task. *Team G* recognized paper-based games (*e.g.*, tic-tac-toe). *Team I* detected boxes users had drawn in a blog entry, and helped users import photos into those areas. *Team*

N recognized handwritten music, including whole, half, quarter, and eighth notes, and translated the composition into MIDI sound files.

The following snippets provide examples of the heuristic-based recognition that the teams used. In *Team I*'s blogging project, the user inserts a picture into an entry by drawing a square with a single stroke (inspired by ButterflyNet's bracket gesture). To determine the bounds, the team iterated through the strokes to find the one with the largest area:

```
getPointForPhotoInsertion() {
  for (ink : inkList) {
     for (strokes : ink.getStrokes()) {
        for (s : strokes) {
           if (s.getArea() >
maxArea) {
              maxArea =
s.getArea();
              boxInkStroke = s;
              boxInk = ink;
              ....
```



Later, they test the box against a size threshold, and position the photo at its boundaries. Similarly, *Team N* applied heuristic measures to strokes to determine the note durations:

```
if (timeOfStroke <= shortTimeThreshold) {
   if (heightOfStroke <= shortHeightThreshold) {
      out.print("Whole note in Key of ");
   } else {
      out.print("1/2 note in Key of ");
   }
} else {
   if (firstY > lastY) { // if the note is going up
      if ((lastY - minY) > 5) { // detect flag
         out.print("1/8th note in Key of ");
      } else { // 1/4 note
   ....
```

This algorithm compares strokes to temporal and spatial thresholds, and detects their direction. An eighth note is recognized when the last ink samples are written in a direction opposite to the main stem (detecting the note's flag). These examples show how teams approached recognition by composing simple thresholds and ink statistics. Heuristics are straightforward to specify and can be robust for many interactions. However, a template-based recognizer may perform better as more gestures are added to the system.

We have since added new methods to search, measure, and cluster digital ink. Additionally, we have included Wobbrock's single-stroke gesture recognizer [Wobbrock, *et al.* 2007]. Developers can use the heuristics to select a subset of input to the full recognizer, without requiring the user to enter and exit a gesture mode.

Tools can help the developer by allowing them to compose and test different recognition approaches rapidly. For example, a developer might provide some sample input, and see which of four recognition approaches does best. We can also improve toolkit support for recognition by including more convenience methods for selecting strokes in space and time, clustering strokes, and visualizing the results.

7.4.3 Creating Interfaces by Example Modification

We observed that developers copy from their own code and from code found on the web to speed up programming. In particular, we found evidence that developers would copy UI classes, paste them into their project, and then modify the skeletons to grow their application around the working base.

To find out what code developers copied, how much they would copy, and from where, we used a combination of analysis methods. First, we used MOSS [Schleimer, *et al.* 2003], a tool traditionally used to detect plagiarism in software, to detect similarities between projects and the toolkit. While MOSS worked for comparing projects fed to it, it was not suited for *locating* clones of code residing on the web. As a result, we manually identified clones in the corpus, noticing that unusual comments and identifiers were effective in pointing out copied code. Once we found a plausible clone, we searched the web and the corpus to establish the source. This dissertation presents the first work that uses static code analysis and search to study copy-and-paste behavior in assessing toolkit usability.

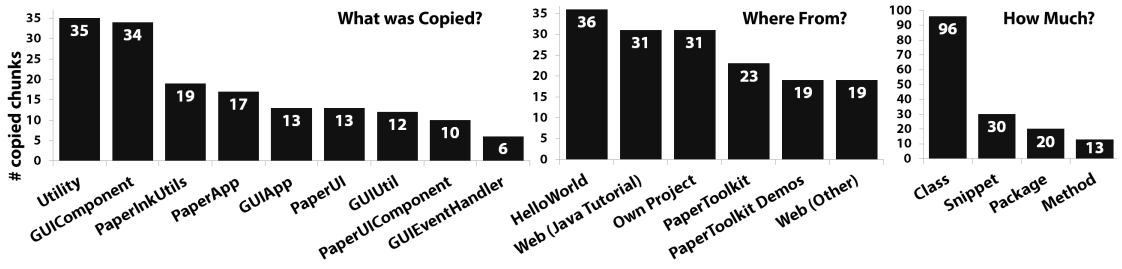


FIGURE 7.9. Teams used copy-and-paste to facilitate programming. *Left*) Developers most often copied and customized snippets for UI construction (paper and graphical), and frequently reused utility methods. *Middle*) Developers selected code from their own project and earlier *HelloWorld* because the code was both familiar and likely to work. *Right*) They would copy an entire class file to get a program working, and then grow it iteratively. This suggests that programmers can benefit from tools that embrace copy and paste. Tools can provide more templates, track code lineage, and facilitate the refactoring of copied code.

Developers copy to reduce the boilerplate code they need to write, and when encountering problems they (or others) have probably solved before. Our data show that 41% of the 159 chunks of copied code directly supported GUIs, and 37% supported paper UIs (see Figure 7.9). Code clones came from several sources, including a Hello World application that they had worked through, the Web (*e.g.*, Java Swing tutorials), and PaperToolkit. In 96 of the 159 observed instances, developers copied an entire class rather than methods or code snippets. They would then modify the class to fit their application. These findings are consistent with earlier studies in programming practices (*e.g.*, [Kim, Bergman, *et al.* 2004, Rosson and Carroll 1996]) that discovered that programmers copy blocks of method calls to save time. We find that these observations still apply for paper + digital programming. Developers copy code for constructing UIs, or when working with new APIs, to reduce errors using an unfamiliar interface framework.

A toolkit can take advantage of this approach. Instead of just presenting an API, a toolkit should integrate example code into documentation, and provide running examples. Alternatively, we could construct an API browser to allow developers to visually inspect different features of the toolkit, and to copy the necessary snippets of functionality into their applications (see SECTION 8.7). In PaperToolkit, we have attempted to provide better online documentation with working code examples.

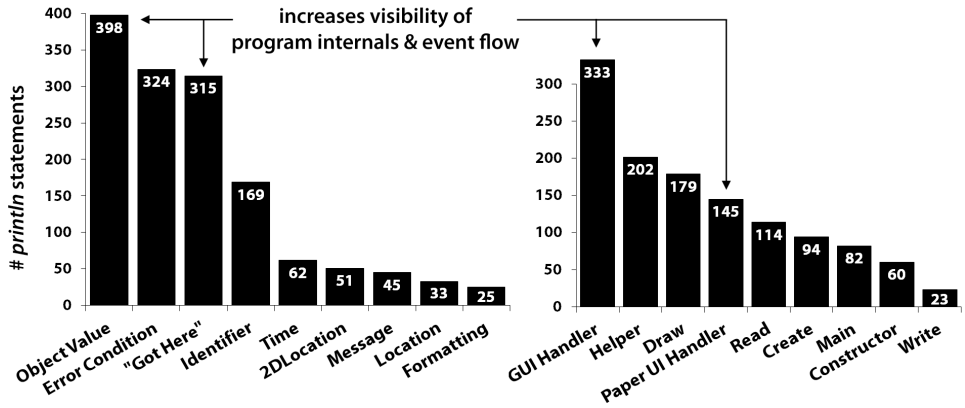


FIGURE 7.10. Analysis of the 17 projects reveals that most debugging statements (*println*s) helped programmers visualize object values particular to the application. Other statements signal when the program encounters an exception, or gets to a particular location. Developers place most of the *println*s in handlers (GUI and PaperToolkit). This suggests a need for tools to help monitor objects and handlers at runtime.

7.4.4 Tracking Event Handlers through Debug Output

The analysis shows that developers used many *print* statements to debug their programs. To discover where the statements were used most, we searched for instances of debug output. As this study was conducted outside our lab, we did not log interactions with the IDE or debugger. The corpus contained 1232 print statements. We annotated each to see where the statement was located and what was printed (see Figure 7.10).

Overall, 39% of all debug statements (478/1232) were found directly inside event handlers; 333 were in GUI handlers, and 145 in PaperToolkit handlers. The top three printed values were *object instances* particular to the program, *error* messages, and *got here* messages serving only to signal that a code block was reached. When combined with where statements were located, we find that half of all GUI handler print statements, and a third of PaperToolkit handler print statements were of the *got here* type. These statements served different roles. They sometimes acted as stubs, to track unimplemented handlers:

```
System.out.println("Zoom In");
// TODO Auto-generated Event stub actionPerformed()
```

As the handler was filled in, debug statements are then used to help developers visualize the program internals, and how it responds to pen input. Overall, this shows that programmers try to visualize their program's (possibly hidden) behavior to help monitor for bugs.

Tools can help developers understand when events are fired, and what object values are at runtime. Developers tend to use print statements first, because they are lightweight when compared with breakpoints, variable watchpoints, or even more full-featured tools that allow forwards and backwards code-stepping (*e.g.*, ZStep [Lieberman 1984] and Omniscient Debugger [Lewis 2007]). Print statements can signal warnings at runtime without stopping (or significantly slowing) execution. They are manifest in the source code, and thus easily shared between teammates.

This is not a new observation, as researchers have previously observed the continued use of *print* statements as the primary form of debugging (*e.g.*, [Ko and Myers 2004, Lieberman 1997]). However, we show that the observations still apply in this genre of programming, and suggest that tools can help augment the practice of using *print* statements (see SECTION 8.3). If a tool can maintain this lightweight property, provide the benefits of a debugger, while reducing the time needed to understand event flow, it can substantially speed up the debugging of applications.

7.4.5 Managing Multiple Coordinate Representations

Finally, we observed two common tasks: converting between different coordinate systems (*e.g.*, from paper to GPS) and mapping stroke locations to different semantic representations (*e.g.*, musical tones). In this study, two projects used maps, and needed methods such as:

```
convertPaperToEarthCoordinates()
getGoogleMapTileCoordinate(lat, lon, zoomlevel)
getStreetIntersection(penX, penY, pageNum)
```

Conversions do not always produce numerical results: `getStreetIntersection()` invokes a database query that returns the names of cross streets in San Francisco. Likewise, the music project interpreted ink strokes as locations on a staff, converting them into musical notes. In *Team O*'s medical imaging project, code existed to align ink to images. In *Team M*'s

flashcards project, they needed to rotate digital ink to handle when users wrote on the back of a card (upside down with respect to the defined region).

The computer graphics community has provided coordinate transformation functions in their libraries for years (*e.g.*, see SECTION 5.4 of [Foley, *et al.* 1995] and CHAPTER 3 of [Woo, *et al.* 1999]). However, our findings suggest that libraries may not be enough; tools can help developers learn and manage these conversions (*e.g.*, by providing visualizations to help understand the different affine transformations or semantic mappings).

7.5 Exploring New Paper + Digital Interactions

PaperToolkit has supported work beyond the UC Berkeley class deployment. For example, researchers in our laboratory have used it to support research projects in augmented paper. This section describes the projects, and provides details on how each demonstrates successes and limitations of the toolkit. The first three projects integrate paper with digital tables, which are tables that can sense the touch of one or more users. In this work, we combine the use of digital pens and paper with the DiamondTouch table [Dietz and Leigh 2001]. The last two projects relate to field notebooks, since much of our work is still inspired by biology practices. These two deal with media tagging and retrieval around the paper notebook. Diamond's Edge used a preliminary version of PaperToolkit (supporting real-time pen input and printing) while the latter three used feature-complete versions.

Diamond's Edge—A Collaborative Design Environment

Two students, Michael Bernstein and Avi Robinson-Mosher, created a tabletop design environment [Bernstein, *et al.* 2006] that used PaperToolkit to capture writing from multiple users, render ink to a digital canvas, and send drawings to a printer. For example, two designers can sit around a table and draw ink sketches on a paper notepad. One designer can then use a pen gesture to circle his sketch, and flick it from the notebook onto the digital table. The sketch is now projected digitally onto the table. The second designer can drag the sketch with a finger onto his physical notepad. Any edits on the second notepad are

mirrored onto the original notepad (by overlaid projection). When the designers are happy with their sketch, they can send it to a printer (see Figure 7.11).

The researchers used an early version of PaperToolkit to capture real-time pen input. A key module was that some code needed to transform pen coordinates into coordinates the projector could display (from Anoto dots to pixels). The project also modeled simple physics, so that a sketch would rotate about its pivot as it was dragged by a user's finger. This meant that the code needed to keep track of the transformation, so that it could invert it later (to mirror edits between one location and another).

When inspecting the source code, we find that the individual methods to keep track of the transformations contain *very few lines of code*. Thus, it may not seem beneficial to introduce a new abstraction for coordinate transformations.

However, comments in the Diamond's Edge source code suggest otherwise—that in fact, *determining the correct transformation* is challenging, even with the availability of Java transformation functions. For example, we found a comment next to the code that transformed a coordinate from a remote sketch back onto the original notebook:

```
// Finally fixed! Can't believe how easy this was… after 7 hours
//   of debugging.
// This will reflect the drawings back in the original,
//   appropriately undoing the rotation and translation.
```

In our own experience, managing different coordinate spaces is cumbersome. When laying out a print, we think in physical measurements (*e.g.*, inches). However, the pen reports coordinates in 0.3-mm dots (Anoto units). To render ink in a GUI (*e.g.*, for a projector), coordinates must be in screen coordinates (pixels), and must fall into the displayable range

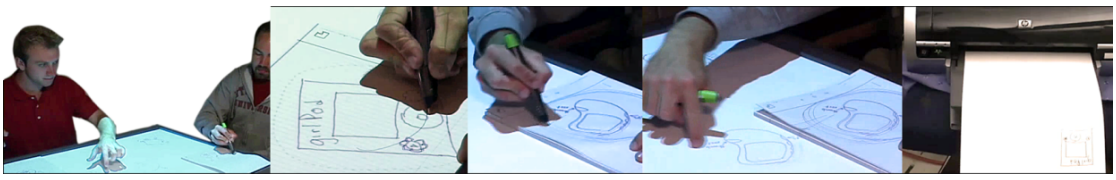
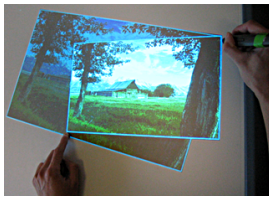


FIGURE 7.11. From *Left to Right*) Two users can sketch collaboratively around a digital table. A designer can circle a sketch, flick it with his pen onto the table, and manipulate the sketch using his fingers. Finally, the designers can print their collaborative design.

(*e.g.*, 1024×768). A toolkit that requires developers to operate in different coordinate spaces (due to multiple devices) should help developers calculate the correct transformations between those spaces. Even if the resulting transformation is just three lines of Java code, it may take much effort to determine the translation, scaling, and rotation, and then successfully debug the code. An augmented paper toolkit should support this through a *CoordinateTransformation* abstraction, and a tool to visually debug the transformations.

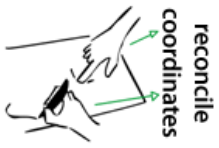
Pointer—Fluid Gesture Input from Multiple Devices

Later projects used a version of PaperToolkit that supported event handlers and regions. The second student project, by Marcello Bastéa-Forte, explored flexible input using digital pens and fingers on a digital table [Bastéa-Forte, *et al.* 2007]. To allow the pen to be used on the table, we printed a large sheet of patterned paper with one patterned region (66×50 centimeters) and placed it on top of the table.



This application supported the manipulation and annotation of photographs. A user can set a pivot point with his pen and rotate/zoom a photograph around that pivot with his other hand. The PaperToolkit handlers located the pen while writing, rotating, and zooming. This project also taught us the importance of abstracting input devices. PaperToolkit now allows developers to simulate the pen with a mouse.

While pen input was physically collocated with finger input, each device reported coordinates that had to be reconciled. The toolkit facilitated coordinate transformations by reporting coordinates relative to the large region (and thus, the table surface). As these coordinates were in percentage units (0 to 100), the researcher worked with all coordinates in the same units.



DigiPost—Using Physical Post-its to Annotate Digital Maps

Hao Jiang, a visiting researcher, created a project to allow users to write annotations on sticky notes and send those annotations to a digital map on the table [Jiang, *et al.* 2007]. The user does this by touching one finger to the map, through the Post-it note. DigiPost accepted input from multiple users (by distinguishing pen IDs), recognized the handwritten input, and associated the resulting text to a map displayed on the table. Cross-out gestures deleted annotations.

Each sticky note was treated as a different region. All regions contained identical event handlers, which used the *HandwritingRecognizer*. The application distinguished between users by the pen's identifier, accessed through the *PenEvent*. Gestures were recognized using a simple heuristic. If a stroke captured by the *InkHandler* matched one of the thresholds—from one corner of the note to an opposite corner—it was recognized as a *cross-out gesture*. The digital annotation would be removed from the table.

For research prototypes, heuristics are often better than template-based recognizers, because they are simple to specify and debug—a developer can quickly prototype and test the technique with users. Of course, heuristics are not as scalable as full recognizers, and as a gesture set grows, it will become cumbersome for a programmer to include new heuristics. An ideal toolkit would support both heuristics and template-based recognizers.

VTags—Searching Virtual Tags with Paper Maps

With Nikhil Raghavan, we explored techniques for biologists to share digital annotations attached to the physical world. VTags is a system for sharing location-tagged annotations to the web. To enable remote collaborators to search this database, a paper map supports pen-based queries. To perform a search, the user circles a location on the map; this gesture specifies the center and the radius of the search in a single motion. PaperToolkit's *CoordinateConverter* translates locations on the map to GPS coordinates. The developer adds a few examples (hand-determined using Google Maps), and the converter uses these samples to interpolate/extrapolate the transformation function. The map contains a *DragHandler*, which provides information about the bounding box of the stroke. The percentage coordinates are sent through the *CoordinateConverter*, which returns a GPS latitude and longitude. These are passed as parameters to the web query; results are returned in a web-page, to be viewed either on a desktop or a handheld display (see Figure 7.12, left). This project demonstrated the utility of the coordinate conversion abstraction, and the ability of pen-gestures to simultaneously specify multiple input parameters (search center and radius).

Media Finder – Associating Media to Informal Sketches

Jürgen Steimle, a visiting researcher, created the Media Finder, which allows a biologist to use informal sketches as a way to organize and retrieve photographs (see Figure 7.12, right). The biologist first draws a sketch (*e.g.*, a map) on paper, and later drags photographs onto

FIGURE 7.12. *Left*) With VTags, a biologist can circle a paper map to retrieve geo-tagged photographs. The results are displayed in a web page. *Right*) Media Finder allows a biologist to associate photographs to parts of sketches drawn on paper. The photos can be retrieved by tapping on the notebook, or by selecting from a digital browser.

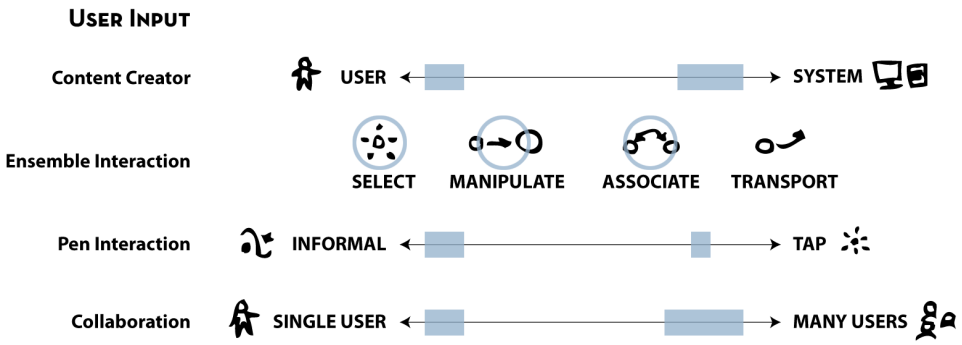


Figure 7.13. The table projects were targeted at multiple users, while the field maps were single-user applications. The applications support both user-generated annotations and system-provided maps. The table projects allowed manipulation of content, while the maps supported associations and selections (*e.g.*, location-based queries).

different parts of the map (using the digital browser). He can then retrieve photos by tapping near different ink clusters, on paper or in the digital interface.

This project taught us the importance of providing better tools for manipulating digital ink. In particular, this project needed clustering algorithms and GUI components that could display (and allow end users to select) the clusters.

Coverage of the Paper + Digital Design Space

The five student projects covered a wide range of the design space. Many of the projects supported informal inking from the user (see Figure 7.13). The table projects provided ensemble interactions that spanned the pen and table. For example, with Pointer, a user can use both his hand and pen to zoom, rotate, and otherwise manipulate a photograph.

With regard to application output, the table-based projects explored overlaid graphics, while the field maps provided co-located feedback, on a nearby device (see Figure 7.14). Overlaid graphics simulates the affordance of electronic paper; the current use of a projector prohibits mobile applications from using overlay. None of these projects explored audio or haptic feedback.

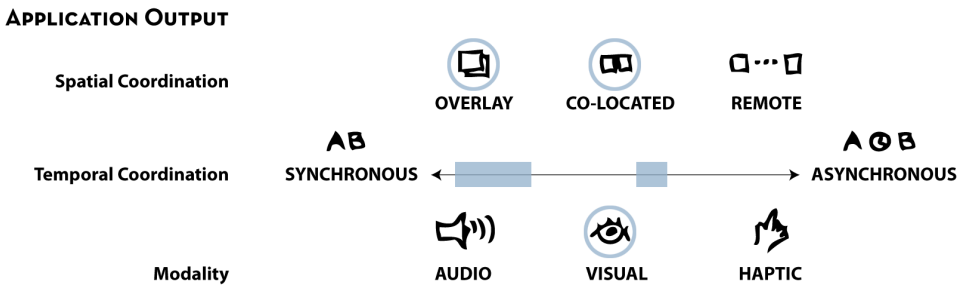


FIGURE 7.14. The three digital table projects provide overlaid graphical output, while the two field maps provide output on a nearby handheld or monitor. All applications provide real-time feedback, but the Media Finder allows input in the field that is processed asynchronously.

7.6 Design Implications

From this evaluation, we have distilled implications to inform the reader of design opportunities for future paper + digital applications and tools. The first two are primarily concerned with how the user works with paper interfaces. The last two deal with techniques to streamline the workflow for creating and testing paper + digital applications.

Interaction Language

Many of the applications in this chapter supported informal ink input and pen gestures (as opposed to form-based input and interaction with paper widgets). These informal interactions allow for flexible and efficient input (*e.g.*, a circle gesture simultaneously specifies the radius and center of the search). However, this can present problems when an end user encounters the paper UI for the first time. GUIs have built up a language of interaction through buttons, sliders, and drop-down menus, but paper UIs have not. Interface widgets translate well (*e.g.*, paper buttons respond to pen taps). But pen gestures are not as obvious. To specify a location on a map, a user might tap, circle, or mark an ×. As a result, applications must embed textual or graphical directions into the UI to communicate these gestures (*e.g.*, see Figure 7.15, from the Palm OS Graffiti manual). In the student projects, we noticed frequent use of

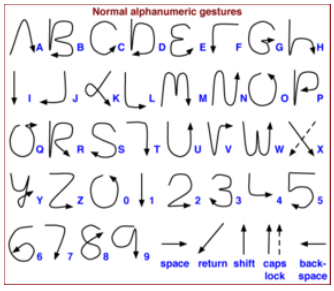


FIGURE 7.15. Palm's Graffiti manual demonstrated how to invoke each gesture with one or two stylus strokes.

TextRegions to provide directions to the user. This is an opportunity for tool support. A toolkit could automatically generate these directions, and include graphical icons of the recognized gestures next to the active regions.

Ordering Constraints

A related issue lies in enforcing interface constraints. In a GUI, developers can gray out a component when it is not appropriate, directing users toward an ordering of interactions. On paper, one can never prevent a user from checking a box or turning the page. Developers must account for this intrinsic limitation by providing clear directions, and dealing with incomplete or out-of-order input. In the class projects, students used text labels to direct the user's interaction. Future augmented paper platforms might help by determining the validity and completeness of input before further processing (*e.g.*, by using state machines for event handling, as in [Appert and Beaudouin-Lafon 2006]).

Supporting Iterative Design

A barrier that developers face when designing paper UIs is the speed of iterative design, development, and testing. In the class deployment, students were reluctant to modify the interfaces, as they took too long to print. Repeated testing was also laborious (the deployed version did not yet support replay). Students were also limited by hardware; each team had one digital pen. Tools should support the design process by enhancing design and testing.

Visual Tools

At its core, PaperToolkit is a set of useful abstractions for the developer. Programmers still need to learn the language of the toolkit, by reading documentation and working through example programs. We suggest that there is an opportunity to provide visual tools to support textual programming. By *visual tools*, we do not mean *visual programming*, but rather tools to make hidden aspects of an application visible, so that *textual programming* can be more efficient and less error prone. In the past, visualizations have helped people understand algorithms and data structures (*e.g.* [Stasko*, et al.* 1998]). In the future, visualizations may

help programmers manipulate and produce code faster, or locate and fix buggy statements. We take some steps toward this goal in CHAPTER 8.

7.7 Dissemination and Broader Impact

The ideas in PaperToolkit have been disseminated in several ways. First, PaperToolkit is an open-source project, hosted at http://papertoolkit.googlecode.com. Researchers are free to download and use this software. Second, we host information on creating applications with PaperToolkit on http://hci.stanford.edu/paper. From August 2006 (when the site first went live), to October 2007, the site and related pages have received 23,836 page views, with more than 25% of occurring on the tutorial and documentation pages. Considering that using this toolkit requires a considerable investment (several thousand US dollars to buy the Anoto SDK, with digital pens and notebooks), these statistics suggest the success of our approach. Finally, we have given public talks on PaperToolkit at Intel Berkeley, IBM Almaden, Yahoo Research, the Stanford HCI Seminar, and over video conference to the Universidade de Fortaleza (Ceará, Brazil) as part of World Usability Day, 2006.

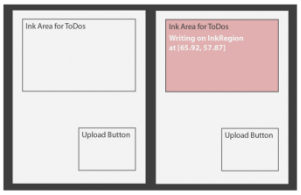
Beyond the dissemination of ideas, PaperToolkit has also supported research around the world. Tabard, Eastmond, Mackay, and others have used PaperToolkit at In|Situ| Lab in Paris to help capture handwritten notes from laboratory biologists. Ehlen and Stimatze at the Computational Semantics Lab at Stanford have used it to monitor real-time note taking in their experiments on team meetings. Küstermann *et al.* at the University of Siegen has developed a customizable television remote for interactive TV applications. Steimle at the Darmstadt University of Technology has created an eLearning system that allows students to share handwritten notes and bookmarks with each other and with the instructor.

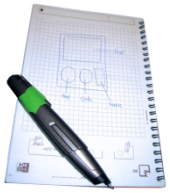
7.8 Summary

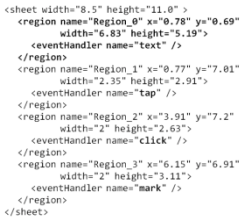
This chapter presented the mixed-methods evaluation of PaperToolkit. In many ways, the findings contributed to features in the current version of the toolkit. For example, replay was introduced during the class deployment, since we observed that testing was a bottleneck

to rapid iterative development. Further longitudinal use would help to determine whether the architecture and tools will have long-lasting impact. CHAPTER 8 details our newest designs *in response* to this evaluation, a step not often taken with toolkit research. Admittedly, some of the material in the next chapter is speculative. However, we include the material to provide other researchers possible avenues for future research.

8







Design Response

The long-term evaluation suggests that novices find the toolkit approachable, because the GUI-like architecture softens the initial learning curve. Experts benefited from the flexibility and variety of tools and services (*e.g.*, the pen and action servers). However, the evaluation also exposed challenges that developers face while creating and debugging applications. For example, some programmers find it difficult to maintain a mental model of the application, and this hinders debugging. Following iterative design principles, we improved the toolkit based on the evaluation results; this chapter presents the design responses.

The first section reviews the high level goals. We then describe development techniques to lower the design threshold, enhance visibility, and speed up testing. These techniques center on tools that the programmer can use throughout her workday, and fall into the categories of sketching, monitoring, simulation, and replay. Finally, we present

improvements to the toolkit abstractions and API. Each design is compared with a related system, and analyzed through the cognitive dimensions framework [Green and Petre 1996].

8.1 Overview

The design responses explore opportunities to address development and debugging challenges that we observed. The findings come from the toolkit evaluation (17 teams and six expert researchers), our own work (two substantial paper + digital projects), and an observational study of three developers (including the author). We distilled four goals.

The first goal is to lower the toolkit's barrier of entry, since it takes much time to become acquainted with a large API. PaperToolkit's primary audience is the GUI programmer, and this is reflected in the evaluation (the HCI class consisted of juniors and seniors in computer science). However, lowering the threshold will allow non-programmers such as artists and designers to participate in the design. SECTION 8.2 describes a method to simplify the process of generating paper interfaces.

The second goal is to increase the visibility of toolkit internals, since we observed that developers relied on *print* statements to debug event flow in the paper applications. SECTION 8.3 introduces a design to capture and present application events. This helps a developer maintain a better mental model of her application, and eliminates the need to use *print* statements to track events.

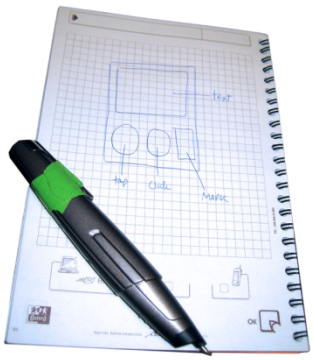
Third, we explore several ideas to enhance testing. Building upon the replay functionality, we can allow developers to replay and share *portions* of test sessions (SECTION 8.4). Additionally, since printing is slow and testing can become tedious, we now allow developers to simulate paper interfaces without printing them (SECTION 8.5). We also explored a debugging technique centered around application monitoring, rather than the traditional method of instrumenting source code (SECTION 8.6).

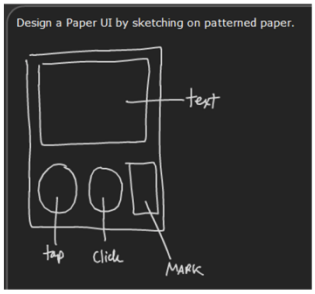
Finally, as expert developers work directly with the PaperToolkit API, they can benefit from improvements to the abstractions, class names, and method calls (SECTION 8.7). We show how we iterated the software abstractions in response to the evaluation.

8.2 Creating Interfaces by Sketching

The source-code analysis revealed that much of the code in applications dealt with paper UI construction (the top three most-used classes in Figure 7.7). This is not surprising, as a prior study showed that GUI code comprises 50% of applications [Myers and Rosson 1992]. The first design response, then, is to reduce the amount of code developers need to write when creating paper UI components and event handlers. This lowers the threshold for novice users, and can speed up the workflow for experts.

We developed a technique to allow a designer to turn a paper sketch into the code that generates the interface (see Figure 8.1). This helps novices learn the patterns for organizing the paper UI, and enables those without any programming experience to create interfaces. The technique supports a simple visual language: the largest box becomes a *Sheet*; internal shapes become *Regions*; lines that exit a region and the sheet are turned into *EventHandlers*. Any text written next to a handler is sent through handwriting recognition and matched to a handler implementation in the PaperToolkit library (*e.g.*, *text* is matched to *InkHandler*). The result is saved as an XML representation of the interface, which is read in at runtime.






```
<sheet width="8.5" height="11.0" >
  <region name="Region_0" x="0.78" y="0.69"
          width="6.83" height="5.19">
    <eventHandler name="text" />
  </region>
  <region name="Region_1" x="0.77" y="7.01"
          width="2.35" height="2.91">
    <eventHandler name="tap" />
  </region>
  <region name="Region_2" x="3.91" y="7.2"
          width="2" height="2.63">
    <eventHandler name="click" />
  </region>
  <region name="Region_3" x="6.15" y="6.91"
          width="2" height="3.11">
    <eventHandler name="mark" />
  </region>
</sheet>
```

FIGURE 8.1. This design lowers the barrier of entry for constructing paper UIs. *Left*) Users sketch a low-fidelity prototype. *Middle*) The system captures and displays the sketch. *Right*) It regularizes the input into rectangles, and outputs a working XML specification of the interface. An improvement beyond prior work [Landay 1996] is that the approach of writing handler names scales as new handlers are added to the toolkit. The architect only needs to add a mapping from the name the designer writes (*e.g.*, "marking menu") to the underlying class (*e.g.*, `papertoolkit.handlers.MarkingMenuHandler`).

Thus, an interface designer can lay out paper interfaces and specify handlers without writing a single line of code. However, there are some limitations. First, this gesture language specifies the layout and the handler, but not what happens when a person interacts with the handler. A programmer still needs to fill in code that specifies the *behavior*. Future revisions could specify behaviors by example (*e.g.*, by recording macros or recognizing sample input). Second, layout is currently too precise, as the sizes and locations of the components are calculated from the sketch (*i.e.*, a sheet may be interpreted as 8.5×9.2-inches when the designer intends to draw a letter-sized sheet). Future versions should snap components to common sizes, and allow the developer to apply constraints (*e.g.*, this region should be 0.5 inches from the lower right corner).

Though this technique was developed for paper, it can also be used to generate graphical interfaces. This approach was first explored by Landay in SILK, which allowed interface designers to sketch interactive prototypes on a digital tablet, because "one of the biggest drawbacks to using paper sketches is the lack of interaction possible between the paper-based design and a user" [Landay and Myers 1995]. Our technique combines this interaction with real paper; designers can design on paper *and* receive the benefits of SILK. This technique goes beyond SILK by introducing the ability to specify handlers by writing *names*; this provides a level of flexibility beyond the shape-based widget recognition in SILK. With this scalable technique, handwritten names are mapped to the underlying implementation. As new handlers are added to the toolkit, new name → handler mappings can be introduced. To verify that this technique works for graphical UIs, we redirected the gestures to map to Java Swing components so that sketches created simple frames with text fields and buttons.

Analysis of Sketching Paper UIs

We can apply the *error-proneness* cognitive dimension to show that sketching is indeed a better alternative than coding paper UIs by hand. Since the *system* generates the UI code, there will be fewer errors that the user needs to debug, at least in interface construction. However, the current design also depends on recognition of the handler names. This can be

a drawback because a direct manipulation paper UI builder would be more exact (*i.e.*, the designer can choose a handler by clicking on an icon instead of writing its name).

When compared with GUI builders, it is clear that the sketching solution has some limitations. Interface builders score higher on *visibility* and *secondary notation*, because they show developers all available components and functions. With the sketching solution (and with Landay's SILK) the designer must learn the gestures or names that the tool can recognize. These controls are hidden. An improved solution might feed a freeform sketch into a standard UI builder, where unrecognized components can be overridden through the direct manipulation interface of the UI builder.

The sketching solution allows designers to avoid *premature commitment*, because they can design different configurations on paper before exporting the paper UI. However, once exported, there is no easy way to modify the design by editing the sketch (high *viscosity*).

8.3 Tracking Event Flow through a Visualization

The evaluation revealed that developers found it challenging to maintain the mental model of their paper application. Many times, developers used *print* statements to track event flow during debugging. In our observations, one programmer used print statements to debug his python scripts. A second programmer used Java *println*s to examine the results of his database query. This programmer also uses Java assert statements, as they quickly signal failing conditions, and provide a link to the failing line of code (in the Eclipse IDE). In response, we designed a monitoring interface to help developers understand how input events result in debugging output and application feedback.

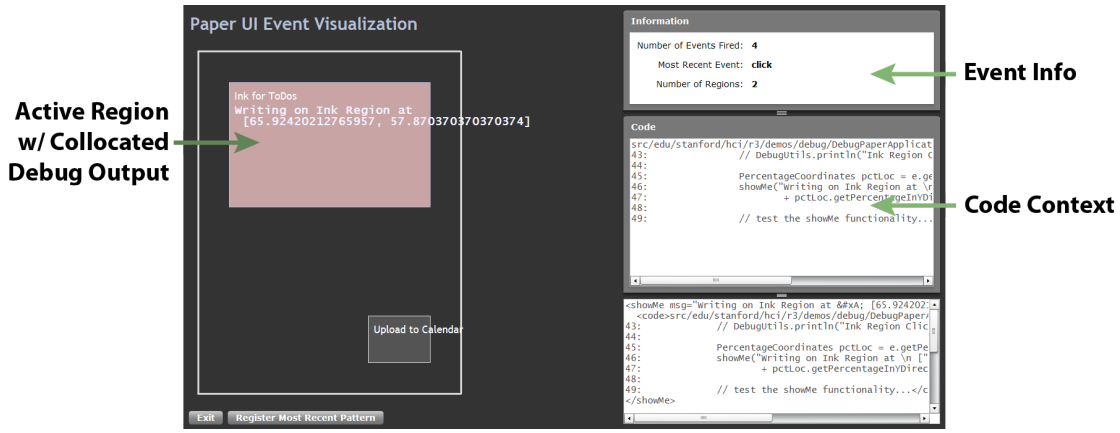


FIGURE 8.2. This design helps programmers visualize events as they fire. When a paper UI component receives input, its on-screen representation glows, eliminating the need for *got here* println*s*. If the developer uses `showMe(message)`, debug output is displayed in the parent region (*left*). The output is collocated with the component being tested, and not lost in the stream of console output. Calls to `showMe` also reveal nearby code (*right*), to help the developer recall the context of the debug output.

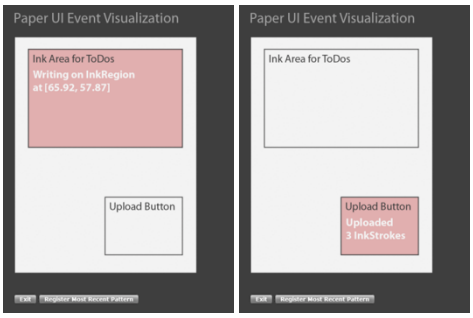
Figure 8.2 displays a working prototype of an event flow monitor. When the developer interacts with a paper UI component, its on-screen representation glows in red (eliminating the need for *print* statements). If the developer decides needs to debug a handler, she can use the `showMe(string)` method to display output on top of the parent region. This separates that particular statement from the other debugging text normally present in the IDE's console. Additionally, a panel displays the source code near the location of the `showMe(…)`. This allows developers to recall the intended behavior and context of the event handler.

Analysis of the Event Flow Visualization

The event visualization with the `showMe()` functionality enhances *visibility* of the toolkit's internal model of the paper application and its events. It was inspired by Papier-Mâché's monitoring window, which shows all physical input objects that are currently sensed by the system [Klemmer, *et al.* 2004]. One advantage of PaperToolkit's visualization is that it overlays debug output on top of location of interest (the paper UI component). One drawback is that using `showMe` requires a recompilation of the class, though this is the case

with using any *print* statement. Like *print* statements, calls to `showMe` can be shared via source control with teammates.

The design provides a *close mapping* to the developer's world, because the visualization looks like a paper UI. The debugging output has good *role-expressiveness*, because it is clearly tied to the underlying UI component. However, compared with a text-only display, common in the debugging console of IDEs, this visualization is much less *terse*.



The main drawback of this design is that when a developer interacts with a second region, the first highlight fades away. The developer cannot navigate the history of interactions, and may have trouble determining causality between input events and application output. This is addressed by a later design response, which places event visualizations side by side (see SECTION 8.6). The side-by-side version of event visualization would trade off *terseness* for increased *juxtaposability* and visibility of *hidden dependencies*. That is, a timeline takes up more space, but this added space allows developers to view interaction history, and understand causality, both of which are normally hidden. An hybrid solution would be a timeline that allows the developer to reanimate events in place.

8.4 Enhancing Testing with Visual Replay

Early developer feedback taught us that paper interfaces provided extra challenges to debugging. For example, a team may have limited pen hardware (*e.g.*, each team of four students shared a single digital pen). Also, debugging may become tedious, if a programmer physically reproduces pen input each time. In response, we introduced rudimentary save and replay into PaperToolkit (see SECTION 6.6). The toolkit automatically saves all input, and the developer can load log files to replay test sessions. This did not allow a developer to choose *which parts* of the log to replay, or at *what speed* to play it.

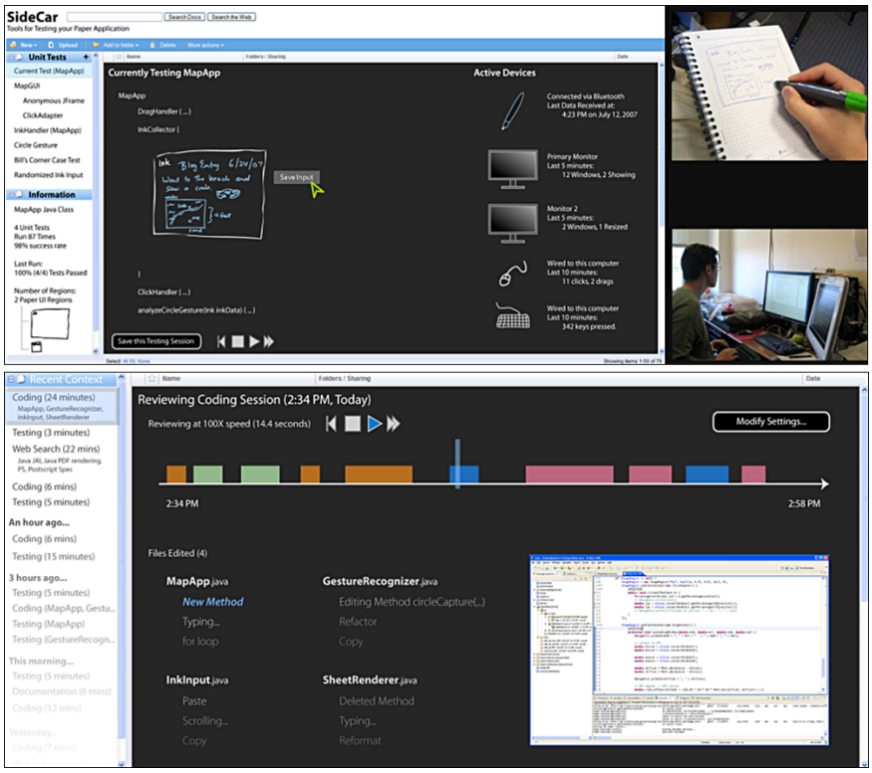


FIGURE 8.3. These frames from a video prototype show a developer defining a test case by providing pen input (*top*), and replaying the input by interacting with the timeline (*bottom*).

Figure 8.3 shows screens from a video prototype which explored the use of timelines to provide the developer fine-grained control over the replay of a test session. A developer can provide test input and save it as a test case (*top*). Then, she can replay the test session (or parts of it). Additionally, the developer can email the test session to a collaborator, who can replicate the input even without a physical pen.

This design provides the developer more control by giving up display space (*i.e.*, the visualization is less *terse*). It increases *visibility*, because the developer can now access parts of a replay log. There is low *premature commitment*, because the developer does not need to plan the input sequence before recording; she can choose portions to replay after the fact.

8.5 Rapid Prototyping by Simulating Printed Interfaces

Building paper applications taught us that the time required to physically print the interface became a bottleneck to testing. This design response speeds up the development workflow by allowing developers to delay printing until it is necessary, such as when deploying to an end user. To minimize the need for printing, PaperToolkit provides techniques to test event handlers using a graphical simulator or pre-printed notebooks.

The graphical simulator is accessed through the tray icon of the running application, and can be used with a mouse or Tablet PC stylus. Input is translated into *PenEvents* and sent to event handlers as if the developer were writing with the actual hardware pen on a paper printout.

To test with a digital pen, the designer uses an off-the-shelf Anoto notebook. Through the tray menu, she first selects a region to test. The system prompts her to define that region by drawing a rectangle (or upper-left bracket ⌜) onto the patterned page; this binding is saved for future test runs. After defining the regions, the

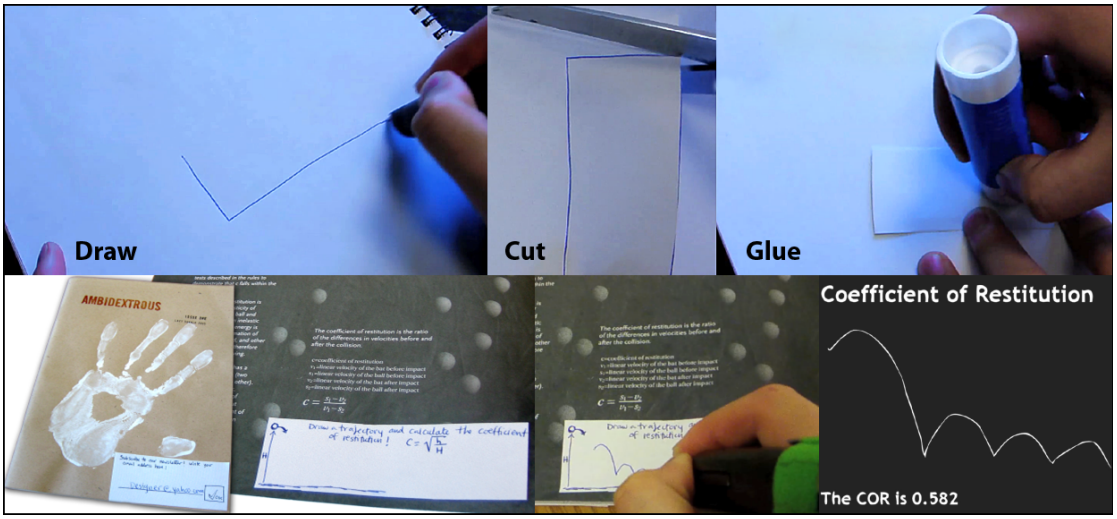


FIGURE 8.4. *Top*) The developer can test an event handler without printing the paper UI. She selects a handler and then draws a rectangle into a patterned notebook. PaperToolkit saves the binding for future test runs. The developer then cuts out and glues the region onto an existing artifact, such as a magazine. *Bottom*) We augmented a magazine to calculate the elasticity of a bouncing ball. The user draws a path, and the results are displayed on a monitor.

designer can cut them out and glue them to existing artifacts to create augmented paper prototypes.

As an example, we augmented a magazine with patterned paper and code specifying some behavior (see Figure 8.4). This technique enables rapid prototyping, as developers start from an existing artifact (*e.g.*, a magazine) and can test early prototypes without waiting for printers. It also allows developers to gather user data before implementing handlers, since pen input is logged. Combining simulation and replay, developers can code in response to real user data. They can provide users a wizard-of-oz prototype that records pen input. Then, they can code the behaviors while replaying the user input repeatedly.

8.6 Enhancing Visibility through Toolkit Monitoring

The technique in SECTION 8.3 and the prototype in SECTION 8.4 were combined into a monitoring environment called SideCar (see Figure 8.5). This working prototype is a vision of a future development and debugging environment based on the idea of capture and

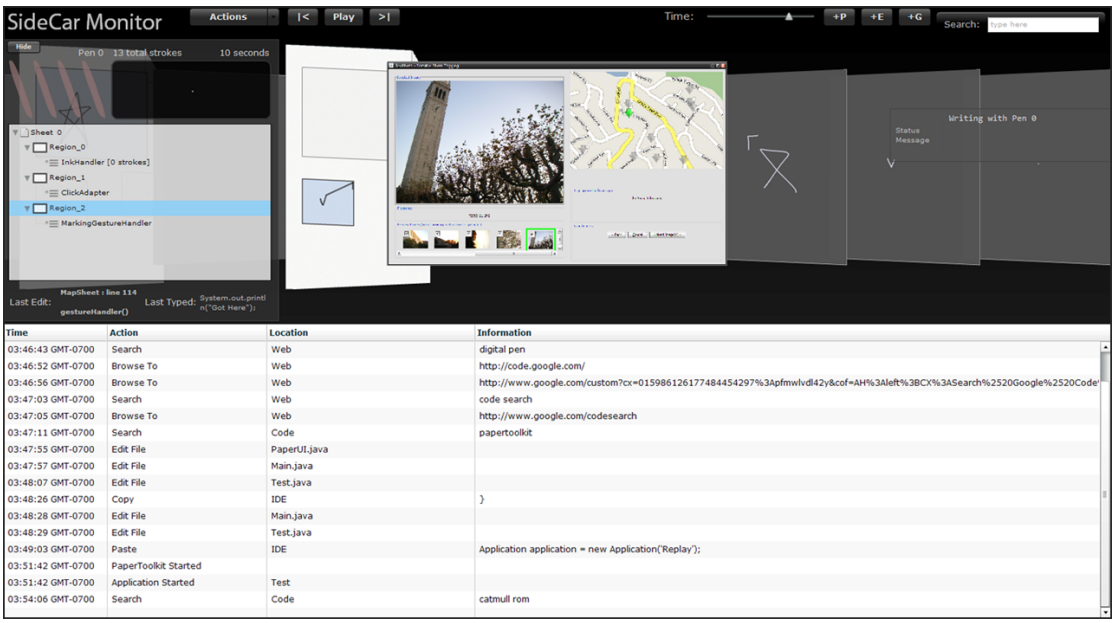


FIGURE 8.5. The SideCar interface sits on the programmer's secondary monitor. It is used to inspect a paper interface, and can help developers debug better through capture, access, and replay techniques. For example, the programmer can select a state and replay all pen input starting from that state (at normal or maximum speed).

access.

Previous systems have explored computer-assisted techniques to make debugging more efficient. Ko's WhyLine provided a menu-based interface to allow programmers to answer questions about a buggy program [Ko and Myers 2004]. Klemmer provided a monitoring interface for Papier-Mâché that helped developers understand the toolkit's internal computer vision algorithm [Klemmer 2004]. Maynes-Aminzade included visualizations to help programmers understand different vision algorithms [Maynes-Aminzade, *et al.* 2007]. Inspired by these techniques, SideCar offers an interface to help developers understand and monitor their paper applications (see Figure 8.5).

Capture and Visualization of Test Input

SideCar observes a developer's actions and the system's feedback during development and debugging, and presents a summary view on a secondary display, since today's programmers commonly use two monitors. To help a programmer track event flow, SideCar highlights the paper UI components as input arrives, as inspired by the design in Figure 8.2. The advantage is that SideCar adds new entries into the visual timeline, allowing the developer to navigate the history of the test session.

Monitoring Developer Actions

The history panel (see Figure 8.5, lower half) displays a view of actions the developer takes in the IDE and web browser. It also logs test input and program output, as reported by the toolkit. This helps reduce the impact of any interruptions a developer may experience. The developer can reacquire recent work context by glancing at the table of recent actions. She can also revisit earlier work by scrolling back in time. Future versions of this interface may allow the developer to access her code by selecting any row that specifies a source file (*e.g.*, clicking on "Edited ToDos.java" will open the file in the IDE).

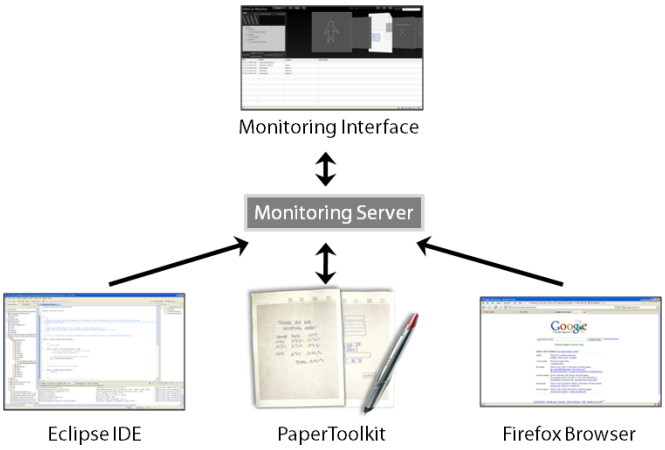


FIGURE 8.6. The SideCar monitoring interface visualizes actions taken in the IDE, browser, and toolkit, and can also help programmers track what the application is doing.

Architecture and Implementation

The SideCar monitoring view communicates with a server that receives information from three sources: the Eclipse IDE, PaperToolkit, and the Firefox web browser. The interface is built in Adobe Flex; the Eclipse IDE plug-in is built in Java; and the Firefox plug-in is built in JavaScript. The IDE extension tracks which files the programmer edits, and all copy/cut/paste interactions. The browser extension captures web interaction, including searches for code and changes to the system clipboard. The toolkit extension monitors system events, such as the firing of event handlers and debug output. These modules coordinate by passing messages over sockets to the main SideCar server, which is started automatically by the IDE. The architecture is summarized in Figure 8.6.

Analysis of SideCar

One of the possible benefits of this approach is that a developer may be able to recover from interruptions by examining the recent history of interactions. Programmers experience many interruptions throughout a workday. For example, in one hour of observing a researcher, we witnessed six interruptions—email, instant messaging, typing breaks, and a colleague. The participant even ignored several incoming emails because he was being observed. In the author's self-monitoring, interruptions and breaks consumed 16% of the

total work time. This is worse when programming paper + digital interfaces, as the developer incurs an interruption every time he prints his interface.

Studies on interruptions have presented similar results. For example, computer users can spend nine minutes responding to an email, and 15 more minutes before resuming their main task [Iqbal and Horvitz 2007]. Moreover, users receive up to eight of these interruptions every hour. Iqbal and Horvitz recommended that tools help suspend and resume work contexts. We took this as inspiration to explore capture and review of development history with SideCar.

The replay interface that PaperToolkit provides (a system tray menu item to load a log file) works well for most cases. Thus, at first glance, having finer-grained timeline control may seem to be overkill for repeated testing; with respect to the *abstraction gradient* cognitive dimension, a timeline view of replay logs may be too low-level a representation.

However, the SideCar timeline view adds value to replay. In this case, the developer can visually inspect which parts of the log she would like to replay. We could also provide an interface for developers to select parts of the log to email to collaborators. Thus, the timeline would provide flexibility and support for sharing.

The Omniscient Debugger [Lewis 2007] is a debugging tool that introduces the ability to rewind and inspect variable assignments in Java. SideCar's timeline operates at a higher abstraction level, and concerns itself with ink input, paper UI events, and application output. For example, the Omniscient Debugger does not present a graphical visualization of application input or output. However, both of these techniques are heavyweight in that they require an extra tool to monitor or control the main program. A long-term deployment could help determine if an environment such as SideCar would be too heavyweight for everyday use (even if it were launched automatically by the IDE).

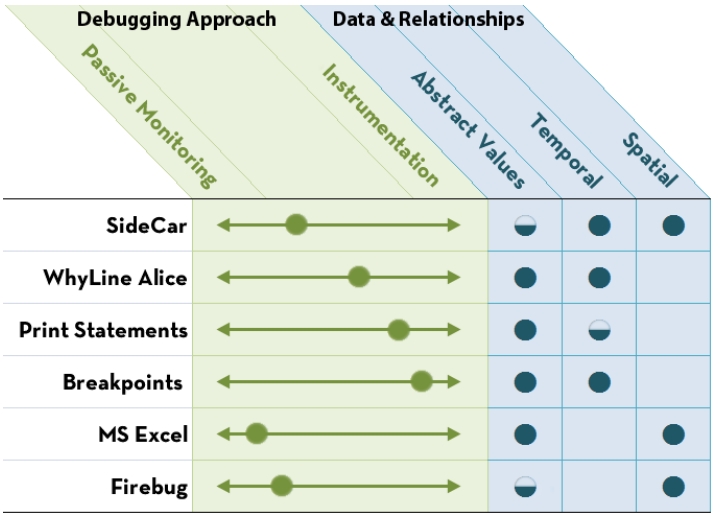


TABLE 8.1. This table compares design concerns for debugging with different tools. Tools like Excel and the Firebug plug-in for Firefox can reveal debugging information without any instrumentation. SideCar and WhyLine require the user to either start the environment, or ask a question. SideCar is good at visualizing temporal and spatial relationships in the program, while breakpoints are best at illustrating temporal cause and effect.

Design Concerns for Debugging Tools

To further analyze SideCar, we selected two design concerns for debugging tools (see Table 8.1). The first concern is the *debugging approach*, what the developer needs to do to elicit debugging information. With SideCar, a developer can take a passive monitoring approach. After the environment is launched, it simply logs all actions and outputs. The developer then needs to browse through the data if he wishes to find anything in particular; future versions can include text search.

Other environments require developers to take an initial action, or actively instrument the application. For example, the timeline in Ko's WhyLine requires users to first ask a question about program behavior. Similarly, *print* statements require a developer to make a hypothesis about the location of the bug, and then insert and later remove the statements. Monitoring and instrumentation both have benefits—monitoring may help reveal bugs, and instrumentation can help to locate them.

A second concern is the *data and relationships* that can be illustrated using the technique. Print statements can illustrate the values of variables, such as the number of unique

objects in a hash table. Breakpoints are good at illustrating temporal relationships, because the developer can step through lines of code to see cause and effect. Modern spreadsheets (*e.g.*, Microsoft Excel, Apple Numbers) provide a live debugging environment that displays the relationships between data spatially: referenced cells are highlighted on the spreadsheet. SideCar illustrates both temporal and spatial relationships well, by showing causality and 2D representations of paper UIs.

8.7 Improving the Learnability and Usability of the API

PaperToolkit provides a large number of classes and methods. Browsing through the API documentation requires a lot of time, even for skilled programmers. To address this, we prototyped a visual environment that allows developers to experiment with API calls (see Figure 8.7). In one sense, the environment acts as live documentation for the API.

The technique helps developers preview the effect of API calls, to make it easier to find the right method. The developer provides sample ink input, and then selects a method from a drop-down menu. In Figure 8.7, the developer has selected a method to find the

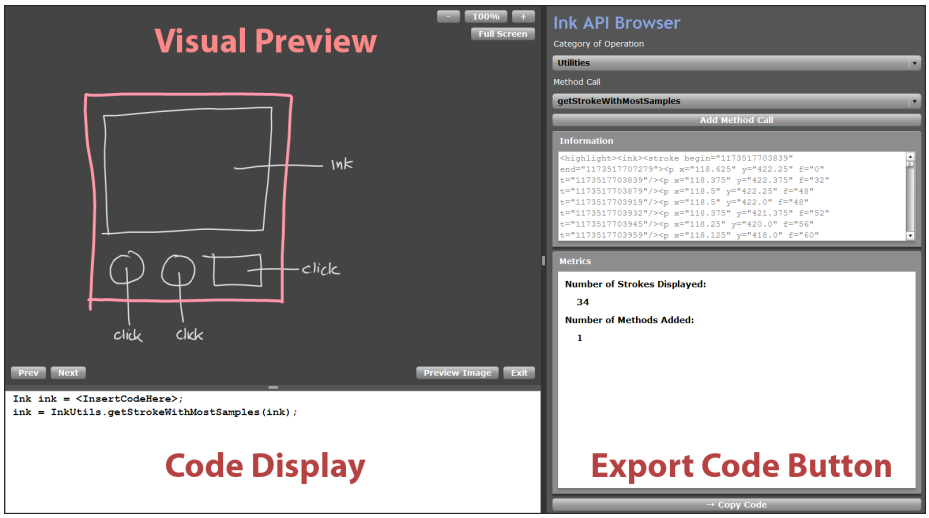


FIGURE 8.7. This working prototype helps developers explore PaperToolkit's Ink API by providing real-time previews of method calls. The developer provides input, displayed in the preview pane. She then selects a method to call from a dropdown (in this case, she has selected a method to find the longest stroke). The visual preview updates immediately and highlights the results of the query. When the developer has selected the right combination of methods, she can export the code to her IDE by clicking the COPY CODE button.

longest ink stroke. The result—the outermost ink stroke—is highlighted in red.

This approach shortens the *gulf of evaluation* in picking the right method, because normally the programmer cannot see the effect of a method until she compiles and runs her program [Hutchins, *et al.* 1985, Lieberman and Fry 1995]. This approach speeds up coding for novice users of the toolkit. However, it only works for API calls that have an obvious visual output, such as the querying or clustering of ink strokes.

API Modifications

Even with visual tools, a developer will at some point need to write code to specify application logic. She will need to interact with PaperToolkit's Java API, the programmer's *user interface*. Observations from deployments to the class and researchers helped us improve the API iteratively. Many of these modifications address concerns articulated by the cognitive dimensions framework (see SECTION 2.4), and those dimensions adapted by Steven Clarke to assess API usability [Clarke 2004]. This section lists examples of how the API was modified.

Terseness refers to how much code a developer needs to write to achieve a particular goal. One outcome of the source-code analysis was that the most-frequently-used classes dealt with paper interface construction. Many lines of code were devoted to *adding* paper UI components to a parent component. Also, to specify the size of components, developers needed to construct *Units* objects (*e.g.*, `new Inches(5)`). To improve code readability, we eliminated these extra steps.

For example, the toolkit now provides `app.createSheet(width, height)` and `sheet.createRegion(x, y, width, height)` methods. These factory methods automatically add child components to the parent components, saving one line of code for each paper UI component. The default methods now also assume that values are in *Inches* unless the developer specifies another *Units* class.

However, this factory pattern may improve readability at the cost of learnability. Introduced by the "Gang of Four" [Gamma, *et al.* 1995], the factory pattern was recently

shown to be less usable than standard constructors in a laboratory study [Ellis, *et al.* 2007]. One problem the researchers identified was that factory patterns usually require developers to navigate a deep tree of abstract classes. PaperToolkit avoids this problem by preserving the simple constructors. The factory methods are not required, but act as convenience methods.

Consistency refers to "how much of the rest of the API can be inferred once part of it is learned" [Clarke 2004]. We applied this metric when improving the UI construction methods. Since `createSheet()` was added to the *Application* class, we also added a `createRegion()` method to the *Sheet* class. Abstractions and class names were also modified to better meet this goal. For example, the first version of the toolkit included *filters* and *handlers*. Both were behaviors that could be added to a region, but filters changed the meaning of the ink strokes (*e.g.*, handwriting recognition) while handlers invoked commands (*e.g.*, click). Developers found this distinction confusing, so later versions of the toolkit provided a consistent naming for all behavior classes. *InkCollector* was renamed to *InkHandler*, and *HandwritingContentFilter* was renamed to *HandwritingHandler*.

8.8 Summary

This chapter presented techniques to improve the usability of PaperToolkit. Sketching lowers the threshold so that non-programmers can create working interfaces. Event flow visualization increases visibility to help developers understand the toolkit's internal model. We also introduced a technique to allow developers to quickly prototype interactive paper applications, by using off-the-shelf notebooks. Toolkit monitoring is a new approach at helping developers review and replay their actions, and recover from distractions (*i.e.*, remember what they were doing earlier in a test session). Finally, we improved the toolkit at the API level by applying the cognitive dimensions principles. This work distinguishes itself from the prior paper + digital toolkits by introducing improved designs as a direct response to results from long-term studies with actual users. Figure 8.8 summarizes this chapter's designs, and points toward opportunities for future work.

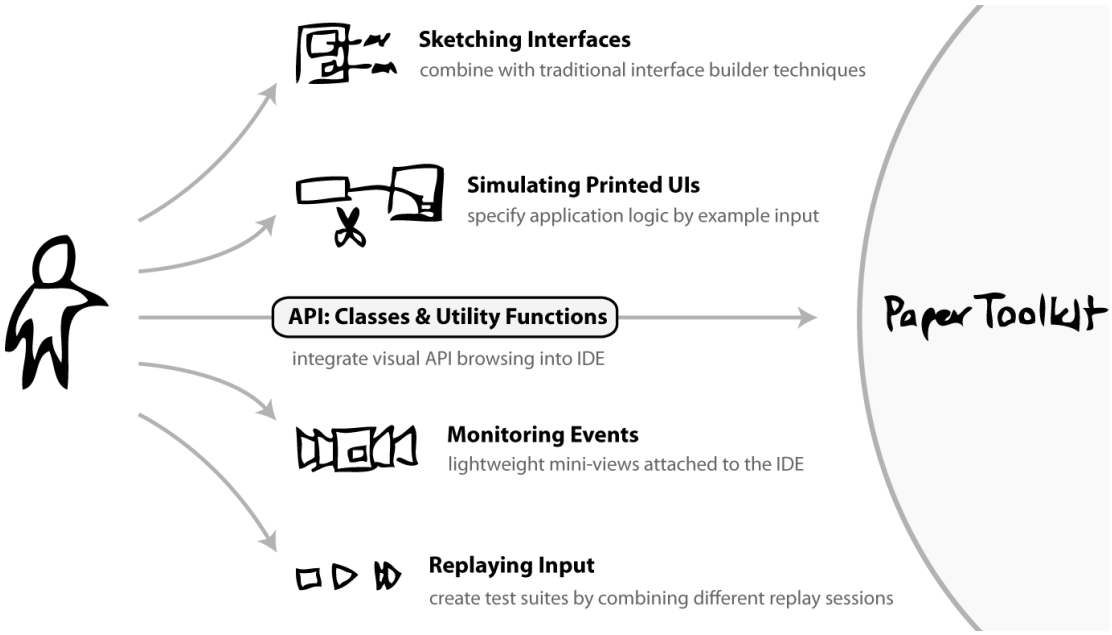


FIGURE 8.8. This summary of the design response suggests a few opportunities for future work. For example, it would be worthwhile to investigate visual API browsing tools for the IDE.

9

Conclusion

This dissertation addressed the problem of designing, developing, and debugging paper + digital applications. The work was motivated by an in-depth observational study of field biologists. We then demonstrated the value of augmented paper interactions through two systems: ButterflyNet and GIGAprints. These systems informed the design of PaperToolkit, a software architecture and suite of tools for this new genre of applications. An evaluation through long-term deployment and source-code analysis showed that the toolkit provides a practical and effective approach to creating paper applications. The evaluation also revealed challenges, which we addressed through a design response—providing better tools to develop and debug applications. To conclude this work, we summarize the contributions and present ideas for future research.

9.1 Summary of Contributions

The study of field biologists helped us determine potential needs for paper + digital interfaces. These needs, in turn, inspired a range of interactions that we built and tested, including a smart notebook (ButterflyNet) and large interactive prints (GIGAprints). The experience informed the design of PaperToolkit, this dissertation's core contribution. This section lists the key aspects of this toolkit research.

Architectural Features

A key design insight was that adopting the graphical interface event model would allow programmers to quickly pick up and use the toolkit. We could then explore more novel aspects of paper + digital applications, such as improved techniques for development.

Though PaperToolkit provides a familiar environment for GUI programmers, it differs in several respects. First, not all input is processed in real-time, so programmers must be able to work with events that are invoked in batch. Second, paper does not provide updates, so programmers must design interactions where output is sent to a nearby desktop or handheld PC. PaperToolkit addresses the first difference by making batched event handling identical to real-time event handling, and addresses the second by introducing a *Device* API to simplify output to remote computers.

Finally, PaperToolkit's software abstraction for the digital pen supports save and replay. When the pen is receiving input, all coordinates are logged automatically. Log files can be replayed at regular or full speed. The event dispatcher acts identically in both cases, dispatching incoming coordinates as events to the handlers.

Evaluation through Deployment and Source-Code Analysis

This is the first pen-and-paper toolkit that has been evaluated through deployments, analysis of the code produced by developers, and inspection through different analytical lenses (*e.g.*, cognitive dimensions). The methods provide distinct benefits. The long-term deployment proved that the toolkit was useful—students created interesting working applications in six weeks, while the author and colleagues required several months to create ButterflyNet using the basic SDK. Source-code analysis suggested improvements at a lower level. We renamed classes and methods, and added convenience methods to make the code easier to write and read. The analysis also informed our design response. Analysis with the cognitive dimensions enabled inspections of both the API and the higher-level workflow.

Tools and Techniques for Designing and Debugging

Some of the most interesting outcomes of this work are the techniques to enhance the process for creating and testing paper interfaces. Paper applications introduce a host of programming and debugging challenges. Printing interfaces is slow, and this can discourage iterative design. Replicating test sessions can become tedious. PaperToolkit solves these problems by enabling developers to simulate printed interfaces with pre-printed Anoto paper, and by replaying test sessions. Simulation of paper interfaces enables rapid prototyping, since developers can implement behaviors without needing to create a fully-working paper interface. The replay technique is especially powerful, as developers can share test sessions to support remote collaboration.

PaperToolkit also suggests a different approach to development and debugging. By passively monitoring an application's input and output, the toolkit can allow the developer to better understand cause and effect in an application. This allows programmers to review test sessions, or use the visual timeline as an index into the source code (*e.g.*, double-click an event handler to jump to the corresponding code).

Design Guidelines for Pen-and-Paper Interfaces

This work also contributes design guidelines for paper interfaces (SECTION 7.6). For example, printed interfaces need clear textual and/or graphical instructions for operation because there is not yet a standardized widget set for paper interfaces (we can borrow some, but not all, of the GUI conventions). Since there is no guarantee that users will follow these printed directions (paper cannot enforce interface constraints) application designers must support out-of-order input and account for incomplete data.

PaperToolkit started out by supporting only real-time input (previous work dealt mostly with batched pen input). However, we quickly learned that many developers also desired support for batched input, as this eases mobile interactions. Thus, paper + digital interfaces should allow seamless switching between batched and real-time modes. In the

future, this distinction will fade away, as digital pens gain more computing power, and are able to communicate in real-time with paper notebooks augmented with e-ink.

9.2 Opportunities for Further Research

This dissertation suggests a range of future work, including research opportunities which are made available by tools like PaperToolkit. This section describes several categories, in the areas of recognition, collaboration, and input/output coordination.

Enhancing Recognition and Transformation of Ink Input

Existing handwriting recognition packages (*e.g.*, Microsoft Tablet PC) perform more poorly when applied to batched input than when run as an on-line recognizer. In our experience, paper notebooks seem to *encourage sloppiness*. Perhaps the medium is more comfortable to scribble on. Perhaps, when real-time recognition results are available, the user actually modifies his handwriting to be more recognizable by the computer.

When the user writes on a paper notepad in the field, he is not thinking about recognition, and does not receive feedback on the quality of his writing. This reveals an opportunity to explore different approaches and algorithms to process handwriting and sketches. New techniques could further enhance data transcription through recognition. For example, the multimedia spreadsheet (see SECTION 4.3) could use recognition results to provide suggestions to the user, highlight potential user errors, and reduce typing.

Searching, Visualizing, and Sharing Paper Notes

Related to the problem of recognition is the task of searching through paper notes. One technique might combine handwriting recognition with visualizations to help designers revisit their old notes. For example, a browsing interface may display key words recognized from a paper notebook. Clicking on any term will reveal those pages that match.

Of course, paper notebooks are not the only interface that people like biologists and designers use. There are opportunities to explore how notes can integrate with the larger ecology of devices and applications in the end user's toolbox. Will the user be able to rapidly

print items from their mobile phone? Will they be able to schedule updates to documents (*e.g.*, automatically print a new version when content changes)?

Coordinating Input and Output

Paper in one form or other has been around since the ancient Egyptians, and there are no signs of it disappearing. Soon, e-ink technology will allow paper to present real-time graphical updates (much like the newspapers in Harry Potter's world). Much of this technology is becoming available commercially (see Amazon's Kindle e-book reader). But until e-ink is ubiquitous, we need to look more into the scenarios where users combine input on paper with output on devices. This is an opportunity to understand how to interact with content on and off the paper surface. For example, a hybrid notebook could combine a paper notebook with a PDA. When the user needs the benefits of paper, he can write on the paper half; when he needs to browse media, he can interact with the digital half.

Yet another technology, erasable paper, seeks to reduce the environmental impact of our paper-centric work practices [Xerox 2007]. Xerox's Dalal has observed that much of the paper that comes out of printers end up in the recycling bin *on the same day* they are printed. Their lab has developed special paper that erases itself in a day, and can be reused until the paper is damaged. With this model, paper is just a temporary display surface, one that refreshes not at 60Hz, but on demand (whenever a person clicks *print*).

Supporting Studies

PaperToolkit provides a low-threshold means to capture handwritten input, both in real-time and in batched mode. This opens up opportunities for using the toolkit to support studies in writing activities. For example, Ehlen and others in the Computational Semantics Laboratory at Stanford have already used PaperToolkit to monitor real-time note taking in collaborative planning sessions. In the future, the toolkit could help experimenters by providing better interfaces for monitoring real-time input, or for providing wizard-of-oz output. PaperToolkit could also include tools to help experimenters gather and visualize

statistics on ink strokes, or apply queries to a database of digital notes (*e.g.*, find all notes where the user paused for three seconds before pressing the OK button in the GUI).

9.3 Closing Remarks

In the introduction, we mentioned existing studies which suggest that a wholesale replacement of paper in work practices would be problematic [Heath and Luff 2000, Sellen and Harper 2001]. In fact, the physical affordances of paper encourage its continued use, even in today's technology-centric world. The ability to flexibly arrange printed artifacts enables users to author for themselves information-rich environments that lower the cost of finding common information [Kirsh 1995].

In this research, we have seen that paper and digital devices can be used in tandem. In the right scenarios, paper + digital applications can provide added robustness, flexibility, mobility, and support for co-located collaboration that may be missing from today's devices. Building these applications is not much more difficult than building graphical applications.

Looking to the future of paper-digital integration, we can return to our original inspiration. In 1993, Wellner wrote that we "interact with documents in two separate worlds....Unfortunately, choosing to interact with a document in one world means forgoing the advantages of the other" [Wellner 1993]. We hope we have convinced the reader that choosing between paper and digital is no longer necessary—we can have both.

A

User Study Materials

A.1 Questionnaires

This section contains questionnaires we used during the laboratory studies for ButterflyNet and GIGAprints. The forms were given to users at the end of their sessions, to help us assess the usability of our systems. In addition to questionnaires, we also held informal debriefing interviews to gather feedback that was not easily expressed on the forms.

Responses were entered into a spreadsheet by hand, where we used Microsoft Excel and SPSS to calculate statistics. Some of the freeform questions were coded so that we could distill recurring themes.

At one point, we explored an idea for automating this process through a paper + digital application. The form would be augmented with PaperToolkit, and incoming pen strokes would be automatically sorted by question. Likert-scale data would be automatically tallied. Such a tool would substantially speed up the workflow of HCI researchers.

Participant Code Name _____

ButterflyNet Questionnaire Part I
General Background

1 Which of the following describes your occupation? (mark all that apply)
- ○ Field researcher
- ○ Lab researcher
- ○ Student
- ○ Professor
- ○ Professional
- ○ Other (please write in)_____

2 What field/major/discipline are you in? (mark all that apply)
- ○ Biology
- ○ Medicine
- ○ Ecology
- ○ Earth Systems
- ○ Geology
- ○ Other (please write in)_____

3 What institution is your occupation associated with? (mark all that apply)
- ○ University
- ○ Museum
- ○ Industrial lab
- ○ Non-profit organization
- ○ Other (please write in)_____

4 What is the highest level of education that you have completed?
- ○ Some High School
- ○ High School Diploma
- ○ Some College
- ○ College Degree
- ○ Masters
- ○ PhD
- ○ Other Professional Degree (please write in)_____

5 How much experience do you have with respect to field data collection methods and techniques?
- ○ None
- ○ Less than 1 year
- ○ 1-2 years
- ○ 2-5 years
- ○ 5-10 years
- ○ over 10 (please write in)_____

Field research

6 For your field work, what data capture devices or tools do you use, and how often do you use them? (please circle one answer for each row; fill in others if applicable)

	Never	Very Infrequently					Very Frequently
Field notebook	0	1	2	3	4	5	6
PDA	0	1	2	3	4	5	6
Audio Recorder	0	1	2	3	4	5	6
Laptop Computer	0	1	2	3	4	5	6
Video Recorder	0	1	2	3	4	5	6
Film Camera	0	1	2	3	4	5	6
Digital Camera	0	1	2	3	4	5	6
GPS Device	0	1	2	3	4	5	6
Other (please write in) _____	0	1	2	3	4	5	6
Other (please write in) _____	0	1	2	3	4	5	6

7 Do you use any computer software while in the field? If so, please name them and describe how/why they are important to your work.

8 How many field notebooks do you generate **per year**? (on average)

- ○ None
- ○ One
- ○ Two to Three
- ○ Three to Five
- ○ Five to Ten
- ○ More than Ten (please write in)_____

Participant Code Name _____

9 When taking field notes in your experiments, how important is it to take photographs that accompany those notes? Please circle a number.

Very Unimportant						**Very Important**
1	2	3	4	5	6	7

10 On average, how many pictures do you take during a day of field work?

- ○ None
- ○ 1-5
- ○ 5-10
- ○ 10-20
- ○ over 20

11 When taking field notes in your experiments, how important is it to capture additional sensor data? (e.g., temperature, humidity, GPS.)

Very Unimportant						**Very Important**
1	2	3	4	5	6	7

12 What are the **significant limiting factors** to the amount of data you capture in the field? (mark all that apply)

- ○ Hardware limitations (e.g., film roll/memory card capacity, lack of thermocouples)
- ○ Battery power and longevity
- ○ Poor software support to organize and retrieve the data
- ○ The field is a harsh environment for electronic equipment.
- ○ Too much data is not useful. I'd rather capture fewer, but more specific, data points.
- ○ Lack of time
- ○ Other (please write in)_____
- ○ Other (please write in)_____

13 When capturing field data, how important is it to have the exact date/time each note, picture, or measurement was collected?

Very Unimportant						**Very Important**
1	2	3	4	5	6	7

14 When working in the field, how important is it to organize the captured data **on-the-spot** (in your notebook or portable computer) instead of doing it later, back at the lab or field station?

Very Unimportant						**Very Important**
1	2	3	4	5	6	7

15 Would each of the following field tools/techniques **increase or decrease** your current field time? By how much?

	Decrease			**Same**			**Increase**
Digital Pen/Notebook	-3	-2	-1	0	1	2	3
Hotspot linking with SmartCamera	-3	-2	-1	0	1	2	3
Visual specimen tags	-3	-2	-1	0	1	2	3

Participant Code Name _____

Questionnaire Part II

16 Please rate the following ButterflyNet features in terms of their usefulness?

	Very Useless						**Very Useful**
Digital Pen/Notebook	1	2	3	4	5	6	7
Hotspot linking with SmartCamera	1	2	3	4	5	6	7
Visual specimen tags	1	2	3	4	5	6	7

17 How likely is it that you would integrate each of ButterflyNet's field tools/techniques as a regular part of your work flow?

	Very Unlikely						**Very Likely**
Digital Pen/Notebook	1	2	3	4	5	6	7
Hotspot linking with SmartCamera	1	2	3	4	5	6	7
Visual specimen tags	1	2	3	4	5	6	7

18 When **organizing, transcribing, or analyzing** scientific research, what software do you use, and how often?

	Never	Very Infrequently					Very Frequently
Personal Organizer (e.g., OneNote, Outlook)	0	1	2	3	4	5	6
Word Processor	0	1	2	3	4	5	6
Spreadsheet (e.g., Excel)	0	1	2	3	4	5	6
Database (e.g., Access)	0	1	2	3	4	5	6
Statistics Package	0	1	2	3	4	5	6
Web Browser	0	1	2	3	4	5	6
Photo Browser	0	1	2	3	4	5	6
Math Tools (e.g., Matlab)	0	1	2	3	4	5	6
Bibliography (e.g., Endnote)	0	1	2	3	4	5	6
Mapping Tools (e.g., GIS)	0	1	2	3	4	5	6
Other (please write in) _____	0	1	2	3	4	5	6
Other (please write in) _____	0	1	2	3	4	5	6

19 When **publishing or presenting research results**, what software do you use, and how often?

	Never	Very Infrequently					Very Frequently
Presentation Tools (e.g., PowerPoint)	0	1	2	3	4	5	6
Email	0	1	2	3	4	5	6
Word processor	0	1	2	3	4	5	6
Adobe PDF	0	1	2	3	4	5	6
Web Browser	0	1	2	3	4	5	6
Other (please write in) _____	0	1	2	3	4	5	6

Participant Code Name _____

20 On average, how much time do **you** spend after each field day on transcribing field data from your notes to your computer software?

- ○ None; field data is only kept in my notebook
- ○ None; other people enter my data
- ○ less than 1 hour
- ○ 1-2 hours
- ○ 2-4 hours
- ○ more than 4 hours
- ○ I don't transcribe field data every day, but do it every _____.

My daily average is_____ hours.

21 Which of the following methods do you use to find data related to your notes? (e.g., photos, temperature records, video)? (mark all that apply)

- ○ browse through handwritten annotations
- ○ use the computer to search for documents/files
- ○ by memory
- ○ Other (please write in)_____

22 How likely is it that the ButterflyNet browser **helps a colleague understand** your field notes better? (suppose that legibility is NOT an issue.)

Very Unlikely						**Very Likely**
1	2	3	4	5	6	7

23 How likely is it that the ButterflyNet browser helps you to **quickly recall** experiments that happened months or years ago?

Very Unlikely						**Very Likely**
1	2	3	4	5	6	7

24 With ButterflyNet, how likely is it that you will spend **less time transcribing data**?

Very Unlikely						**Very Likely**
1	2	3	4	5	6	7

25 Would you agree or disagree with the following statement?
The ButterflyNet tools are effective in helping me capture/manage large amounts of data.

Strongly Disagree						**Strongly Agree**
1	2	3	4	5	6	7

26 Compared to conventional tools, will ButterflyNet require **more or less time** in the following phases of your research?

	Much Less Time						**Much More Time**
Hypothesis proposal	1	2	3	4	5	6	7
Protocol creation	1	2	3	4	5	6	7
Field data capture	1	2	3	4	5	6	7
Data organization	1	2	3	4	5	6	7
Data transcription	1	2	3	4	5	6	7
Data Analysis	1	2	3	4	5	6	7
Paper Publication	1	2	3	4	5	6	7

27 How likely is it that you would integrate ButterflyNet's various lab tools/techniques as a regular part of your work flow?

	Very Unlikely						**Very Likely**
ButterflyNet Browser	1	2	3	4	5	6	7
Navigating by Pen	1	2	3	4	5	6	7
Sending Data to the Spreadsheet	1	2	3	4	5	6	7
Multimedia Spreadsheet	1	2	3	4	5	6	7

Participant Code Name _____

Overall Evaluation

28 What aspects of ButterflyNet do you particularly **like** or find most useful?

29 What aspects of ButterflyNet do you particularly **dislike** or want changed?

30 What additional features would you like to see in ButterflyNet? What would it take for ButterflyNet to be truly useful to you?

Thank you for your participation! ☺

Participant Code Name _____

GIGAprints Questionnaire

Background

1 Which of the following describes your occupation? (mark all that apply)
- O Student
- O Researcher
- O Professor
- O Professional
- O Other (please write in)_____

2 What field/major/discipline are you in? (mark all that apply)
- O Biology
- O Medicine
- O Geology
- O Computer Science
- O Anthropology
- O Other (please write in)_____

3 What institution is your occupation associated with? (mark all that apply)
- O University
- O Museum
- O Industrial lab
- O Non-profit organization
- O Other (please write in)_____

4 What is the highest level of education that you have completed?
- O Some High School
- O High School Diploma
- O Some College
- O College Degree
- O Masters
- O PhD
- O Other Professional Degree (please write in)_____

5 How many hours **per week** do you use a computer? (circle the appropriate range)

Few						**All the time**
< 2 hours	2 - 4	5-10	11-20	21-30	31-50	> 50

6 How familiar are you with computer software and hardware?

Very Unfamiliar						**Like the back of my hand**
1	2	3	4	5	6	7

Twistr, BuddySketch, and AudioGuide GIGAprints
Let's evaluate the GIGAprints that you used today.

7 Please indicate how strongly you **agree** or **disagree** with the following statements:

I found *Twistr—The Photo Search Game*…

	Disagree			**Neutral**			**Agree**
…easy to use.	-3	-2	-1	0	1	2	3
…fun to play.	-3	-2	-1	0	1	2	3

A large printed sheet of photographs allows me to *find photographs faster* than a webpage with the same set of photos does.

	Disagree/It's Slower			**Neutral**			**Agree/It's Faster**
Find Photos Faster	-3	-2	-1	0	1	2	3

A large printed sheet of photographs allows me to *see more photographs at once* than a webpage with the same set of photos does.

	Disagree/See Fewer			**Neutral**			**Agree/See More**
See More Photos	-3	-2	-1	0	1	2	3

8 Can you use the techniques demonstrated in *Twistr* in your own work? If so, how?

9 What additional features would you like to see in *Twistr—The Photo Search Game*?

Participant Code Name _____

10 Please indicate how strongly you **agree** or **disagree** with the following statements:

I found *BuddySketch—The Remote Sketching Tool*…

	Disagree			**Neutral**			**Agree**
…easy to use.	-3	-2	-1	0	1	2	3
…useful.	-3	-2	-1	0	1	2	3

The sketching tool allows me to collaborate better with a remote colleague.

	Disagree/Worse			**Neutral**			**Agree/Better**
Collaborate Better	-3	-2	-1	0	1	2	3

11 Can you use the techniques demonstrated in *BuddySketch* in your own work? If so, how?

12 What additional features would you like to see in *BuddySketch*?

13 Please indicate how strongly you **agree** or **disagree** with the following statements:

I found *AudioGuide—An Augmented Tourist Map*…

	Disagree			**Neutral**			**Agree**
…easy to use.	-3	-2	-1	0	1	2	3
…useful.	-3	-2	-1	0	1	2	3

The audio interface allows me to be *more mobile* than when using a laptop computer.

	Disagree/Less Mobile			**Neutral**			**Agree/More Mobile**
More Mobile	-3	-2	-1	0	1	2	3

The audio interface will be *useful* when I am outdoors.

	Disagree/ NOT Useful Outdoors			**Neutral**			**Agree/ Useful Outdoors**
Useful Outdoors	-3	-2	-1	0	1	2	3

14 Can you use the techniques demonstrated in *AudioGuide* in your own work? If so, how?

15 What additional features would you like to see in *AudioGuide*?

Participant Code Name _____

The GIGAprints Concept in General
Let's evaluate the general concept of GIGAprints (paper interfaces, pens, and devices).

16 Suppose you were to integrate a GIGAprint into your workflow (not necessarily the ones you used today). Would the following GIGAprint abilities/techniques **improve/speed up** or **worsen/slow down** your work? By how much?

	Worsen/ Slow Down			**Same**			**Improve/ Speed Up**
Visually Locate Photos	-3	-2	-1	0	1	2	3
Retrieve Photos to another Device	-3	-2	-1	0	1	2	3
Work Away from the Desk	-3	-2	-1	0	1	2	3
Leverage Multiple Pens	-3	-2	-1	0	1	2	3
Work with a Colleague	-3	-2	-1	0	1	2	3
Work by Myself	-3	-2	-1	0	1	2	3

17 Please rate the following GIGAprint features in terms of their **usefulness**.

	Very Useless						**Very Useful**
Digital Pen and Paper	1	2	3	4	5	6	7
Large Sheets of Paper	1	2	3	4	5	6	7
Multiple Pens	1	2	3	4	5	6	7
See Lots of Content on a Single Page	1	2	3	4	5	6	7
Can Move or Reconfigure Prints	1	2	3	4	5	6	7
Can Collaborate Around a Print	1	2	3	4	5	6	7

Overall Evaluation

18 What aspects of GIGAprints do you particularly **like** or find most useful?

19 What aspects of GIGAprints do you particularly **dislike** or want changed?

20 What additional features would you like to see in GIGAprints? What would it take for GIGAprints to be truly useful to you?

Thank you for your participation! ☺

6

A.2 Consent Forms

We obtained consent from participants before each study. Following is the form we used for the ButterflyNet study. The other lab studies used similar forms.

ButterflyNet Study Consent Form

FOR QUESTIONS ABOUT THE STUDY, CONTACT:
Ron B. Yeh, ronyeh@stanford.edu, 650-823-7898.
353 Serra Mall, Gates Building Rm. 386, Stanford University, Stanford CA 94305.

DESCRIPTION: You are invited to participate in a research study conducted by Ron B. Yeh on the practices of field biology researchers. Our goal is to test a new computer system designed to help field researchers perform their tasks in the field and in the lab. You will be asked to use the technologies to perform a mock field experiment. After returning to the lab, you will use the software to browse a set of photos and notes, and perform several tasks using them.

Throughout the experiment, you will be audio-taped, photographed, and video-taped for the purposes of keeping a detailed account of the steps you take. These tapes will be used internally by the research group to support the revision and future design of our computer systems. After the project is completed, these tapes will be destroyed.

Observations and/or interpretations made during this study may be presented in research papers/conferences. However, your name or other information that identifies you will not be published or disclosed.

RISKS AND BENEFITS: The risks associated with this study are minimal. The benefits which may reasonably be expected to result from this study are that you will interact with next-generation computer tools for biologists.

TIME INVOLVEMENT: Your participation in this study will take approximately 2.5 hours.

PAYMENTS: You will receive $45 cash for your participation.

YOUR RIGHTS: If you have read this form and have decided to participate in this project, please understand your participation is voluntary and you have the right to withdraw your consent or discontinue participation at any time without penalty. You have the right to refuse to answer particular questions. Your individual privacy will be maintained in all published and written data resulting from the study.

If you have questions about your rights as a study participant, or are dissatisfied at any time with any aspect of this study, you may contact—anonymously, if you wish—the Administra-

tive Panels Office, Stanford University, Stanford, CA (USA) 94305-5401 (or by phone (650) 723-2480—you may call collect).

I give consent to be photographed/audiotaped/videotaped during this study. These materials will be used by the researcher to recall steps taken while using the technologies. Although full and exact transcriptions of audio and video will not be published, short excerpts may be published to support the work. Non-identifying photos may also be used in research publications:

Please initial: ____ Yes ____ No

I give consent for recorded materials resulting from this study to be used for discussion in the primary research group working on the BioACT! project:

Please initial: ____ Yes ____ No

The extra copy of this consent form is for you to keep.

PRINTED NAME _____

BIOLOGIST IDENTIFIER: [BNET.]

SIGNATURE _____ DATE _____

EMAIL ADDRESS _____

Approval Date: ___11/19/2004____

Expiration Date: ____11/18/2005____

A.3 Tasks for ButterflyNet

The following sheet of directions was provided to all ButterflyNet participants. GIGAprints users were given a similar sheet of directions.

ButterflyNet User Study

Introduction & Consent Form

Field Study Tutorial

Ron will describe the field techniques, and you can try them outside the field station.

- Digital Pen/Notebook and normal Digital Camera
- Hotspot Linking using the SmartCamera
- Visual Specimen Tags

Field Protocol—Gall Survey at Jasper Ridge Biological Preserve

Survey *Line Transect B*, a 40m line transect from Oak 1 to Oak 2.

Every object within 1 m perpendicular to the line is inside the sampling area.

For each gall in the transect, write down the survey measurements in your notebook (Size, Color, Distance). Also, take a picture of each gall with the normal digital camera. Feel free to include observations of other items of interest along the line (*e.g.*, terrain features, animal scat and tracks) either in the same table with the gall information, or in another table.

- At each GREEN marker, use Hotspot Linking with the SmartCamera to link a photo of your choice to some place in your notebook. This photo can be a photo you browse to, or a photo you have just taken (using the SmartCamera's red button).
- At each RED marker, use the Visual Specimen Tags to take a picture of and annotate a specimen of your choice (oak leaf, soil sample, etc). Place this specimen in the envelope.

Break, Snacks,Water

Questionnaire Part I—Background

Lab Activities Tutorial

Ron will describe the lab software, including the ButterflyNet browser, which associates photographs with your notes, and the Multimedia Spreadsheet, which enables you to lay out your photographs with your data.

Lab Task

Create a multimedia spreadsheet describing your field day. The best spreadsheets will win a Jamba Juice or Bookstore gift card.

Questionnaire Part II—Evaluation of ButterflyNet

A.4 Los Tuxtlas Data

USER	D First	D Start Time	D End Time	D Total Time	D Num Keystrokes	D Num Backspaces	D Num Value Changes	D Num Cell Errors
A	N	1111722128387	1111722931642	803255	1142	41	186	8
B	N	1111735410907	1111735956511	545604	1150	36	198	2
C	N	1111719550260	1111720237358	687098	1105	60	189	3
D	N	1111815642235	1111816634272	992037	1229	83	197	6
E	Y	1111732508864	1111733233916	725052	1075	41	189	3
F	Y	1111726583694	1111727535693	951999	1095	51	194	1
G	Y	1111737064765	1111738150897	1086132	1256	135	214	11
H	Y	1111715963383	1111716783662	820279	1017	58	191	10
I	Y	1111729796664	1111730769343	972679	1202	67	213	9
				Mean:	Mean:	Mean:	Mean:	Mean:
				842682	1141	64	197	6

D = Digital
P = Paper

Percent Speedup:
4.33

Percent Speedup (if we can match the Paper transi
13.42

USER	Backspaces (D	Value Chang	KeyStrokes (	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9
A	-7.00	-1	16	4	5	4	6					
B	-4.00	-43	-156	5	7	7	7					
C	-54.00	-16	-410	X	6	7	4					
D	-29.00	0	-205	5	4	3	4					
E	18.00	0	4	2	3	4	2					Ask
F	-17.00	-6	-78	5	5	7	6					
G	20.00	0	23	3.5	6	7	7					
H	8.00	-1	-40	X	7	6	5					
I	8.00	11	24	5	6	6	5					

USER	Time Spent in InkNotes	D Startup to Page 1 (ms)	D Page 1 to 2	D Page 2 to 3	D Page 3 to 4	D Total Transition Net	D Entry
A	51282	33158	25807	27329	18977	**105271**	**697984**
B	32436	17014	21842	16574	11827	**67257**	**478347**
C	22443	13700	21061	22032	19178	**75971**	**611127**
D	68930	16874	37995	42742	18456	**116067**	**875970**
E	37275	19328	26328	15943	19287	**80886**	**644166**
F	76018	21230	49651	32697	25737	**129315**	**822684**
G	69219	21170	24034	26488	23363	**95055**	**991077**
H	78313	24546	40248	35781	25788	**126363**	**693916**
I	40298	19277	25657	33518	21430	**99882**	**872797**
						Mean:	Mean:
						99563	743119

USER	P Start Time	P End Time	P Total Time	P Num Keystrokes	P Num Backspa	P Num Value Chan	P Num Cell Erro	P Startup to Page
A	1111721054163	1111721967927	913764	1126	48	187	7	2604
B	1111734715917	1111735283764	567847	1306	40	241	3	6349
C	1111718541049	1111719439862	898813	1515	114	205	13	4136
D	1111814308247	1111815431122	1122875	1434	112	197	5	7792
E	1111733275316	1111733892523	617207	1071	23	189	2	2373
F	1111727598353	1111728543001	944648	1173	68	200	2	3014
G	1111738199857	1111739276575	1076718	1233	115	214	3	3085
H	1111716864178	1111717695153	830975	1057	50	192	4	2654
I	1111730850119	1111731804361	954242	1178	59	202	3	3996
			Mean:	Mean:	Mean:	Mean:	Mean:	
			880788	1233	70	203	5	

USER	P Page 1 to 2	P Page 2 to 3	P Page 3 to 4	P Total Transition Time	Net P Entry	Total Time (D-P)	Percentage Savings
A	1823	5258	4176	13861	899903	**-110509.00**	12.87
B	3646	5518	2334	17847	550000	**-22243.00**	4.00
C	3615	2934	4276	14961	883852	**-211715.00**	26.70
D	7461	6970	19888	42111	1080764	**-130838.00**	12.37
E	3115	4907	2734	13129	604078	107845.00	-16.07
F	2954	6569	2273	14810	929838	7351.00	-0.78
G	7351	3295	2514	16245	1060473	9414.00	-0.87
H	4096	8392	4917	20059	810916	**-10696.00**	1.30
I	5808	9985	2544	22333	931909	18437.00	-1.91
				Mean:	Mean:		
				19484	861,304		

A.5 ButterflyNet Data

Participa	Gender	Part I	Q1-Occupation	Q2-Discipline	Q3-Institution	Q4-Education	Q5-FieldE	Q6-Devices						
								Field Noteb	PDA	Audio R	Lapto	Video R	Film C	Digital C
A	Male		Field Researcher; V	Engineering	University	PhD	5-10	6		6	3			1
B	Female		Field Researcher; L	Ecology	University	4th year PhD prc	2-5	6	0	0	3	0	0	4
C	Male		Field Researcher; L	Ecology	University	PhD	12	6	0	0	4	0	6	6
D	Male		Field Researcher; S	Anthropological Science	University	College Degree	1-2	6	0	5	6	2	0	6
E	Female		Field Researcher; L	Ecology	University; Non-profit	Masters	12	6	6	0	3	5	6	5
F	Female		Field Researcher; L	Ecology	University; Non-profit	College Degree	1-2	6	2	0	2	0	0	4
G	Male		Lab Researcher; St	Ecology	University	Some College	1-2	6	0	0	0	0	0	5
H	Female		Professional	Other; Human Behavior	Government agency	Masters	None	0	0	0	2	0	2	1
I	Male		Field Researcher; P	Biology; Ecology; Ornith	University; Non-profit	PhD	15	6	0	3	1	2	6	6
J	Female		Professional	Ecology; Conservation E	University	PhD	18	0	0	0	0	0	1	3
K	Female		Field Researcher; L	Human Biology minor; E	University	Some College	< 1	3						
L	Male		Professional	Computer Science	Private Corporation	Masters	2-5	6						4
M	Female		Student	Biology	University	Some College	1-2	6	0	0	0	0	0	6
N	Female		Field Researcher; L	Bioogy; Ecology	University; Non-profit	Masters	> 10	6	0	0	0	0	0	6
Summary	**of Questionnaire Results**													
N	14		Everyone has at mir	Mostly Ecologists	University Affiliations	4 PhDs	4 with more than 10 years of field experience.							
Male	6					Three with Maste	Others vary	MEDIAN	MEI	MEDIAI	MEDI	MEDIAI	MEDIA	MEDIAN
Female	8					4 with college de	Two with m	6	0	0	1.5	0	0	4.5

Participa				Q7-Software In the Field	Q8-Notebooks per	Q9-Importance of F	Q10-# Pictures /	Q11-Additional Se	Q12-Significant Limiting I	Q13-Exact Dat
	GPS	Othe	Other							
A		Data Logg		Yes, I use software to download	1	1	0	7	Hardware	
B	5			Arc programs for GPS/GIS	2-3	3	1-5	6	Poor software support; Oth	
C	6				1	7	5-10	7	Battery power; Poor softwa	
D	4			I once took a class which emplo	2-3; Summers only t	7	10-20	3	Hardware; Battery power; F	
E	6			In the past, excel. We directly re	2-3	5	10-20	7	Hardware; Lack of time	
F	0	Sample co		A program for scanning root ima	1	6	1-5	2	Other: Still developing ways	
G	3			No	2-3	3	1-5	4	Poor software support; Toc	
H	0			No	0					
I	6	Humidity a		No	30	7	10-20	7	Hardware; Battery; Poor so	
J	3			Not while actually collecting data	1; Do not take most	3	Varies tremendou	Temperature, Hum	Lack of time; Other: Lack o	
K		Coin	Ziplo	No	1	5	None; but I think it	7	Hardware; Lack of time	7; especially for
L	6			Trimble GeoExplorer, Records (	0	3	1-5	7; Circled GPS	Fumbling with Equipment; I	
M	2	0	0	No	2-3	7	> 20	3	Battery power; Poor softwa	
N	3	Clipb	Digita	Word & Excel for entering data	1	7	10-20	7	Lack of Time	
Summary										
N				Some do; Some don't; At least t	30 is the ornithologist. Most will generate t	3 people responded 10-20; and 3 responded 1-5				
Male	**MEDIAN**					**MEDIAN**				**MEDIAN**
Female	3					5				

Participa	Q14-Organize Data on-t		Q15-Increase or Decrease Field Time		Part II	Q16-Usefulness of Features			Q17-Integrate ButterflyNet as		
	Digital Pen/Notebc		Hotspots	Visual Specimen Tags		Digital Pen/Notebc	Hotspot Linkin	Visual Specimen Ta	Digital Pe	Hotspot Linkin	Visual Sp
A	1	-2				4	1	1	4	1	
B	2	0	1	3		5	5	3	6	2	
C	7	0	1	0		7	6	7	7	7	
D	4	0	-2	-2		7	7	7	7	4	
E	7	3	2	2		6	4	4	7	5	
F	2	-3	0	1		7	7	3	7	7	
G	2	1	2	2		7	4	4	7	4	
H						7	6	7	4	3	
I	7	0	0	1		7	5	3	7	3	
J	6	3	3	3		2	2	3	1	1	
K	5	0	0	1		7	7	7	7	7	
L	2	-3	1	-1		6	6	6	7	6	
M	3	0	0	0		7	7	4	5	5	
N	4	3	3	3		7	7	7	7	7	
Summary											
N											
Male	**MEDIAN**		**MEDIAN**	**MEDIAN**		**MEDIAN**	**MEDIAN**	**MEDIAN**	**MEDIAN**	**MEDIAN**	**MEDIAN**
Female	3.5	0	0.5	1		7	6	4	7	4.5	

Participay	Net as part of you	Q18-What Software? For Organizing, Transcribing, Analyzing												Q19
	Visual Specimen Ta	Personal Organiz	Word Processi	Spreadshee	Database	Statistics	Web Brows	Photo Brow	Math Toc	Bibliograp	Mapping To	Other	Other	Pre
A	1	5	6	6		6	6							
B	2	0	6	6	0	6	3	3	3	6	6			
C	7		5	6	5	5	4	4	5	2	3	SigmaPlot t	GRAMS/32	
D	6	0	6	6	4	3	6	6	0	3	1			
E	3	0	6	6	6	5	6	6	0	5	3	Adobe Writ		
F	5		2	6	4	3	1	0	0	0	0			
G	2	1	6	6	3	6	2	3	3	4	4			
H	4	0	6	0	6	0	0	0	0	0	0			
I	1	6	3	6	5	6	2	6	1	6	6	HOBO Soft		
J	1	0	6	6	3	3	0	1	0	0	4			
K	7	0	3	3	0	0	3	0	0	0	0			
L	2	0	4	6	6	0	4	1	0	3	6			
M	2	1	6	6	1	4	5	5	2	1	1	Ace FTP (fi		
N	7	0	6	6	4	6	6	4	3	6	1			
Summary														
N														
Male	**MEDIAN**	**MEDIAN**	**MEDIAN**	**MEDIAN**	**MEDIAN**	**MEDIAN**	**MEDIAN**	**MEDIAN**	**MEDIAN**	**MEDIAN**	**MEDIAN**			**MEI**
Female	2.5	0	6	6	4	4.5	3.5	3	0	2.5	2			

Participa	Q19-Software for Publishing/Presenting						Q20-Time To Transcribe p	Q21-Methods to Find Relat	Q22-Helps a colleague understa	Q
	Presentat	Email	Word Process	PDF	Web Brows	Other				
A	4	4	5		4		0.1	browse handwritten annotatio	5	
B	6	6	6	6	3		1-2	use computer to seach; by m	2	
C	6	3	6	5	2	Sign	1-2 (I do it every 2 days)	browse handwritten annotatio	7	
D	6	6	6	4	3	Adob	1-2 (I do it at the end of the s	browse handwritten annotatio	7	
E	6	6	6	4	6	Sign	4-8 (very variable; I do it ever	browse; computer search; m	6	
F	6	6	6	6	4		0.5 (per entry period, which i	browse; dates + IDs	5	
G	6	2	6	5	3		1-2	browse; computer search	7	
H	2	6	6	0	0		I don't transcribe field data e	browse	7	
I	6	2	6	6	1	Delta	0, Other people enter my dat	browse; computer search; m	6	
J	6	Sending Files Or	To prepare pub	To submit pu	As required by J		1-2 (amount of time it takes t	browse; computer search; m	1	
K	3	6	4	0	0		None; other people enter my	browse; computer search; m	7	
L	4	5	4	3	4		2-4	computer search	2	
M	6	6	6	5	5	Spre	1-2	browse; by memory	6	
N	6	6	6	6	6		1-2	by memory	2	
Summary										
N										
Male	**MEDIAN**	**MEDIAN**	**MEDIAN**	**MEDIAN**	**MEDIAN**				**MEDIAN**	**M**
Female	6	6	6	5	3				6	

Participa	Q23-Helps me quickly rec	Q24-Less Time Transcribing D	Q25-Helps me capture/manage large amts of c	Q26-More or Less Time					
				Hypothes	Protoc	Field Data Cap	Data Organiza	Transcript	
A	7	5	3	4	4	3	3	3	
B	5	6	3	4	4	3	3	2	
C	6	7	6	4	3	2	2	1	
D	7	7	7	4	3	4	1	1	
E	5	7	7						
F	3	7	6	4	1	4	2	1	
G	7	5	6	4	4	5	5	7	
H	5	7	7	N/A	N/A	N/A	N/A	N/A	
I	7	7	5	4	4	5	3	2	
J	1	1	1	N/A	7	7	7	7	
K	7	7	7	4	4	5	2	1	
L	3	3	4			4	3	3	
M	5	7	7	5	5	3	1	1	
N	7	2	3	4	7	3	3	4	
Summary									
N									
Male	**MEDIAN**	**MEDIAN**	**MEDIAN**	**MEDIAN**	**MEDI**	**MEDIAN**	**MEDIAN**	**MEDIAN**	
Female	5.5	7	6	4	4	4	3	2	

Participa			Q27-Integrate BNet Lab Tools as part of your workflow				Q28-What do you like or find most use	Q29-What do you dislike o
	Data Analy	Paper Public	Browser	Navigation by Pe	Sending Data to Spreadsh	Multimedia Spreadsheet		
A	4	4	1	1	1	1	1) Integration of photos w/ notes. 2) ease	Software unreliable in part a
B	4	4	3	6	6	6	I like having everything linked together + tl	The spreadsheet interface is
C	3	4	7	6	7	7	The cool paper w/ built in time-stamping a	The spreadsheet as currentl
D	1	2	5	6	7	6	Spreadsheet sender is very valuable. The	It would be nice if photos cou
E			6	6	6	6	Potentials for future use! Quick digital do	When browsing photos, it wa
F	2	2			7	7	Tracking data by lines for entry of data in	No bugs! Better photo naviga
G	4	5	7	7	7	5	I like the fact that you can instantly have a	At some point in taking so m
H	N/A	N/A	N/A	N/A	N/A	N/A	The ability to immediately transfer data to	It would be helpful to not hav
I	5	5	3			2	Digital pen. Exporting table to Excel.	The bugs with photos.
J	7	7	1	1	1	1	In some cases having a digital backup of	cells in data sheet correspor
K	3	4	6	7	7	6	Digitized notebook as backup; ability to lir	I seemed to lose quite a bit c
L	2	4	2	2	4	4	Gathering transect data and subsequent a	The transcription process to
M	2	3	5	1	6	2	Sending to the Spreadsheet was very use	The spreadsheet program w
N	4	4	5	5	2	7	Backup of field notes + ability to make co	Handwriting recognition for d
Summary								
N								
Male	MEDIAN	MEDIAN	MEDIAN	MEDIAN	MEDIAN	MEDIAN		
Female	3.5	4	5	6	6	6		

Participa	Q30-Additional Features for true usefulness?	Coded Q28	Coded 29	Coded 30
A	To be able to download data from a bat detector an	LINK; EXP; RED	PEN	GPS/SENS
B	more user-friendly spreadsheet with features like e	LINK; BKUP	SS; HOTSPOT	SS; HANDREC; PENTOOLS
C	Once data from the special field book is pasted in t	EXP	SS	HANDREC; GPS/SENS
D	Managing two cameras can be a bit tricky. If hotspo	EXP; RED; LINK	BROWSER; SS	PENTOOLS
E	If this had a capability of automatically converting	EXP	BROWSER	HANDREC
F	Be able to choose which non-contiguous pages to	RED; LINK	BROWSER; SS	BROWSER
G	If the spreadsheet portion was compatible/integrate	LINK; BKUP	HOTSPOT	SS
H	See #29	BKUP	PENTOOLS	PENTOOLS
I	Handwriting recognition software. Assocation of ph	EXP	BROWSER	HANDREC; BROWSER
J	The effectiveness may be correlated positively w/ c	BKUP	HANDREC	
K	OCR technology-so handwritten notes could be sto	BKUP; RED; LINK; SHARE	PEN; BROWSER	HANDREC
L	Handwriting recognition of tabular data: grid --> s/s		SS	HANDREC; BROWSER
M	The digital pens w/ hotspot linking photographs wo	EXP; LINK	SS	
N	ImageJ? GPS/GIS?	BKUP; SHARE; LINK	HANDREC	GPS/SENS
Summary		LINK = Integration of Photos w/ Notes	PEN = Pen not robust enough	GPS/SENS = GPS or Sensor Data integ
N		EXP = Export to Spreadsheets	HOTSPOT = Didn't work as planned	HANDREC = Handwriting Recognition
Male		BKUP = Backup of Notes		PENTOOLS = Robustness or Handy fea
Female		SHARE = Sharing w/ Colleagues		SS = Better Spreadsheet Integration or T
		RED = Red Line Decreases Transcription Errors		BROWSER = Improvements to Visualiza

A.6 GIGAprints Data

							Scale of -3 to +3			
Participant Nar	M/F	Occupation			Hours of Compu	Familiar	Twistr Easy	Fun	Faste	See Mor
1	M	Researcher	Biology	Universit	PhD	> 50	5	3	3	3
2	F	Researcher/Postl	Biology	Universit	PhD	31-50	4	2	3	3
3	M	Researcher/Stude	Computer Scier	Universit	Mas	31-50	6	2	3	1
4	M	Researcher/Stude	Computer Scier	Universit	Mas	31-50	6	1	2	1
5	M	Professional	Computer Scier	Company	PhD	> 50	7	3	3	3
6	F	Professional	Law	Law Firm	J.D.	> 50	6	3	3	3
7	M	Professional	Biology; Educat	Universit	Coll	> 50	6	3	3	3
8	F	Facility Mgr.	Construction/Fa	Universit	Coll	> 50	3	1	1	0
9	M	Researcher/Stude	Biology	Universit	Mas	21-30	4	3	3	0
10	F	Researcher/Stude	Biology	Universit	Mas	21-30	5	3	3	1
11	F	Researcher	Biology	Universit	PhD	21-30	4	3	3	0
12	F	Professional	Geography	Universit	Coll	31-50	6	2	3	0
Means							5.17	2.42	2.75	1.50	2.67
Stdev							1.19	0.79	0.62	1.38	0.65
					0 to 6			5.42	5.75	4.50	5.67
					1 to 7			6.42	6.75	5.50	6.67

Participant Nar	Your Own Work?	Additional Features in Twistr
1	Easier to look at animal photos on sheets of paper rather than scroll up/down. Also necessary to compare injury levels on leaves	Group Photos on Screen or In Folders; Annotate Photos
2	Large set of photos of bobcats: Try to identify individuals.	Include stronger feedback for choosing the right photo.
3	Inspect technical diagrams of computer systems at large scales & high resolution.	Make double selection score higher (now relatively low re
4	Large scale / lot of pixels is useful.	maybe the number of points ticks down like a clock the l
5	I like the large workspaces, but to be useful for computer work, the display would need tob e dynamicall updatable.	It may be interesting to have players have separate sheet
6	No, law uses a lot of documents, but docs arenot as distinguishable as photos.	X
7	With vegetatio surveys, known plants cuold be shown and just tapped for each survey point where they are present. Outreach edu	X
8	Similar to library function in CADD but probably easier to find items.	Rather than a single large sheet, interactive catalog/broch
9	Could use it to identify species. Does the species look more like this photo? Or THIS photo?	X
10	Not in terms of findign photos.	X
11	for imperfect matches.. Such as displaying a real image of an organism and using a page of iconic images to match up.	One pen, but with a left/right click button.
12	X	X

	-3 to 3					
Participant Nar	Buddy Easy	Usefu	Collaborate Bett	Your Own Work?	Additional Features?	
1	1	3	3	Plan Experiments when CoWorker is away; e.g., deciding location	Erase =); Grid on the paper to know where you are drawing on the screen;	
2	2	3	3	In general some things are easier to explain when you can show a	have some preset shapes to make drawing easier. Copy/Paste & Erase…	
3	2	2	2	"More intuitive digital whiteboard. Pen/Paper/Hybrid experience wa	Register a paper position w/ screen position without making a mark (either	
4	-3	-2	-3	I do visual stuff a lot, and a shared drawing space is useful. It cuol	Registration! Maybe use projector to show my buddies strokes on my pape	
5	3	0	0	Most conferencing I do does not involve much writing.	Erase. See cursor position, to collaborate with other people's sketches.	
6	2	1	1	Could be used w/ legal pad for taking notes and digitally recording	Be able to see where the other person's drawing you are drawing on. A text	
7	3	3	2	Maybe, altho currently I don't ork w/ many remote partners	Easily find a point on the paper where the other person was drawing or writi	
8	-1	0	1	Minimally	Able to enter a full sheet background to share for mutual viewing or editing.	
9	2	2	3	Could be very useful in: keeping calendar for scheduling a LOT of d	To make it easier to draw on top of someone else's drawing. Could have a g	
10	-1	2	2	Probably-I could see it making remote collaboration a lot easier. E	it would be nice to be able to see where your partner's sketches show up o	
11	-1	2	2	Computer shows a shared document you wanted to revise/edit. Th	Have matching monitors. Person 1's photo is on left of both machines.	
12	1	-2	0	I don't typically have remote/video conf meetings.	Reference grid on paper + screen. ERASE.	
Means	0.83	1.17	1.33			
Stdev	1.90	1.80	1.72			
	3.83	4.17	4.33			
	4.83	5.17	5.33			

	AudioGuide					
Participant	Easy	Usefu	More Mobil	Useful Outdoc	Your Own Work?	Additional Features
1	3	3	3	3	To get field sites that are not our own; Add info like exact location and data	collected on sites; pictures or text/numbers (retrieved to some display)
2	1	3	3	3	Walk around an experimental plot in the field and retrieve information on the li	GPS
3	1	-1	3	-1	Nope; I do not work outdoors.	Audio device is slowe than reading off paper. However digital device might
4	-2	1	3	1	Not in my work; but might be useful as a navigational aid; Why not print numl	2 headpieces or speaker so we can both hear. Show me on the computer
5	3	3	3	3	Audio feedback is not a big part of my work, nor is being outside.	make new audio segments stop old ones.
6	3	2	3	3	No.	Audio output for multiple users simultaneously.
7	2	1	1	1	Not sure at this pont.	X
8	2	2	2	2	X	X
9	3	3	2	3	When training volunteers, the earpiece can remidn them to do certaint asks.	Have a GPS cmoponent, so the perosn doesn't' have to know exactly whe
10	-1	2	3	0	Possibly if an audioguide type map could be combined with data entry capab	Paper map unweildy. Smaller map, or map folded into moer cmopact broc
11	3	3	3	2	Providing instructiosn or reminders on what u need to do at a series of sites.	Would be handy to hang the pen around your neck so u don't' lose it, or t
12	3	-2	0	2	Not sure.	Direction of travel. View awareness.
Means	1.75	1.67	2.42	1.83		
Stdev	1.71	1.67	1.00	1.34		
	4.75	4.67	5.42	4.83		
	5.75	5.67	6.42	5.83		

	GIGAprints in General		Speed… or S	-3 to 3			Usefulne	1 to 7					
Participant	Visually Locate		Retrieve Phot	Work Away	Multiple Per	Work with Coll	Work by Mys	Digital Pe	Large Shee	Multiple Per	See Lots of	Reconfigure Prir	Collaborate
1	3		3	2	3	3	2	7	6	6	7	6	7
2	3		3	3	3	3	2	7	7	7	7	7	7
3	3		0	-2	1	2	2	6	5	2	7	2	5
4	1		1	1	0	1	0	2	7	1	7	2	3
5	3		3	3	3	3	3	4	7	4	7	7	4
6	0		0	0	0	0	0	7	7	7	7	6	6
7	1.5		2	3	1	3	2	7	5	5	7	7	7
8	1		1	1	0	0	1	7	4	3	4	5	5
9	3		3	3	0	3	3	7	7	3	7	3	7
10	0		0	2	0	3	0	7	2	7	4	4	7
11	0		1	3	1	3	1	7	6	5	6	6	7
12	2		2	0	0	1	0	6	5	5	5	5	6
Means	1.71		1.58	1.58	1.00	2.08	1.33	6.17	5.67	4.58	6.25	5.00	5.92
Stdev	1.29		1.24	1.62	1.28	1.24	1.15	1.59	1.56	2.02	1.22	1.86	1.38
	4.71		4.58	4.58	4.00	5.08	4.33	5.17	4.67	3.58	5.25	4.00	4.92
	5.71		5.58	5.58	5.00	6.08	5.33	6.17	5.67	4.58	6.25	5.00	5.92

Participant	Overall Like
1	That the files can be manipulated by MORE than one person at a time (multiple pens/users) and that can help store much more data than test notes.
2	Visualize large set of photos; link photos w/ laptop
3	I think the digital pen is very useful. With super high res tablet displays, paper's utility might be more limited. But just seeing the pen's markign on paper made writing w
4	large size with lots of detail; this is the only thing that really distinguishes it.
5	Like the large workspace
6	Multiplayer game is fun
7	Ease of visually scanning a large piece of paper. The fact that several people can be looking at the same large piece of paper & interacting with it.
8	Handy input form.
9	I like that you can easily keep track of what you've already looked at. That you always would have a backup copy in case of rain/battery failure.
10	Just the basic concept-digital pen and paper- is very useful and flexible idea. Of the appliations today, the conferencing application seemed particularly powerful & usefu
11	**I especially like the idea of taking an airphoto out in the field and being able to identiy a location and record something about it. It feels natural but pow**
12	Normal writing with a pen. Computer part is transparent.

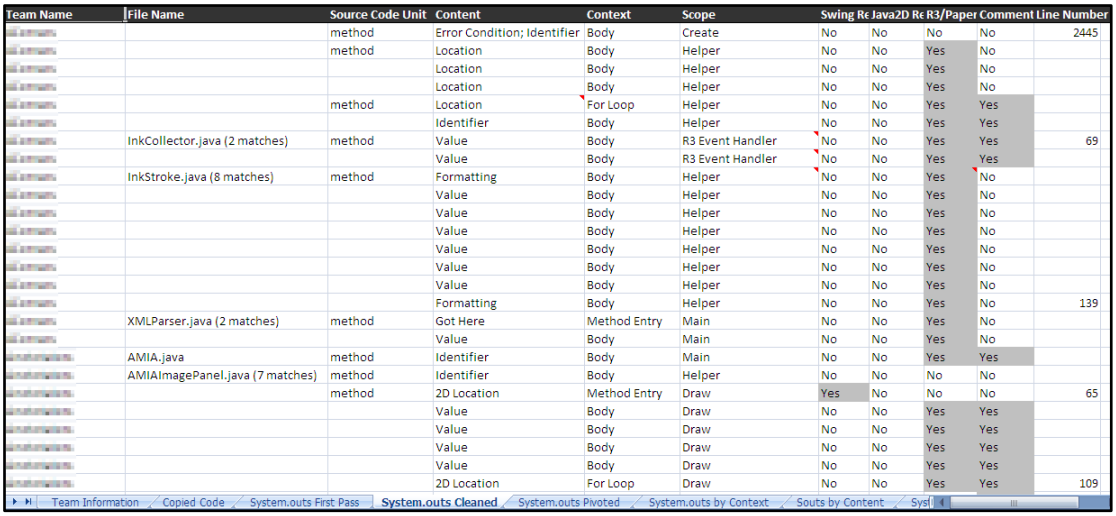
Dislike	Additional Features
ıore data than test notes.	
Fewer computers involved would be nice.	Classify photos using the printout and store them on my computer. Compare how others class
Collaborative aspects might be more valuable than organizing photos, since organizing is such a	Interesting to have digital desktop UI design changes assuming gigaprint interface exists. Migl
hard to see advantages over alternatives like tablets for interaction/collaboration	I think integration with a projector has a lot of potential.
Digital documents I can erase or undo, gigaprints doesn't seem as good about that.	Moving or reconfiguring prints would be useful.
Could not see in BuddySketch where I'm drawing on other person's work	Digitize handwritten text and make them searchable by date and word.
None.	This could be very useful now--both for collaboration and for a teaching tool. i.e. JR stuff could
Lag time between actual writing/drawing and image appearance on screen.	Quick refresh or rectifying. How do you erase clear something? Why not use pencils? Integrat
For the cap not to fall off.	To be able to use write-in-the-rain paper.
Would be a lot more useful along with good handwriting recognition. Today's emphasis on large	Record data directly from pen & paper to a spreadsheet would be good.
Not knowing where u are on the screen w/ buddy sketch, until u put your pen down. Save scree	Try gigaprints in context of laying out a photo rich document, where u want to be able to easily
Handling large sheets of paper.	There's definitely something to it... but exactly what is yet to be determined.

A.7 PaperToolkit Data

The data came from 1) manually reviewing the source code of 17 projects and 2) scripts that processed the code to create statistics (*e.g.*, number of *println*s). The first table presents the number of classes and lines of code each project contained.

Number (Rando Team "Name"								
		Effort (Person-Months)	Num Statements	Num Stmts PTK	Num Stmts Not	Num Classes	Num PTK	Num Not PTK
2	B	9.45	**3688**	16	3672	**40**	1	39
17	Q	9.90	**3857**	246	3611	**37**	13	24
11	K	8.03	**3159**	156	3003	**34**	6	28
1	A	5.03	**2024**	98	1926	**34**	4	30
6	F	6.75	**2679**	259	2420	**30**	7	23
12	L	4.51	**1824**	198	1626	**21**	10	11
10	J	6.70	**2659**	116	2543	**20**	4	16
16	P	4.77	**1922**	163	1759	**20**	10	10
5	E	2.05	**860**	200	660	**19**	5	14
7	G	4.50	**1819**	0	1819	**17**	5	12
13	M	5.20	**2087**	132	1955	**15**	7	8
15	O	3.18	**1309**	236	1073	**15**	10	5
9	I	6.24	**2483**	96	2387	**14**	10	4
4	D	3.74	**1527**	363	1164	**13**	10	3
8	H	4.46	**1804**	242	1562	**10**	5	5
3	C	1.92	**807**	181	626	**10**	4	6
14	N	2.95	**1218**	286	932	**8**	6	2
		Mean	Sum			Median		
		5.26	**35726**			**19**		
		Mean	Mean			Mean		
		5.26	**2101.53**	**175.76**		**21.00**		
		STDEV	STDEV			STDEV		
		2.34	**892.55**			**10.17**		

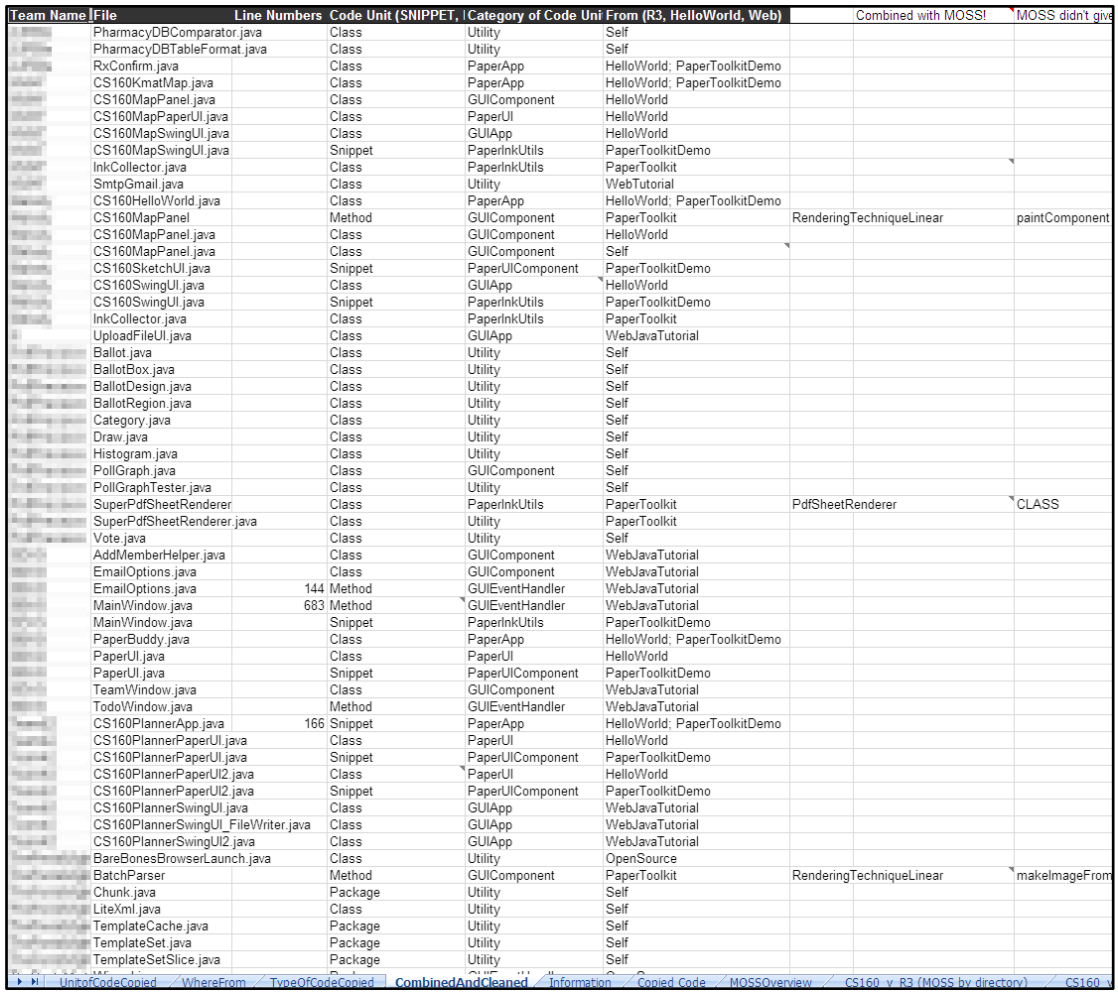
To study how students used *println*s to debug their paper applications, we searched for *println*s in Eclipse, and categorized each by hand. Following is an excerpt (1232 total entries):



Team Name	File Name	Source Code Unit	Content	Context	Scope	Swing Re	Java2D Re	R3/Paper	Comment	Line Number
[redacted]		method	Error Condition; Identifier	Body	Create	No	No	No	No	2445
[redacted]		method	Location	Body	Helper	No	No	Yes	No	
[redacted]			Location	Body	Helper	No	No	Yes	No	
[redacted]			Location	Body	Helper	No	No	Yes	No	
[redacted]		method	Location	For Loop	Helper	No	No	Yes	Yes	
[redacted]			Identifier	Body	Helper	No	No	Yes	Yes	
[redacted]	InkCollector.java (2 matches)	method	Value	Body	R3 Event Handler	No	No	Yes	Yes	69
[redacted]			Value	Body	R3 Event Handler	No	No	Yes	Yes	
[redacted]	InkStroke.java (8 matches)	method	Formatting	Body	Helper	No	No	Yes	No	
[redacted]			Value	Body	Helper	No	No	Yes	No	
[redacted]			Value	Body	Helper	No	No	Yes	No	
[redacted]			Value	Body	Helper	No	No	Yes	No	
[redacted]			Value	Body	Helper	No	No	Yes	No	
[redacted]			Value	Body	Helper	No	No	Yes	No	
[redacted]			Value	Body	Helper	No	No	Yes	No	
[redacted]			Formatting	Body	Helper	No	No	Yes	No	139
[redacted]	XMLParser.java (2 matches)	method	Got Here	Method Entry	Main	No	No	Yes	No	
[redacted]			Value	Body	Main	No	No	Yes	No	
[redacted]	AMIA.java	method	Identifier	Body	Main	No	No	Yes	Yes	
[redacted]	AMIAImagePanel.java (7 matches)	method	Identifier	Body	Helper	No	No	No	No	
[redacted]		method	2D Location	Method Entry	Draw	Yes	No	No	No	65
[redacted]			Value	Body	Draw	No	No	Yes	Yes	
[redacted]			Value	Body	Draw	No	No	Yes	Yes	
[redacted]			Value	Body	Draw	No	No	Yes	Yes	
[redacted]			Value	Body	Draw	No	No	Yes	Yes	
[redacted]			2D Location	For Loop	Draw	No	No	Yes	Yes	109

Team Information | Copied Code | System.outs First Pass | System.outs Cleaned | System.outs Pivoted | System.outs by Context | Souts by Content | Sys(

To identify copied code, we used a combination of MOSS and text search. In the end, text search and manual review yielded better results. An excerpt from the entries is below:



Team Name	File	Line Numbers	Code Unit (SNIPPET,	Category of Code Uni	From (R3, HelloWorld, Web)		Combined with MOSS!	MOSS didn't give
	PharmacyDBComparator.java		Class	Utility	Self			
	PharmacyDBTableFormat.java		Class	Utility	Self			
	RxConfirm.java		Class	PaperApp	HelloWorld; PaperToolkitDemo			
	CS160KmatMap.java		Class	PaperApp	HelloWorld; PaperToolkitDemo			
	CS160MapPanel.java		Class	GUIComponent	HelloWorld			
	CS160MapPaperUI.java		Class	PaperUI	HelloWorld			
	CS160MapSwingUI.java		Class	GUIApp	HelloWorld			
	CS160MapSwingUI.java		Snippet	PaperInkUtils	PaperToolkitDemo			
	InkCollector.java		Class	PaperInkUtils	PaperToolkit			
	SmtpGmail.java		Class	Utility	WebTutorial			
	CS160HelloWorld.java		Class	PaperApp	HelloWorld; PaperToolkitDemo			
	CS160MapPanel		Method	GUIComponent	PaperToolkit	RenderingTechniqueLinear		paintComponent
	CS160MapPanel.java		Class	GUIComponent	HelloWorld			
	CS160MapPanel.java		Class	GUIComponent	Self			
	CS160SketchUI.java		Snippet	PaperUIComponent	PaperToolkitDemo			
	CS160SwingUI.java		Class	GUIApp	HelloWorld			
	CS160SwingUI.java		Snippet	PaperInkUtils	PaperToolkitDemo			
	InkCollector.java		Class	PaperInkUtils	PaperToolkit			
	UploadFileUI.java		Class	GUIApp	WebJavaTutorial			
	Ballot.java		Class	Utility	Self			
	BallotBox.java		Class	Utility	Self			
	BallotDesign.java		Class	Utility	Self			
	BallotRegion.java		Class	Utility	Self			
	Category.java		Class	Utility	Self			
	Draw.java		Class	Utility	Self			
	Histogram.java		Class	Utility	Self			
	PollGraph.java		Class	GUIComponent	Self			
	PollGraphTester.java		Class	Utility	Self			
	SuperPdfSheetRenderer		Class	PaperInkUtils	PaperToolkit	PdfSheetRenderer		CLASS
	SuperPdfSheetRenderer.java		Class	Utility	PaperToolkit			
	Vote.java		Class	Utility	Self			
	AddMemberHelper.java		Class	GUIComponent	WebJavaTutorial			
	EmailOptions.java		Class	GUIComponent	WebJavaTutorial			
	EmailOptions.java	144	Method	GUIEventHandler	WebJavaTutorial			
	MainWindow.java	683	Method	GUIEventHandler	WebJavaTutorial			
	MainWindow.java		Snippet	PaperInkUtils	PaperToolkitDemo			
	PaperBuddy.java		Class	PaperApp	HelloWorld; PaperToolkitDemo			
	PaperUI.java		Class	PaperUI	HelloWorld			
	PaperUI.java		Snippet	PaperUIComponent	PaperToolkitDemo			
	TeamWindow.java		Class	GUIComponent	WebJavaTutorial			
	TodoWindow.java		Method	GUIEventHandler	WebJavaTutorial			
	CS160PlannerApp.java	166	Snippet	PaperApp	HelloWorld; PaperToolkitDemo			
	CS160PlannerPaperUI.java		Class	PaperUI	HelloWorld			
	CS160PlannerPaperUI.java		Snippet	PaperUIComponent	PaperToolkitDemo			
	CS160PlannerPaperUI2.java		Class	PaperUI	HelloWorld			
	CS160PlannerPaperUI2.java		Snippet	PaperUIComponent	PaperToolkitDemo			
	CS160PlannerSwingUI.java		Class	GUIApp	WebJavaTutorial			
	CS160PlannerSwingUI_FileWriter.java		Class	GUIApp	WebJavaTutorial			
	CS160PlannerSwingUI2.java		Class	GUIApp	WebJavaTutorial			
	BareBonesBrowserLaunch.java		Class	Utility	OpenSource			
	BatchParser		Method	GUIComponent	PaperToolkit	RenderingTechniqueLinear		makeImageFrom
	Chunk.java		Package	Utility	Self			
	LiteXml.java		Class	Utility	Self			
	TemplateCache.java		Package	Utility	Self			
	TemplateSet.java		Package	Utility	Self			
	TemplateSetSlice.java		Package	Utility	Self			

UnitofCodeCopied / WhereFrom / TypeOfCodeCopied / CombinedAndCleaned / Information / Copied Code / MOSSOverview / CS160_v_R3 (MOSS by directory) / CS160_v

A.8 Human Subjects Information

Studies conducted as part of this dissertation research were covered by Stanford University human subjects protocols. ButterflyNet is covered under IRB-approved protocol 82471. GIGAprints is covered under protocol 83056. The PaperToolkit research is covered under protocol 83443, and is titled "A Software Toolkit for Paper Application Mashups."

B

Algorithms and Tools

Appendix B describes several of the algorithms used in ButterflyNet, GIGAprints, and PaperToolkit. We present alternative algorithms, and discuss tradeoffs. Finally, we also present several of the tools that help users work with PaperToolkit.

B.1 Correlating Media with Ink Strokes

Correlating media (*e.g.*, photos) with ink strokes allows end users to find data files related to handwritten notes (and vice versa). We treat each stream of data (*e.g.*, ink, photos, GPS logs) as a stream of <OBJECT, TIMESTAMP> pairs, and provide methods to find related objects of one type by presenting one or more objects of another type.

The simplest method provides the nearest N matches. This will work for finding the nearest 5 photographs to a particular ink stroke. However, using the algorithm naïvely may provide useless results in some cases (*e.g.*, finding ink strokes related to a particular photo would return the 5 nearest ink strokes). Thus, the toolkit provides clustering algorithms, which allow us to work with groups of ink strokes (by hour, day, or page).

The simplest clustering algorithm is a one-pass algorithm that groups ink strokes within a time (M milliseconds) or space window (P pixels). A hierarchical clustering solution would allow correlations at different granularities—a GUI slider could allow the user to

specify the grouping interactively. Tools like ButterflyNet would benefit from such features.

B.2 Converting Batched Input to Real-time Input

With the Nokia SU-1B pen, batched and real-time data are formatted differently. Batched data is formatted by the Anoto SDK, which provides coordinates relative to page addresses (*e.g.*, X=62.5 Y=512.625 on page 33.0.16.25). Real-time data is sent through the serial port as packets which contain coordinates in the absolute Anoto space (*e.g.*, X=281057930.375 and Y=195.875). The event dispatcher locates regions based on absolute coordinates. Thus, to support event handling on each pen synch, PaperToolkit must convert the incoming batched coordinates to real-time coordinates before sending them to the dispatcher. This requires a calibration procedure, because the Anoto transformation function is not published, and the function is not be linear with respect to page addresses (pages are of different sizes, and coordinates are processed through functions defined by **.pad* files).

Fortunately, the pen saves strokes to its memory even when in streaming mode. To calibrate a pen for a particular notebook, the developer puts the pen into streaming mode by tapping on the STREAMING ON/OFF paper button. She starts the calibration tool, and then marks on several different pages within a notebook. During this time, the calibration tool is logging the real-time coordinates. After turning off streaming, she places the pen into the cradle. The calibration tool receives the batched input, and pairs each batched sample with a real-time sample, based on their timestamps. This mapping is saved to a calibration file. The toolkit interpolates and extrapolates absolute coordinates by applying the mapping function to any incoming batched coordinates.

B.3 Rendering Digital Ink

Anoto digital pens capture ink as individual samples (X, Y, force, timestamp) from 35 to 70 times per second (or even higher, depending on the pen model). For example, the Nokia SU-1B samples at 35Hz in streaming mode, but at 70Hz in batched mode.

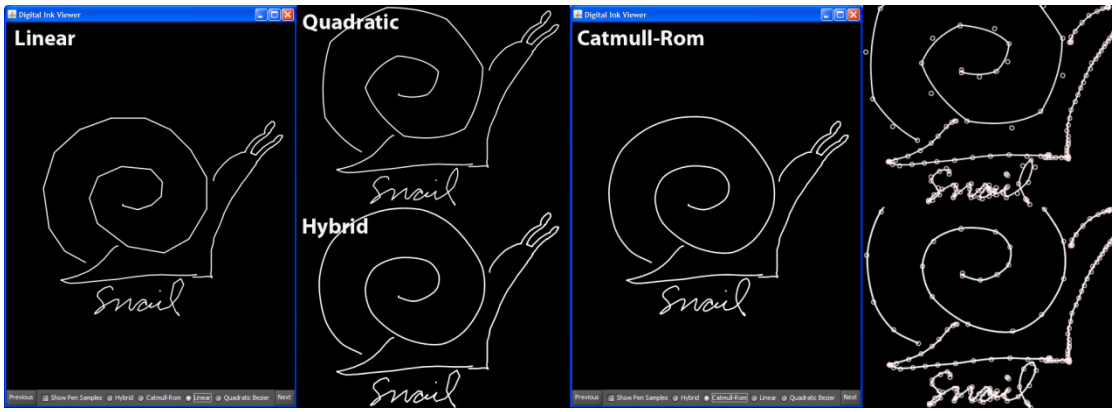


FIGURE B.1. *Left*) The simplest rendering algorithm draws lines between incoming samples. *Middle*) Quadratic curves make drawings look smoother. Catmull-Rom splines interpolate every sample to produce great results. A hybrid solution can enhance performance by using linear rendering when samples are close, and Catmull-Rom when samples are spaced farther apart. *Right*) The naïve quadratic curve algorithm uses samples as control points, so the digital ink does not pass through all points. Catmull-Rom splines pass through all pen samples.

The simplest way to render ink in a GUI is to draw line segments between each sample. However, the connections can become visible if the user draws quickly (see Figure B.1, left). For better rendering of handwriting, a second method is to treat the incoming samples as control points on a quadratic Bézier curve; the rendered curves look smoother, but are not completely accurate, since quadratic curves do not interpolate their control points. The best solution is to use Catmull-Rom splines, which pass through all of their control points.

Spline rendering may have a negative impact on rendering performance for pages packed with dense handwriting. Thus, PaperToolkit uses a hybrid solution—linear rendering for densely spaced samples, and Catmull-Rom for large and fast strokes. Future pens may sample higher rates (*e.g.*, > 100Hz); with better sampling, we can return to using linear interpolation between samples.

B.4 Using the Toolkit

PaperToolkit is an open-source Java toolkit hosted at: http://papertoolkit.googlecode.com. To use the toolkit, download it with a subversion client (*e.g.*, Tortoise SVN for Windows). Unfortunately, due to dependencies on Anoto software, the toolkit currently does not work with Mac OS X. To receive batched data from the pens, you will also need to acquire Anoto's

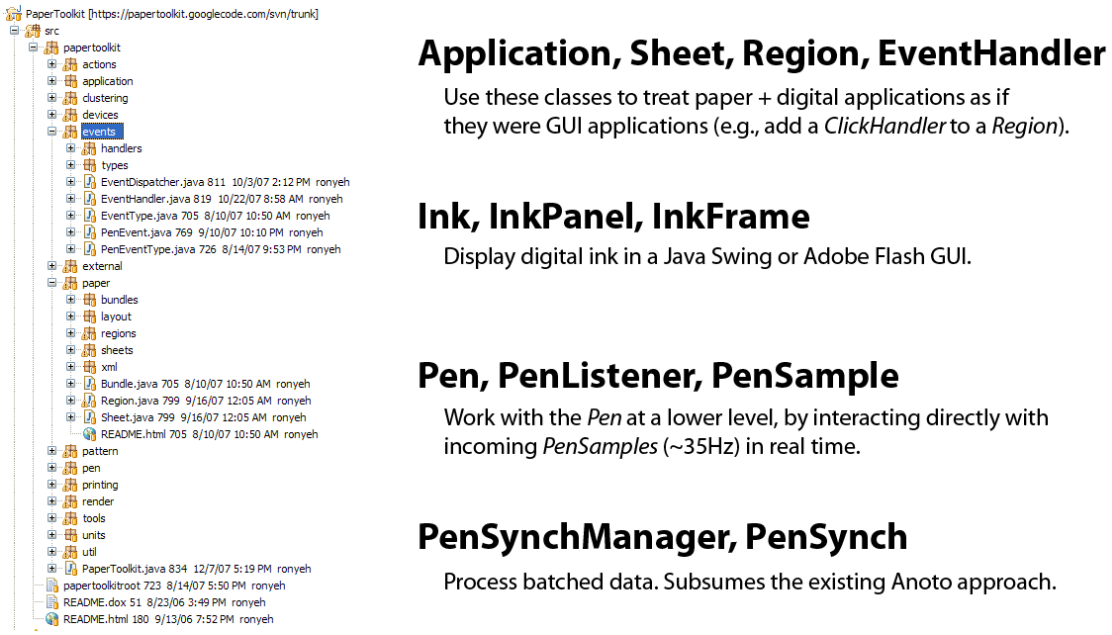


Figure B.2. To start using the toolkit, import the project into the Eclipse IDE. Example code is available at: http://papertoolkit.googlecode.com/svn/demos/trunk/.

SDK (http://www.anoto.com). PaperToolkit only supports Nokia SU-1B pens for streaming, but can be extended to support other pens once Anoto makes the streaming segment (70.*.*.*) publicly available.

The toolkit allows developers to make paper+digital applications using the familiar GUI programming model. The core part of the API that handles this is distributed across the the *application*, *events.handlers*, and *paper* packages (see Figure B.2). The source code of these applications look like Java Swing applications (see Figure 6.2).

However, it is possible to use PaperToolkit as simply an interface to the pen, in real-time and batched modes. To work with the pen in real-time, instantiate a *Pen* object and add a *PenListener* to it:

```
Pen pen = new Pen();
pen.startLiveMode();
pen.addLivePenListener(getPenListener());

    ..............................

private static PenListener getPenListener() {
  return new PenListener() {
```

```
        public void penDown(PenSample sample) {
           System.out.println("Pen Down: " + sample);
           down = sample;
        }

        public void penUp(PenSample sample) {
           System.out.println("Pen Up: " + sample);
           if (up.y > down.y) {
              System.out.println("Dragged Down");
              robot.keyPress(KeyEvent.VK_PAGE_DOWN);
              robot.keyRelease(KeyEvent.VK_PAGE_DOWN);
           } else {
              System.out.println("Dragged Up");
              robot.keyPress(KeyEvent.VK_PAGE_UP);
              robot.keyRelease(KeyEvent.VK_PAGE_UP);
           }
        }

        public void sample(PenSample sample) {
           System.out.println(sample);
           // last sample will be checked by the penUp method
           up = sample;
        }
     };
  }
```

For example, the code above talks to the streaming pen directly. When the pen lifts (*penUp*), the code examines the samples to ask PowerPoint to navigate to the next or previous slide. To work with batched data (stored in PaperToolkit\penSynch\data\XML), PaperToolkit provides a *PenSynch* abstraction:

```
PenSynchManager synchManager = new PenSynchManager();
PenSynch penSynch = synchManager.getMostRecentPenSynch();
List<Ink> importedInk = penSynch.getImportedInk();
for (Ink ink : importedInk) {
   DebugUtils.println(ink.getNumStrokes());
}

// Visualize This in a GUI Frame...
InkFrame inkFrame = new InkClustersFrame();
inkFrame.setInk(importedInk);
```

As seen above, digital ink that is retrieved from the *PenSynch* can be displayed in a Java GUI (*InkFrame* and its subclasses). The *InkFrame* offers an easy way to visualize synchronized digital ink (see Figure B.3). The next and previous buttons allow for navigation between the different pages of the synchronized notes.

A useful tool for debugging a streaming pen connection is the *PenCoordinateDebugger*. When running, it prints out all incoming pen samples to the text console. The coordinates are in raw Anoto format (*i.e.*, not page-relative):

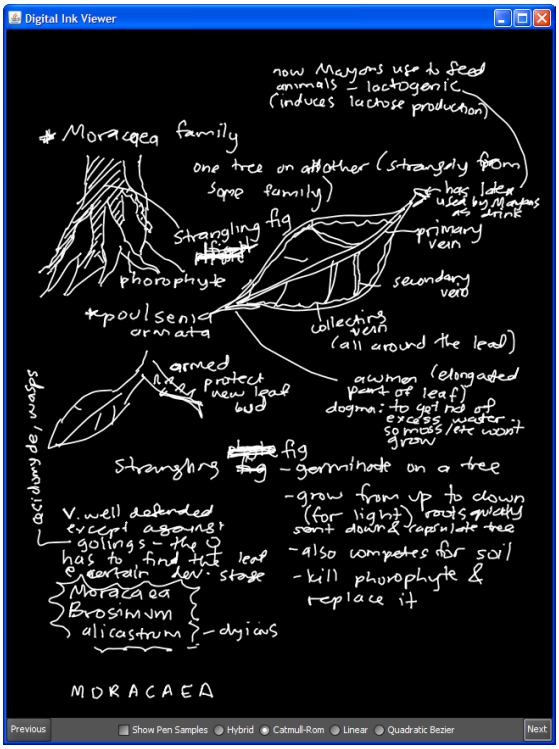


FIGURE B.3. Synchronized ink can be viewed in an InkFrame (see *InkRendererFrame.java*).

```
Down [Debugging Pen
Listener]: Sample:
[4448457.500,
8389734.250]
  F=0 T=1197514799812
P=DOWN
[Debugging Pen Lis-
tener]: Sample:
[4448456.875,
8389735.125] F=0
  T=1197514800093
P=DOWN
[Debugging Pen Lis-
tener]: Sample:
[4448456.750,
8389735.750] F=36
  T=1197514800093
P=DOWN
[Debugging Pen Listener]: Sample: [4448457.125, 8389736.500] F=60
    T=1197514800093 P=DOWN
[Debugging Pen Listener]: Sample: [4448459.375, 8389736.875] F=78
    T=1197514800093 P=DOWN
[Debugging Pen Listener]: Sample: [4448466.750, 8389736.250] F=84
    T=1197514800093 P=DOWN
...........more samples here.............
Up [Debugging Pen Listener]: Sample: [4448505.500, 8389774.000]
    F=0 T=1197514800421 P=UP
```

C

Digital Pen and Paper

Appendix C presents tips for working with Anoto's digital paper platform. We cover how PaperToolkit receives batched and real-time data from the digital pens, and how it translates the raw coordinates into region-relative coordinates. This text refers to specific locations in the PaperToolkit source code. To follow along, use a subversion (SVN) client to download a copy from http://papertoolkit.googlecode.com/svn/trunk.

C.1 Receiving Pen Data

The Anoto SDK allows developers to create .NET applications that receive notifications when a user drops his pen into the USB-cradle. PaperToolkit includes a C# program that registers itself with the Anoto platform. This information is kept in the Windows registry.

In the `PaperToolkit/penSynch/Monitor` directory, the `Monitor.cs` file contains the code that receives data from Anoto. When a pen connects to the computer, the pen driver reads the data and notifies our monitoring program. To receive the notification, the user must install the pen driver software (*e.g.*, Nokia SU-1B's Docking Director or software that comes with the Logitech or Maxell pens). The `Anoto.Notification.IDataReceiver.Notify(string category, object data)` function receives the synchronization data, processes it, and writes out an XML file.

At this point, the batched pen data is serialized, and any program can operate on it. PaperToolkit launches to detect if there is a running program listening at port 9999. If so, the path of the newly saved XML file is sent to the application.

To get the monitoring program hooked into Anoto's infrastructure, we need to first add entries into the Windows registry and then add `*.pad` files that handle the notebooks our application will work with (see `PaperToolkit/penSynch/Registration/Register.cs`). This program adds entries to bind the path of the COM object (a DLL) to the name (a ProgID) specified in `Monitor.cs`.

All the `*.pad` files are copied by the registration program to the right location (`C:\Documents and Settings\All Users\Application Data\Anoto\PADs`). To direct the pen driver to send data to our monitor and not to the default software, we edit the `*.pad` files to point to the PaperToolkit program. Each of the files in the `penSynch/PADs` directory contains lines that specify the target program by name:

```
<application_info name="paper_category" value="fieldtools::R3"/>
```

The value field matches the name specified in `Monitor.cs`:

```
string Anoto.Notification.IDataReceiver.Category {
    get {
        return "fieldtools::R3";
    }
}
```

After registration is run (`penSynch/bin/Registration.exe`), this information will be reflected in the Windows registry in the following key: `HKEY_LOCAL_MACHINE\SOFTWARE\Anoto\2.0\NotificationHandler\Subscriptions\Data\`.

Streaming

The streaming pens we used for PaperToolkit (Nokia SU-1B) are now discontinued. These Nokias are great for research and development, because they can be kept plugged in (and in streaming mode) for long debugging sessions. Additionally, the SU-1B can stream from any dot-patterned paper. The user turns on streaming by tapping on a special paper button.

The newer pens (Logitech 102 and Maxell DP-201) require a special segment of Anoto patterned paper (from address 70.*.*.*). They will not stream from standard notebooks, but will automatically start streaming when the user works with segment 70 paper. LiveScribe's pens currently do not support streaming.

The PaperToolkit streaming package supports the SU-1B pens, since Anoto has not made segment 70 paper available for public purchase. In the `papertoolkit.pen.streaming` package, the *PenStreamingConnection* class talks to a serial port (*e.g.*, COM5) attached to a Bluetooth connection on a Windows PC. Once paired and streaming, the Nokia pen looks for a port named ANOTO STREAMING (space required). To read from this port, PaperToolkit uses the JavaCOMM libraries with the `win32com.dll` file accessible in the system path.

C.2 Pattern Files

Anoto also sells a Forms Development Kit (FDK) so that developers can create PostScript files that include the dot pattern. To create pattern, developers first purchase a license (an XML file specifying the ranges the FDK can render). To use the FDK, the developer provides a PDF to the FDK software (an Adobe Acrobat Pro plug-in). The result is a PS file containing dot pattern overlaid on the images in the source PDF.

The PS file consists of 1) header code that defines the radius and font for the dots, and 2) a large section encoding the pattern, which looks like many lines of:

```
dudlrluudldrlrrulrdurlrdrr...
```

This code specifies the jitter of the dots—up, down, left, or right. The first version of PaperToolkit read in this information to render patterned PDF files. However, recent versions of Acrobat have trouble reading and printing files containing too much pattern. PaperToolkit now renders paper interfaces to a PS file, and then overlays pattern by inserting chunks of pattern information into the PostScript code.

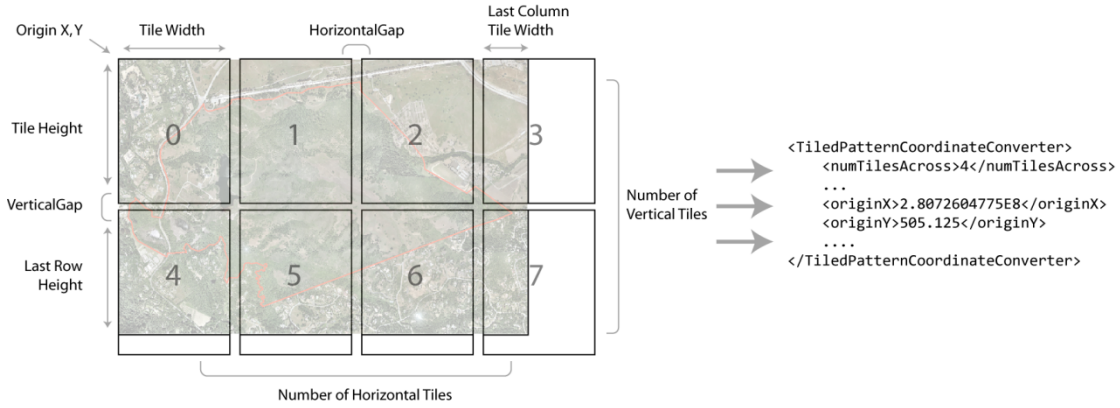


FIGURE C.1. To create a large print, we tiled consecutive pages of dot pattern (*e.g.*, pages 0 through 7). If a user drags a pen through tiles 1, 5, and 6, the pen will report a PEN DOWN on tile 1 and a PEN UP on tile 6. The coordinates are discontinuous in Anoto space, but are translated to print-relative coordinates before being dispatched to event handlers.

C.3 Tiling Dot Pattern for Large Prints

Ideally, a print driver would manage the rendering of dot pattern onto any sheet of paper. Larger prints would simply require larger contiguous segments of the pattern space. However, Anoto does not provide software to render large sheets of dot pattern. The provided pattern license is a *wide* and *short* segment of the pattern space. That is, for consecutive letter-sized sheets, the X-coordinate starts from `pageZeroOffsetX + pageNum * (pageWidth + horizontalGap)`, and the Y-coordinate starts from `pageZeroOffsetY`.

To create large sheets, PaperToolkit tiles the letter-sized chunks of pattern (see Figure C.1). For each sheet, it saves out a **.patternInfo.xml* file describing the tiling. This allows the runtime to translate incoming pen data into relative coordinates. If a pen is dragged between tiles, the toolkit sees the discontinuity, but the event handlers only see a continuous stream of region-relative coordinates.

Implementation

To obtain the pattern information, we rendered 255 blank patterned pages using our license. We then extracted the encoding into individual documents containing the (`dudlrluudl...`).

The rendering software then combines the PostScript code into larger pages that contain tiled blocks of pattern.

Joel Brandt wrote an early version of this in python. This approach reads in the **.pattern* files, compiles the tiled pattern, and writes out a **.ps* file along with the required Anoto headers, font, and footers. The original script can be found at: http://papertoolkit.googlecode.com/svn/trunk/scripts/tiledPageGenerator. PaperToolkit includes a Java implementation of this approach: it reads in a template file (containing header, font, and footer), drops in the PostScript of the document (*e.g.*, a map image), and overlays the document with the tiled pattern.

C.4 Alternative Technologies

Researchers may want to look into the EPOS pen [EPOS 2007], which does not require a dot-patterned paper, but it gives up the ability to revisit pages in a notebook. Alternatively, the LiveScribe corporation is making their SDK available in early 2008. LiveScribe's smartpen licenses the Anoto technology. However, the company plans to provide Java-based tools that integrate with Eclipse, this researcher's preferred development environment.

Bibliography

1 Abowd, G. D. and E. D. Mynatt. Charting Past, Present, and Future Research in Ubiquitous Computing. *Transactions on Computer-Human Interaction* **7**(1): ACM Press. pp. 29–58, 2000.

2 Amazon, *Kindle Wireless Reading Device*, 2007. http://www.amazon.com/Kindle-Amazons-Wireless-Reading-Device/dp/B000FI73MA

3 Anoto AB, *Anoto Technology*, 2007. http://www.anoto.com

4 Appert, C. and M. Beaudouin-Lafon. SwingStates: Adding State Machines to the Swing Toolkit. *UIST: ACM Symposium on User Interface Software and Technology*. pp. 319–22, 2006.

5 Arnstein, L.*, et al.* Labscape: A Smart Environment for the Cell Biology Laboratory. *IEEE Pervasive Computing* **1**(3): IEEE Educational Activities Department. pp. 13–21, 2002.

6 Avrahami, D., S. E. Hudson, T. P. Moran, and B. D. Williams. Guided gesture support in the paper PDA. *UIST: ACM Symposium on User Interface Software and Technology*. pp. 167–68, 2001.

7 Back, M., J. Cohen, R. Gold, S. Harrison, and S. Minneman. Listen Reader: an Electronically Augmented Paper-based Book. *CHI: ACM Conference on Human Factors in Computing Systems*. pp. 23–29, 2001.

8 Ballagas, R., M. Ringel, M. Stone, and J. Borchers. iStuff: a physical user interface toolkit for ubiquitous computing environments. *CHI: ACM Conference on Human Factors in Computing Systems*. pp. 537–44, 2003.

9 Bastéa-Forte, M., R. B. Yeh, and S. R. Klemmer. Pointer: Multiple Collocated Display Inputs Suggests New Models for Program Design and Debugging. *UIST Extended Abstracts (Posters)*, 2007.

10 Beckmann, C. and A. Dey. SiteView: Tangibly Programming Active Environments with Predictive Visualization. *Ubicomp Extended Abstracts (Posters)*. pp. 167–68, 2003.

11 Bederson, B. B., J. Grosjean, and J. Meyer. Toolkit Design for Interactive Structured Graphics. *IEEE Transactions on Software Engineering* **30**(8). pp. 1–12, 2004.

12 Bellotti, V. and Y. Rogers. From Web Press to Web Pressure: Multimedia Representations and Multimedia Publishing. *CHI: ACM Conference on Human Factors in Computing Systems*. pp. 279–86, 1997.

13 Bernstein, M., A. Robinson-Mosher, R. B. Yeh, and S. R. Klemmer. Diamond's Edge: From Notebook to Table and Back Again. *Ubicomp Extended Abstracts (Posters)*, 2006.

14 Beyer, H. and K. Holtzblatt, *Contextual Design: Defining Customer-Centered Systems*: Morgan Kaufmann. 500 pp. 1997.

15 Brown, J. S. and P. Duguid, *The Social Life of Information*: Harvard Business School Press. 352 pp. 2002.

16 Butler, D. A New Leaf. *Nature* **436**: Nature Publishing Group. pp. 20–21, 2005.

17 Buxton, B., *Sketching User Experiences: Getting the Design Right and the Right Design*: Morgan Kaufmann. 448 pp. 2007.

18 Chen, N., F. Guimbretière, M. Dixon, C. Lewis, and M. Agrawala, Navigation Techniques for Dual-Display E-Book Readers. 2007, University of Maryland.

19 Chi, E., J. Riedl, P. Barry, and J. Konstan. Principles for Information Visualization Spreadsheets. *Computer Graphics and Applications* **18**(4): IEEE. pp. 30–38, 1998.

20 Clarke, S. Measuring API Usability. *Dr. Dobb's Journal: Windows/.NET Supplement*. pp. S6–S9, 2004.

21 Clarke, S., *stevencl's WebLog*, 2005. http://blogs.msdn.com/stevencl/

22 Cohen, P. R. and D. R. McGee. Tangible Multimodal Interfaces for Safety Critical Applications, *Communications of the ACM*, vol. 47(1): pp. 41–46, 2004.

23 Conroy, K., D. Levin, and F. Guimbretière. ProofRite: A Paper-Augmented Word Processor. *UIST Extended Abstracts (Demos)*, 2004.

24 Cooper, A., *The Inmates Are Running the Asylum: Why High Tech Products Drive Us Crazy and How to Restore the Sanity*: Sams. 288 pp. 1999.

25 Cooper, A., *The Origin of Personas*, 2007. http://www.cooper.com/insights/journal_of_design/articles/the_origin_of_personas_1.html

26 Crossbow Technology, *MICAz*, 2005. San Jose, CA. http://www.xbow.com/Products/productsdetails.aspx?sid=101

27 Culler, D. E. and H. Mulder. Smart Sensors to Network the World. *Scientific American* **290**(6). pp. 84–91, 2004.

28 CyberTracker, *CyberTracker*, 2007. http://www.cybertracker.co.za

29 Dann, W., S. Cooper, and R. Pausch, *Learning To Program With Alice*: Prentice Hall. 375 pp. 2005.

30 Dietz, P. and D. Leigh. DiamondTouch: A Multi-User Touch Technology. *UIST: ACM Symposium on User Interface Software and Technology*. pp. 219–26, 2001.

31 Dymetman, M. and M. Copperman. Intelligent Paper. *EP'98: International Conference on Electronic Publishing*: Springer-Verlag GmbH. pp. 392–406, 1998.

32 E Ink, *Electronic Paper Displays*, 2007. http://www.eink.com

33 Edwards, W. K.*, et al.* Using Speakeasy for Ad Hoc Peer-to-Peer Collaboration. *CSCW: ACM Conference on Computer Supported Cooperative Work*. pp. 256–65, 2002.

34 Elliott, A. and M. A. Hearst. A Comparison of the Affordances of a Digital Desk and Tablet for Architectural Image Tasks. *International Journal of Human-Computer Studies* **56**(2). pp. 173–97, 2002.

35 Ellis, B., J. Stylos, and B. Myers. The Factory Pattern in API Design: A Usability Evaluation. *ICSE: International Conference on Software Engineering*: IEEE. pp. 302–12, 2007.

36 Embedded Data Systems, LLC., *iButton*, 2005. http://embeddeddatasystems.com

37 EPOS, *EPOS Digital Pen*, 2007. http://www.epos-ps.com

38 Everitt, K. M., S. R. Klemmer, R. Lee, and J. A. Landay. Two Worlds Apart: Bridging the Gap Between Physical and Virtual Media for Distributed Design Collaboration. *CHI: ACM Conference on Human Factors in Computing Systems*. pp. 553–60, 2003.

39 Foley, J. D., A. v. Dam, S. K. Feiner, and J. F. Hughes, *Computer Graphics: Principles and Practice*. 2 ed: Addison-Wesley Professional. 1200 pp. 1995.

40 Foley, J. D. and V. L. Wallace. The Art of Natural Graphic Man-Machine Conversation. *IEEE* **62**(4). pp. 462–71, 1974.

41 Forsythe, D. E. It's Just a Matter of Common Sense: Ethnography as Invisible Work. *CSCW: ACM Conference on Computer-Supported Cooperative Work* **8**(1). pp. 127–45, 1999.

42 Gamma, E., R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-oriented Software*. Boston, MA: Addison-Wesley. 416 pp. 1995.

43 Gigapxl Project, *Gigapxl Project*, 2006. http://www.gigapxl.org

44 Green, T. R. G. and M. Petre. Usability Analysis of Visual Programming Environments: A Cognitive Dimensions Framework. *Journal of Visual Languages and Computing* **7**(2). pp. 131–74, 1996.

45 Guimbretière, F. Paper Augmented Digital Documents. *UIST: ACM Symposium on User Interface Software and Technology*. pp. 51–60, 2003.

46 Hall, E. T., *The Hidden Dimension*: Anchor. 240 pp. 1990.

47 Haller, M., P. Brandl, D. Leithinger, J. Leitner, T. Siefried, and M. Billinghurst. Shared Design Space: Sketching Ideas Using Digital Pens and a Large Augmented Tabletop Setup. *ICAT: International Conference on Artifical Reality and Telexistence*. pp. 185–96, 2006.

48 Hartmann, B.*, et al.* Reflective Physical Prototyping through Integrated Design, Test, and Analysis. *UIST: ACM Symposium on User Interface Software and Technology*. pp. 299–308, 2006.

49 Heath, C. and P. Luff, *Technology in Action (Learning in Doing: Social, Cognitive & Computational Perspectives)*: Cambridge University Press. 286 pp. 2000.

50 Heiner, J. M., S. E. Hudson, and K. Tanaka. Linking and Messaging from Real Paper in the Paper PDA. *UIST: ACM Symposium on User Interface Software and Technology*. pp. 179–86, 1999.

51 Holmquist, L. E., J. Redström, and P. Ljungstrand. Token-Based Access to Digital Information. *Handheld and Ubiquitous Computing*: Springer-Verlag, 1999.

52 Hong, J. I. and J. A. Landay. SATIN: a Toolkit for Informal Ink-based Applications. *UIST: ACM Symposium on User Interface Software and Technology*. pp. 63–72, 2000.

53 Horn, R. E., Infrastructure for Navigating Interdisciplinary Debates: Critical Decisions for Representing Argumentation, in *Visualizing Argumentation: Software Tools for Collaborative and Educational Sense-Making*, P. Kirschner, S.B. Shum, and C. Carr, Editors. Springer-Verlag. pp. 165–84, 2003.

54 Hourcade, J. P. and B. B. Bederson, *Architecture and Implementation of a Java Package for Multiple Input Devices (MID)*. Technical Report, University of Maryland 1999. http://www.cs.umd.edu/hcil/mid

55 HSQLDB, *HSQL Database Engine*, 2005. http://www.hsqldb.org

56 Hudson, S. E., J. Mankoff, and I. Smith. Extensible Input Handling in the subArctic Toolkit. *CHI: ACM Conference on Human Factors in Computing Systems*. pp. 381–90, 2005.

57 Hutchins, E. L., J. D. Hollan, and D. A. Norman. Direct Manipulation Interfaces. *Human-Computer Interaction* **1**(4). pp. 311–38, 1985.

58 IBM Pen Technologies, *CrossPad and TransNote*, 2007. http://researchweb.watson.ibm.com/electricInk

59 Imhof, E., *Cartographic Relief Presentation*: Walter De Gruyter, Inc. 388 pp. 1982.

60 Iqbal, S. and E. Horvitz. Disruption and Recovery of Computing Tasks: Field Study, Analysis, and Directions. *CHI: ACM Conference on Human Factors in Computing Systems*: ACM Press. pp. 677–86, 2007.

61 Ishii, H., M. Kobayashi, and K. Arita. Iterative Design of Seamless Collaboration Media, *Communications of the ACM*, vol. 37(8): pp. 83–97, 1994.

62 Jiang, H., R. B. Yeh, T. Winograd, and Y. Shi. DigiPost: Writing on Post-its with Digital Pens to Support Collaborative Editing Tasks on Tabletop Displays. *UIST Extended Abstracts (Posters)*, 2007.

63 Johnson, W., H. Jellinek, L. K. Jr., R. Rao, and S. Card. Bridging the Paper and Electronic Worlds: The Paper User Interface. *CHI: ACM Conference on Human Factors in Computing Systems*. pp. 507–12, 1993.

64 Kam, M.*, et al.* Livenotes: A System for Cooperative and Augmented Note-Taking in Lectures. *CHI: ACM Conference on Human Factors in Computing Systems*. pp. 531–40, 2005.

65 Kandel, S., A. Paepcke, M. Theobald, and H. Garcia-Molina, PhotoSpread: A Spreadsheet for Managing Photos. 2007, Stanford University Computer Science Department.

66 Kelleher, C. and R. Pausch. Lowering the Barriers to Programming: A Survey of Programming Environments and Languages for Novice Programmers. *CSUR: ACM Computing Surveys* **37**(2). pp. 83–137, 2005.

67 Kerouac, J., *On the Road*. Reprint ed: Penguin Books. 307 pp. 1991.

68 Kim, J., S. M. Seitz, and M. Agrawala. Video-based Document Tracking: Unifying Your Physical and Digital Desktops. *UIST: ACM Symposium on User Interface Software and Technology*. pp. 99–107, 2004.

69 Kim, M., L. Bergman, T. Lau, and D. Notkin. An Ethnographic Study of Copy and Paste Programming Practices in OOPL. *International Symposium on Empirical Software Engineering*. pp. 83–92, 2004.

70 Kirsh, D. The Intelligent Use of Space. *Artificial Intelligence* **73**(1-2): Elsevier. pp. 31–68, 1995.

71 Klemmer, S. R., *Tangible User Interface Input: Tools and Techniques*, Unpublished PhD, University of California, Computer Science, Berkeley, CA, 2004. http://hci.stanford.edu/srk/KlemmerDissertation.pdf

72 Klemmer, S. R., J. Graham, G. J. Wolff, and J. A. Landay. Books with Voices: Paper Transcripts as a Tangible Interface to Oral Histories. *CHI: ACM Conference on Human Factors in Computing Systems*. pp. 89–96, 2003.

73 Klemmer, S. R., B. Hartmann, and L. Takayama. How Bodies Matter: Five Themes for Interaction Design. *DIS: ACM Conference on Designing Interactive Systems*, 2006.

74 Klemmer, S. R., J. Li, J. Lin, and J. A. Landay. Papier-Mâché: Toolkit Support for Tangible Input. *CHI: ACM Conference on Human Factors in Computing Systems*. pp. 399–406, 2004.

75 Klemmer, S. R., M. W. Newman, R. Farrell, M. Bilezikjian, and J. A. Landay. The Designers' Outpost: A Tangible Interface for Collaborative Web Site Design. *UIST: ACM Symposium on User Interface Software and Technology*. pp. 1–10, 2001.

76 Ko, A. J. and B. A. Myers. Designing the Whyline: A Debugging Interface for Asking Questions About Program Behavior. *CHI: ACM Conference on Human Factors in Computing Systems*. pp. 151–58, 2004.

77 Ko, A. J. and B. A. Myers. Development and Evaluation of a Model of Programming Errors. *IEEE Symposium on Human Centric Computing Languages and Environments*. pp. 7–14, 2003.

78 Krasner, G. E. and S. T. Pope. A cookbook for using the model-view controller user interface paradigm in Smalltalk-80. *Object Oriented Programming* **1**(3). pp. 26–49, 1988.

79 Lamming, M. and M. Flynn. Forget-me-not: Intimate Computing in Support of Human Memory. *FRIEND21: International Symposium on Next Generation Human Interface*, 1994.

80 Landay, J. and B. A. Myers. Interactive sketching for the early stages of user interface design. *CHI: ACM Conference on Human Factors in Computing Systems*. pp. 43–50, 1995.

81 Landay, J. A., *Interactive Sketching for the Early Stages of User Interface Design*, Unpublished PhD, Carnegie Mellon University, Computer Science, Pittsburgh, PA, 1996.

82 Lee, B. A., *Adaptive Interaction Techniques for Sharing and Reusing Design Resources*, Unpublished PhD, Stanford University, Computer Science, Stanford, CA, 2007.

83 Lee, J. C., P. H. Dietz, D. Leigh, W. S. Yerazunis, and S. E. Hudson. Haptic Pen: A Tactile Feedback Stylus for Touch Screens. *UIST: ACM Symposium on User Interface Software and Technology*. pp. 291–94, 2004.

84 Levis, P. and D. Culler. Maté: a Tiny Virtual Machine for Sensor Networks. *ACM SIGOPS* **36**(5). pp. 85–95, 2002.

85 Levoy, M. Spreadsheets for Images. *SIGGRAPH: ACM Conference on Computer Graphics and Interactive Techniques*. pp. 139–46, 1994.

86 Lewis, B., *Omniscient Debugging*, 2007. http://www.lambdacs.com/debugger/

87 Liao, C., F. Guimbretière, and K. Hinckley. PapierCraft: A Command System for Interactive Paper. *UIST: ACM Symposium on User Interface Software and Technology*. pp. 241–44, 2005.

88 Liao, C., F. Guimbretière, and C. E. Loeckenhoff. Pen-top Feedback for Paper-based Interfaces. *UIST: ACM Symposium on User Interface Software and Technology*. pp. 201–10, 2006.

89 Lieberman, H. The Debugging Scandal and What to Do About It, *Communications of the ACM*, vol. 40(4): pp. 26–78, 1997.

90 Lieberman, H. Steps Toward Better Debugging Tools for LISP. *ACM Symposium on LISP and Functional Programming*. pp. 247–55, 1984.

91 Lieberman, H. and C. Fry. Bridging the Gulf between Code and Behavior in Programming. *CHI: ACM Conference on Human Factors in Computing Systems*. pp. 480–86, 1995.

92 Livescribe Inc., *Livescribe*, 2007. http://www.livescribe.com

93 Ljungstrand, P., J. Redström, and L. E. Holmquist. WebStickers: Using Physical Tokens to Access, Manage, and Share Bookmarks to the Web. *DARE: Designing Augmented Reality Environments*: ACM Press. pp. 23–31, 2000.

94 Lowagie, B. and P. Soares, *iText Java-PDF Library*, 2007. http://www.lowagie.com/iText

95 Mackay, W. E. Is Paper Safer? The Role of Paper Flight Strips in Air Traffic Control. *ACM Transactions on Computer-Human Interaction* **6**(4). pp. 311–40, 1999.

96 Mackay, W. E., A.-L. Fayard, L. Frobert, and L. Médini. Reinventing the Familiar: Exploring an Augmented Reality Design Space for Air Traffic Control. *CHI: ACM Conference on Human Factors in Computing Systems*. pp. 558–65, 1998.

97 Mackay, W. E., *et al.* Ariel: Augmenting Paper Engineering Drawings. *CHI: ACM Conference on Human Factors in Computing Systems*. pp. 421–22, 1995.

98 Mackay, W. E., G. Pothier, C. Letondal, K. Bøegh, and H. E. Sørensen. The Missing Link: Augmenting Biology Laboratory Notebooks. *UIST: ACM Symposium on User Interface Software and Technology*. pp. 41–50, 2002.

99 Maldonado, H., B. A. Lee, S. R. Klemmer, and R. D. Pea. Patterns of Collaboration in Design Courses: Team Dynamics Affect Technology Appropriation, Artifact Creation, and Course Performance. *CSCL: ACM Conference on Computer-Supported Collaborative Learning*, 2007.

100 Mankoff, J., *An Architecture and Interaction Techniques for Handling Ambiguity in Recognition-based Input*, Unpublished PhD, Georgia Institute of Technology, Computer Science, 2001. http://www.cs.cmu.edu/~assist/publications/thesis.pdf

101 Maynes-Aminzade, D., T. Winograd, and T. Igarashi. Eyepatch: Prototyping Camera-based Interaction through Examples. *UIST: ACM Symposium on User Interface Software and Technology*. pp. 33–42, 2007.

102 Microsoft, *Tablet PC SDK*, 2007. http://msdn2.microsoft.com/en-us/library/ms840463.aspx

103 Miller, R. B. Response Time in Man-Computer Conversational Transactions. *AFIPS Fall Joint Computer Conference* **33**. pp. 267-77, 1968.

104 Minneman, S., *et al.* A Confederation of Tools for Capturing and Accessing Collaborative Activity. In Proceedings of *ACM International Conference on Multimedia*: ACM Press. pp. 523–34, 1995.

105 Moran, T. P., *et al.* "I'll get that off the audio": a Case Study of Salvaging Multimedia Records. *CHI: ACM Conference on Human Factors in Computing Systems*. pp. 202–09, 1997.

106 Mutton, P., *Java EPS Graphics2D*, 2007. http://www.jibble.org/epsgraphics

107 MVZ, *The Grinnell Method*, 2005. University of California: Berkeley. http://mvz.berkeley.edu/Grinnell_Method.html

108 Myers, B. A New Model for Handling Input. *ACM Transactions on Information Systems* **8**(3): ACM Press. pp. 289–320, 1990.

109 Myers, B., S. E. Hudson, and R. Pausch. Past, Present, and Future of User Interface Software Tools. *ACM Transactions on Computer-Human Interaction* **7**(1). pp. 3–28, 2000.

110 Myers, B. A. and M. B. Rosson. Survey on User Interface Programming. *CHI: ACM Conference on Human Factors in Computing Systems*. pp. 195–202, 1992.

111 Nelson, L., S. Ichimura, E. R. Pedersen, and L. Adams. Palette: A Paper Interface for Giving Presentations. *CHI: ACM Conference on Human Factors in Computing Systems*. pp. 354–61, 1999.

112 Newman, M. W., J. Lin, J. I. Hong, and J. A. Landay. DENIM: An Informal Web Site Design Tool Inspired by Observations of Practice. *Human-Computer Interaction* **18**(3). pp. 259–324, 2003.

113 Nielsen, J., *Ten Usability Heuristics*, 2007. http://www.useit.com/papers/heuristic/heuristic_list.html

114 Norrie, M. C., B. Signer, and N. Weibel. Print-n-Link: Weaving the Paper Web. *DocEng: ACM Symposium on Document Engineering*, 2006.

115 OQO, *model 01+*, 2006. http://www.oqo.com

116 Pane, J., *A Programming System for Children that is Designed for Usability.*, Unpublished PhD, Carnegie Mellon University, Computer Science, Pittsburgh, PA, 2002. www.cs.cmu.edu/~pane/thesis

117 Phelps, T. A. and R. Wilensky. The Multivalent Browser: a Platform for New Ideas. *DocEng: ACM Symposium on Document Engineering*. pp. 58–67, 2001.

118 Rettig, M. Prototyping for tiny fingers, *Communications of the ACM*, vol. 37(4): pp. 21-27, 1994.

119 Romero, P., B. du Boulay, R. Cox, and R. Lutz. Java Debugging Strategies in Multi-Representational Environments. *Psychology of Programming Interest Group*. pp. 421–34, 2003.

120 Rosson, M. B. and J. M. Carroll. The Reuse of Uses in Smalltalk Programming. *ACM Transactions on Computer-Human Interaction* **3**(3). pp. 219–53, 1996.

121 Saund, E., D. Fleet, D. Larner, and J. Mahoney. Perceptually-Supported Image Editing of Text and Graphics. *UIST: ACM Symposium on User Interface Software and Technology*. pp. 183–92, 2003.

122 Schilit, B. N. and U. Sengupta. Device Ensembles. *Computer* **37**(12). pp. 56–64, 2004.

123 Schleimer, S., D. S. Wilkerson, and A. Aiken. Winnowing: Local Algorithms for Document Fingerprinting. *SIGMOD: ACM International Conference on Management of Data*. pp. 76–85, 2003.

124 Sellen, A. J. and R. H. R. Harper, *The Myth of the Paperless Office*. 1st ed: MIT Press. 242 pp. 2001.

125 Shneiderman, B., G. Fischer, M. Czerwinski, B. Myers, and M. Resnick, *Creativity Support Tools*. Washington, DC: National Science Foundation. 83 pp. 2005.

126 Signer, B., *Fundamental Concepts for Interactive Paper and Cross-Media Information Spaces*, Unpublished PhD, ETH Zurich, Computer Science, Zurich, 2006. http://people.inf.ethz.ch/signerb/publications/signer16218.pdf

127 Signer, B. and M. C. Norrie. PaperPoint: a Paper-based Presentation and Interactive Paper Prototyping Tool. *International Conference on Tangible and Embedded Interaction*. pp. 57-64, 2007.

128 Simard, P. Y., D. Steinkraus, and M. Agrawala. Ink Normalization and Beautification. *ICDAR: International Conference on Document Analysis and Recognition*. pp. 1182–87, 2005.

129 Smart Technologies, *SmartBoard*, 2007. http://smarttech.com/

130 Stasko, J., J. Domingue, M. H. Brown, and B. A. Price, *Software Visualization: Programming as a Multimedia Experience*: MIT Press. 550 pp. 1998.

131 Steven, J., P. Chandra, B. Fleck, and A. Podgurski. jRapture: A Capture/Replay Tool for Observation-based Testing. *SIGSOFT: ACM Special Interest Group onSoftware Engineering*. pp. 158–67, 2000.

132 Stifelman, L., B. Arons, and C. Schmandt. The Audio Notebook: Paper and Pen Interaction with Structured Speech. *CHI: ACM Conference on Human Factors in Computing Systems*. pp. 182–89, 2001.

133 Sun Microsystems, *Robot: Java Platform SE6*, 2007. http://java.sun.com/javase/6/docs/api/java/awt/Robot.html

134 Sun Microsystems, *Swing*, 2007. http://java.sun.com/javase/6/docs

135 Tang, J. C. and S. L. Minneman. VideoDraw: A Video Interface for Collaborative Drawing. *ACM Transactions on Information Systems* **9**(2). pp. 170–84, 1991.

136 Thorn, T. Programming Languages for Mobile Code, *ACM Computing Surveys (CSUR)*, vol. 29(3): pp. 213–39, 1997.

137 TIER, *Technology and Infrastructure for Emerging Regions, UC Berkeley*, 2007. http://tier.cs.berkeley.edu

138 Truong, K. N., G. D. Abowd, and J. A. Brotherton. Personalizing the Capture of Public Experiences. *UIST: ACM Symposium on User Interface Software and Technology*. pp. 121–30, 1999.

139 Tufte, E. R., *Envisioning Information*: Graphics Press. 126 pp. 1990.

140 Tufte, E. R., *The Visual Display of Quantitative Information*. 2 ed: Graphics Press. 197 pp. 2001.

141 Weiser, M. The Computer for the 21st Century. *Scientific American*. pp. 94–104, 1991.

142 Wellner, P. Interacting With Paper on the DigitalDesk, *Communications of the ACM*, vol. 36(7): pp. 87–96, 1993.

143 White, S. M., S. Feiner, and J. Kopylec. Virtual Vouchers: Prototyping a Mobile Augmented Reality User Interface for Botanical Species Identification. *3DUI: IEEE Symposium on 3D User Interfaces*. pp. 119–26, 2006.

144 White, S. M., D. Marino, and S. Feiner. Designing a Mobile User Interface for Automated Species Identification. *CHI: ACM Conference on Human Factors in Computing Systems*. pp. 291–94, 2007.

145 Whittaker, S., P. Hyland, and M. Wiley. FILOCHAT: Handwritten Notes Provide Access to Recorded Conversations. *CHI: ACM Conference on Human Factors in Computing Systems*. pp. 271–77, 1994.

146 Wilson, A. D. PlayAnywhere: A Compact Interactive Tabletop Projection-Vision System. *UIST: ACM Symposium on User Interface Software and Technology*. pp. 83–92, 2005.

147 Wobbrock, J. O., A. D. Wilson, and Y. Li. Gestures without Libraries, Toolkits or Training: A $1 Recognizer for User Interface Prototypes. *UIST: ACM Symposium on User Interface Software and Technology*, 2007.

148 Woo, M., J. Neider, T. Davis, and D. Shreiner, *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 1.2*. 3 ed: Addison-Wesley Professional. 730 pp. 1999.

149 Xerox, *Erasable Paper*, 2007. http://www.xerox.com/innovation/exp_paper.shtml

150 Yeh, R. B., J. Brandt, S. R. Klemmer, J. Boli, E. Su, and A. Paepcke, Interactive Gigapixel Prints: Large Paper Interfaces for Visual Context, Mobility, and Collaboration. 2006, Stanford University Computer Science Department.

151 Yeh, R. B., *et al.* ButterflyNet: A Mobile Capture and Access System for Field Biology Research. *CHI: ACM Conference on Human Factors in Computing Systems*. pp. 571–80, 2006.