d.note: Revising User Interfaces Through Change Tracking, Annotations, and Alternatives

Björn Hartmann[1], Sean Follmer[2], Antonio Ricciardi[2], Timothy Cardenas[2], Scott R. Klemmer[2]

1–Computer Science Division
University of California, Berkeley, CA 94720
bjoern@cs.berkeley.edu

2–Stanford University HCI Group
Computer Science Dept, Stanford, CA 94305
srk@cs.stanford.edu

ABSTRACT

Interaction designers typically revise user interface prototypes by adding unstructured notes to storyboards and screen printouts. How might computational tools increase the efficacy of UI revision? This paper introduces d.note, a revision tool for user interfaces expressed as control flow diagrams. d.note introduces a command set for modifying and annotating both appearance and behavior of user interfaces; it also defines execution semantics so proposed changes can be tested immediately. The paper reports two studies that compare production and interpretation of revisions in d.note to freeform sketching on static images (the status quo). The *revision production* study showed that testing of ideas during the revision process led to more concrete revisions, but that the tool also affected the type and number of suggested changes. The *revision interpretation* study showed that d.note revisions required fewer clarifications, and that additional techniques for expressing revision intent could be beneficial.

Author Keywords

Interaction design tools, prototyping, revision, annotation.

ACM Classification Keywords

H.5.2. [Information Interfaces]: User Interfaces – *prototyping.* D.2.2 [Software Engineering]: Design Tools and Techniques – *state diagrams; user interfaces.*

General Terms

Design, Human Factors.

INTRODUCTION

Interaction design teams oscillate between individual work and team reviews and discussions. Team reviews of user interface prototypes provide valuable critique and suggest future design directions [25, pp. 374-5]. However, proposed changes can rarely be realized immediately: the proposer may lack implementation knowledge, the changes may be too complex, or the ideas are not sufficiently resolved.

In many areas of design, annotations layered on top of existing drawings and images, or "sketches on top of sketches" [3], are the preferred way of capturing proposed changes. They are rapid to construct, they enable designers to handle different levels of abstraction and ambiguity simultaneously [4], and they serve as common ground for members with different expertise and toolsets [27]. Individual designers later incorporate the proposed changes into the next prototype. This annotate-review-incorporate cycle is similar to revising and commenting on drafts of written documents [26]. While word processors offer specialized revision tools for these tasks, such tools do not exist for the domain of interaction design.

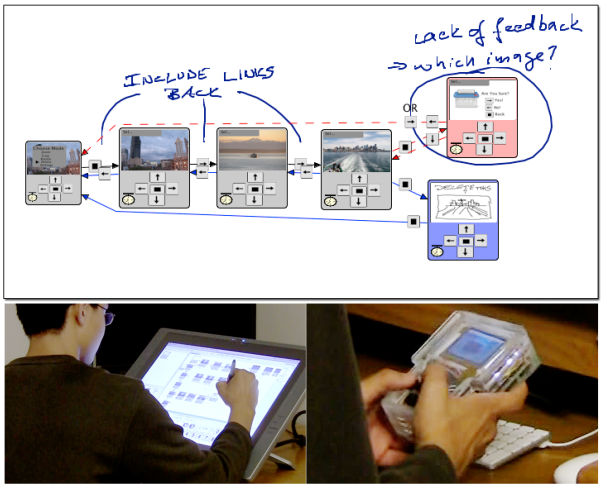

Figure 1. d.note enables interaction designers to revise and test functional prototypes of information appliances.

This paper demonstrates how three primary text revision techniques apply to interaction design: *commenting*, *tracking changes*, and *visualizing those changes*. It also introduces revision tools unique to interaction design: *immediate testing* of revisions and *proposing alternatives*. Because interaction design specifies both appearance and behavior, revisions should be testable immediately when possible. Because enumeration and selection of alternatives is fundamental to design [3,10,15,32], revisions should also be expressible as alternatives to existing functionality.

The proposed revision techniques are embodied in d.note (Figure 1), a tool for interaction designs created with d.tools [11]. The d.note notation supports modification, commenting, and proposal of alternatives for both appearance and

CHI 2010, April 10–15, 2010, Atlanta, Georgia, USA.

behavior of information appliance prototypes. Concrete modifications to behavior can be tested while a prototype is running. Such modifications can exist alongside more abstract, high-level comments and annotations.

This paper also characterizes the benefits and tradeoffs of digital revision tools such as d.note through two user studies. We show that the choice of revision tool affects both what kind of revisions are expressed, as well as the ability of others to interpret those revisions later on. Participants who used d.note to express revisions focused more on the interaction architecture of the design, marked more elements for deletion, and wrote fewer text comments than participants without d.note. Participants that interpreted d.note diagrams asked for fewer clarifications than participants that interpreted freeform annotations, but had more trouble discerning the reviser's intent.

In the remainder of the paper, we discuss related work, survey today's UI revision practices, and describe revision principles from related domains. We then introduce d.note and its implementation. We present results from two studies of revision expression and interpretation, and conclude with a look at the larger design space of revision tools.

RELATED WORK

d.note draws on existing work in four areas: annotation and revision tools, difference visualization techniques, design histories, and informal design tools.

Annotation Tools

Change tracking and commenting tools are pervasive in word processors. Such functions enable asynchronous collaboration, where different members may have different functions, such as author, commenter, and reader [26]. d.note applies change tracking and commenting to the domain of interaction design. It takes inspiration from tools that capture sketched comments and interpret these as commands to change an underlying software model. In Paper Augmented Digital Documents [7], annotations are written on printed documents with digital pens; pen strokes change the corresponding digital document. In ModelCraft [30], users draw on physical 3D models created from CAD files, to express extrusions, cuts, and notes. These annotations then change the underlying CAD model.

Capturing Design History

Design histories capture and visualize the sequence of actions that a designer or a design team took to get to the current state of their work. The visual explanations tend to focus on step-by-step transformations, *e.g.*, for web site diagrams [18], illustrations [19,31], or information visualizations [13]. Revision tools such as d.note focus on a larger set of changes to a base document version, where the order of changes is not of primary concern. Design histories offer timeline-based browsing of changes in a view external to the design document; d.note offers a comprehensive view of a set of changes *in situ*, in the design document itself.

Comparing Alternatives

Design histories and d.note track changes while they are made at design time. Another approach is to compute and visualize differences of a set of documents after they were edited. The well-known diff algorithm shows differences between two text files [14]. Offline comparison algorithms also exist for pairs of UML diagrams [6] and for multiple versions of slide presentations [5]. The d.note visual language is most closely related to diagram differencing techniques introduced for CASE diagrams [24] and for statecharts [9] in the Kiel Integrated Environment for Layout [28]. Such research contributes algorithms to identify and visualize changes. d.note contributes interaction techniques to create, test, and share such changes.

Informal Design Tools

Prior work has demonstrated techniques for designers to sketch GUIs [20], web sites [22], and multimedia content [2]. TEAM STORM [8] enabled collaborative sketching for multiple co-located participants. Topiary [21] exported sketched interfaces to mobile devices and allowed sketched comments as a secondary notation in the editor. d.note builds on these sketching techniques and contributes a sketch-based visual language for revising interfaces.

SUEDE [17] and d.tools [11] introduced the concept of integrating design, test, and analysis in a single authoring environment. We are inspired by the approach to explicitly add support for the larger context of design activity into a prototyping tool. The two prior systems focused on user testing; d.note focuses on design revision.

SURVEY OF CURRENT TOOLS AND PRACTICES

In this section we review current UI revision practices, and discuss related tools from other domains of creative work.

UI Revision Practices Today

We contacted practitioners to find out how interaction design teams currently communicate revisions of user interface designs. Ten designers responded through a professional mailing list; seven of them shared detailed reports. There was little consistency between the reported practices — techniques included printing out screens and sketching on them; assembling printouts on a wall; capturing digital screenshots and posting them to a wiki; and using version control systems and bug tracking databases. We suggest that the high variance in approaches is due to a lack of specific tool support for UI designers.

We also noted a pronounced divide between physical and digital processes [16]. One designer worked exclusively on printouts; four reported a mixture between working on paper and using digital tools; and two relied exclusively on digital tools. To make sense of this divide, it useful to distinguish between two functions: recording changes that should be applied to the *current* design; and keeping track of *multiple* versions over time. For expressing changes to a current design, five of the surveyed designers preferred sketching on static images because of its speed and flexibility. In contrast, designers preferred digital tools to capture history over time and to share changes with others. We hypothesize that designers will benefit from tools that bridge the gap between capturing changes and tracking history.

Revision Practices in Other Domains

Interaction designers are concerned with both look and feel of applications [25]. Absent a complete solution for both aspects, we can draw on important insights from revising textual documents, source code, movies, and games.

Text Documents

The fundamental actions in written document revision are history-preserving modification and commenting. Each operation has two components: visual syntax and semantics. For example, a common interlinear syntax to express deletion is striking through the deleted text (see Figure 2); the semantics are to remove the stricken text from the next version of the document, should the revision be accepted. Original and modification are visible simultaneously, to communicate the nature of a change. Furthermore, edits are visually distinguished from the base version. When editing documents collaboratively, different social roles of co-author, commenter, and reader exist [26]. Offering ways to modify the text as well as adding meta-content that suggests further modification serves these different roles well.

Source Code Documents

Source-code revision tools, such as visual difference editors, enable users to compare two versions of source files side-by-side [12] (Figure 3). In contrast to document revision tools, changes are generally not tracked incrementally, but computed and visualized after the fact. Comments in source code differ from comments in text documents as they are part of the source document itself. Meta-commenting (comments about changes) is generally only available at the level of an entire set of changes.

Visual Media

WYSIWYG document editors do not distinguish between source and final document; authors revise a single, shared representation. For program source code, there is no way to comment directly on the *output* of the program, only the source. In contrast, movie producers and video game developers convey revisions by drawing directly on *output*, *i.e.*, rendered video frames (Figure 4). Because the revisions address changes in appearance, sketching rather than text is the preferred method of expression. Working in the output domain is a compelling approach, but has thus far been limited to static content [3].

Comparing these three existing domains leads us to formulate four design principles. UI revision tools should support the following:

1. History-preserving incremental modification of the source representation
2. Commenting outside the underlying source language
3. Sketching as an input modality for graphical content
4. Revising the *output*, *i.e.*, the resulting user interface screens, not just the source

A VISUAL GRAMMAR FOR REVISING INTERACTIONS

Guided by our assessment of current practice and tools available in other domains, we developed d.note, a revision notation for user interface prototypes. d.note extends d.tools [11], a visual authoring environment that simultaneously

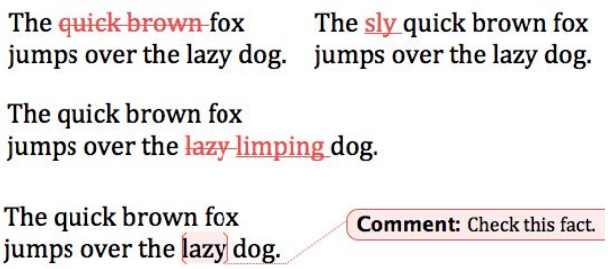

Figure 2. Interlinear revision tracking and comment visualization in word processing.

Figure 3. Source code comparison tools show two versions of a file side-by-side.

Figure 4. Video game designers draw annotations directly on rendered still images (from [4], p. 179).

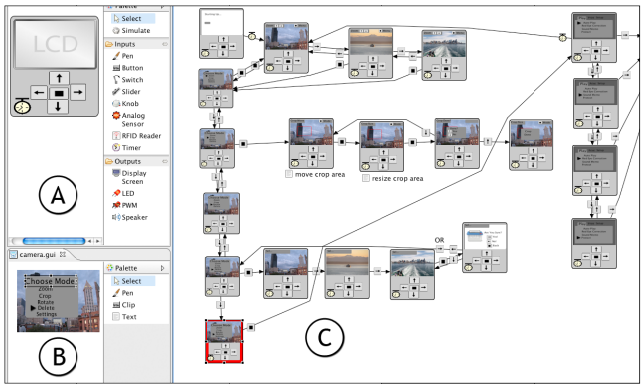

Figure 5. The d.tools environment with device editor (A), graphics editor (B), and storyboard editor (C).

shows both appearance and interaction logic of user interfaces (Figure 5). d.tools offers a visual control flow language for prototyping *information appliances*, physical devices with graphical user interfaces. In d.tools, designers define the hardware (what inputs/outputs exist - Figure 5a), graphical output (Figure 5b), and interaction logic (Figure 5c). Logic diagrams are inspired by the statechart formalism [9]: transitions express control flow ("if button X is pressed, go to state Q"). GUI output is defined uniquely for

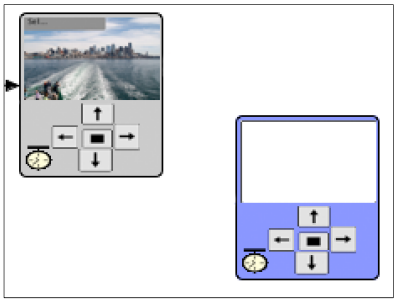

Figure 6. States added during revision are rendered in blue.

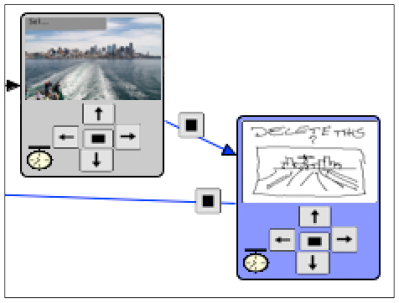

Figure 7. New screen graphics can be sketched in states.

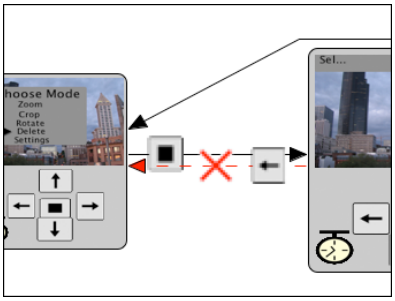

Figure 8. Transition deletions are marked with a red cross and dashed red lines.

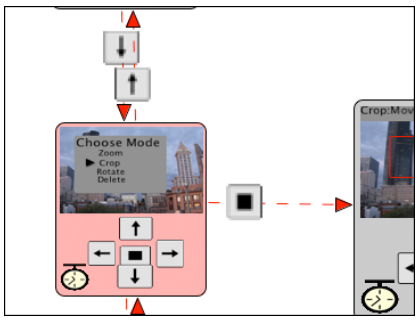

Figure 9. State deletions are rendered in red. Connections are marked as inactive.

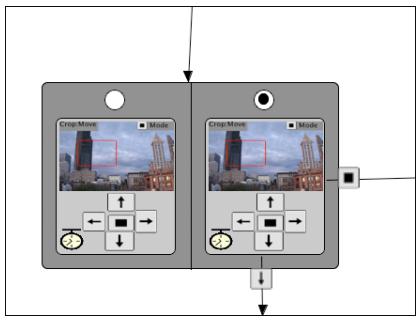

Figure 10. Alternative containers express different options for a state.

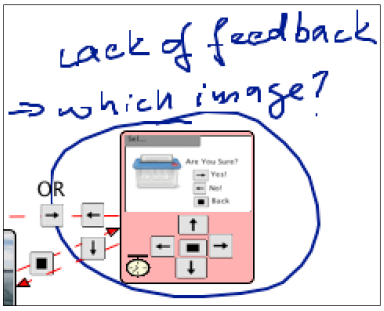

Figure 11. Comments can also be attached to any state.

each state. Additional dynamic behavior (e.g., tying screen position of a graphic to accelerometer input values) can be expressed through a scripting language.

Scenario

The following scenario introduces the benefits d.note provides to interaction design teams. Adam is designing a user interface for a new digital camera with on-camera image editing functions. To get feedback, he drops his latest prototype off in Betty's office. Betty picks up the camera prototype, and tries to crop, pan and color-balance one of the pictures that Adam pre-loaded on the prototype. She opens up the d.tools diagram for the prototype. She notices that the image delete functionality is lacking a confirmation screen – images are deleted right away. To highlight this omission, Betty creates a new state (Figure 6) and sketches a rudimentary confirmation dialog, which she connects to the rest of the diagram with new transitions so she can immediately test the new control flow (Figure 7). She next notices that exiting to the top level menu is handled inconsistently in the three different edit modes. She deletes some incorrect transitions to the menu state (Figure 8), as well as a superfluous state (Figure 9). Betty is not convinced that the mapping of available buttons to crop an image region is optimal. She selects the crop state and creates an alternative for it (Figure 10). In the alternative, she redirects button input and adds a comment for Adam to compare the two implementations.

Revision primitives & display principles

In text, the atomic unit of modification is a character. Because interactive systems have a larger set of primitives, the set of possible revision actions is more complex as well. In d.tools, the primitives are states, transitions, the device definition, and graphical screens. With each primitive, d.note defines both syntax and semantics of modification. This section will provide examples of each operation.

d.note uses color to distinguish base content from elements added and removed during revision. States and transitions rendered in black outline are elements existing in the base version; added elements are shown in blue; deleted elements in red. Currently, d.note focuses on supporting actions of a single reviewer. However, collected meta-data make distinguishing between multiple revision authors straightforward. Revised document elements could show author identity through icons, tooltips, or unique colors.

Revising Behavior

Users can add states and transitions in revision mode as they normally would; these states and transitions are rendered in blue to indicate their addition (Figure 6). These states and transitions behave like their regular counterparts.

When users remove states from the base version, the state is rendered as inactive in red. To visually communicate that this state can no longer be entered or exited, all incoming and outgoing transitions are rendered as inactive with dashed lines (Figure 9). At runtime, incoming transitions to such states are not taken, making the states unreachable.

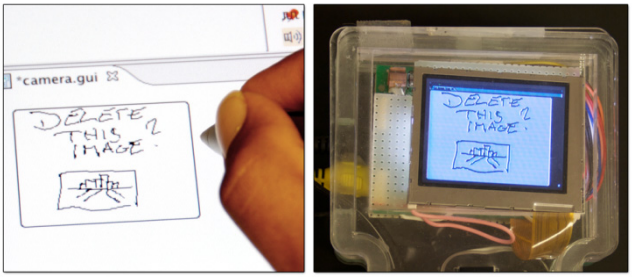

Figure 12. Sketched updates to screen content are immediately visible on attached hardware.

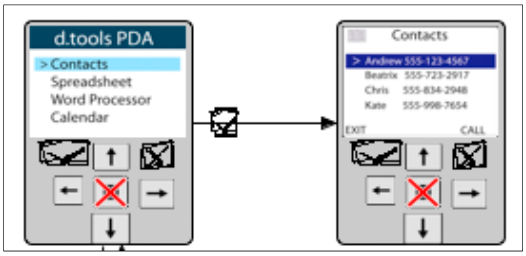

Figure 13. Changes to the device configuration are propagated to all states. Here, one button was deleted while two others were sketched in.

Individual transitions can also be directly selected and deleted. Deleted transitions are shown with a dashed red line as well as a red cross, to distinguish them from transitions that are inactive as a result of a state deletion (Figure 8). When users remove states or transitions that were added in revision mode, they are completely removed from the diagram.

Revising appearance

Designers can modify graphics by sketching directly on top of them with a pen tool within the graphics editor. Sketched changes are then rendered on top of the existing graphics in a state at runtime (Figure 12).

In addition to sketching changes to appearance, users may also rearrange or otherwise modify the different graphical components that make up the screen output of a state. d.note indicates the presence of such changes by rendering the screen outline in the state editor in a different color, as keeping the original graphics present would interfere with the intended design. The changes are thus *not* visualized on the level of an individual graphical widget.

Revising device definition

Thus far, we have described changes to the information architecture and graphic output of prototypes. When prototyping products with custom form factors such as medical devices, the set of I/O components used on the device may also be subject to change and discussion. When revising designs in d.note, users can introduce new physical hardware elements by sketching them in the device editor (Figure 5a, Figure 13). Prior to binding the new software component to an actual piece of hardware, designers can simulate its input during testing. A simulation tool in the device editor injects events for the new component into the logic model. This simulation tool is inspired by DART's Wizard of Oz prototyping support [23]. Currently, the d.note implementation does not support adding output devices; we believe adding output within this paradigm would be fairly straightforward.

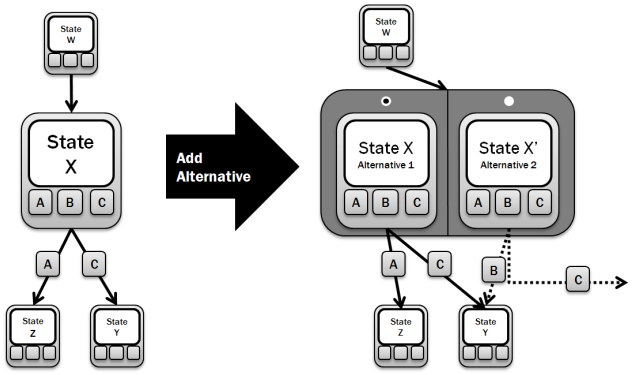

Figure 14. Schematic of state alternatives: alternatives are encapsulated in a common container. One alternative is active at a time. Alternatives have different output and different outgoing transitions.

Commenting

In addition to functional revision commands, users can sketch comments on the canvas of device, graphics, and state editors (Figure 11). Any stroke that is not recognized as a revision command is rendered as ink. This allows tentative or ambiguous change proposals to coexist with concrete changes. Inked comments are bound to the closest state so they automatically move with that state when the user rearranges the state diagram.

Proposing Alternatives

With d.note, users can introduce *alternatives* for appearance and application logic. d.note represents the alternative by duplicating the original state and visually encapsulating both original and alternative (Figure 14). Incoming transitions are re-routed to point to the encapsulating container. Each state maintains its own set of outgoing transitions. To define which of the alternative states should become active when control transfers to an alternative set, the container shows radio buttons, one above each contained state. To reduce visual clutter, only outgoing transitions of the active alternative are shown; other transitions are hidden until their alternative is activated.

THE D.NOTE JAVA IMPLEMENTATION

d.note was implemented as an extension to d.tools. As such, it was written in Java 5 and makes use of the Eclipse platform, specifically the Graphical Editing Framework [1]. d.note runs on both Windows and Mac OS X.

Specifying actions through stylus input

Because much of early design relies on sketches as a visual communication medium [3], d.note's revision interface can be either operated through mouse and keyboard commands, or it can be entirely stylus-driven. Stylus input allows for free mixing of commands and non-command sketches. When using the stylus, strokes are sent through a recognizer

(the Paper Toolkit [35] implementation of Wobbrock et al.'s $1 recognizer [33]) to check if they represent a command. Command gestures to create states and alternatives use a pigtail delimiter, to reduce the chance of misinterpretation of other rectangular strokes. Gesture recognition takes into account what existing diagram element (if any) a gesture was executed on top of. The gesture set contains commands to delete the graphical element underneath the gesture, and to create new states, transitions and alternatives. All other strokes are interpreted as comments (Figure 15).

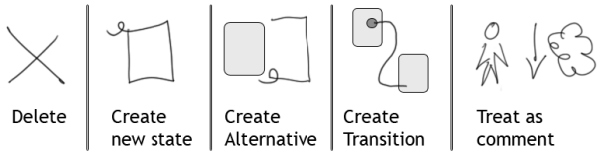

Figure 15. The d.note gesture set for stylus operation. Any stroke not interpreted as one of the first four actions is treated as a comment.

In addition to providing drawing and gesture recognition, d.note extends the d.tools runtime system to correctly handle the interaction logic semantics of its notation, *e.g.*, ignore states marked for deletion.

COMPARING INTERACTIVE AND STATIC REVISIONS

To understand the user experience of the interactive revision techniques manifest in d.note, we conducted two studies: the first compared *authoring* of revisions with and without d.note; the second compared *interpretation* of revisions with and without d.note. We recruited product design and HCI students at our university. Because the required expertise in creating UIs limited recruitment, we opted for a within-subjects design, with counter-balancing and randomization where appropriate.

Study 1: Authoring Revisions

For word processing, Wojahn [34] found that the functionality provided by a revision interface influenced the number and type of problems discussed. Do users revise interaction designs differently with an interactive tool than with freeform, static annotations on a diagram?

Method

We recruited twelve participants. Participants each completed two revision tasks: one without d.note and one with. The non-d.note condition was always assigned first to prevent the exposure to d.note notation from influencing freeform annotation patterns. Each revision task asked participants to critique one of two information appliance prototypes, one for a keychain photo viewer, and one for the navigation and management of images on a digital still camera (Figure 17). The tasks were inspired by student exercises in Sharp's interaction design textbook [29]. We counterbalanced task assignment to the conditions.

Participants were seated in front of a Mac OS X workstation with an interactive 21", 1600×1200 pixel tablet display (see Figure 16). Participants could control this workstation with stylus as well as keyboard and mouse. We first

Figure 16. Participants in study 1 revised d.tools designs on a large tablet display.

Figure 17. Participants were given a prototype device with a color display and button input. They were asked to revise designs for a keychain display and a digital camera, both running on the provided device.

demonstrated d.tools to participants and had them complete a warm-up menu navigation design (taken from [11]) to become familiar with the visual authoring language.

In the d.note condition, students were given a demonstration of its revision features, and five minutes to become familiar with the commands using the warm-up project they completed earlier. Participants were then given a working prototype, run by d.tools and d.note, and asked to take 15 minutes to revise the prototype directly in the application using d.note's commenting and revision features.

In the non-d.note condition, participants were given a working prototype along with a static image of the d.tools state diagram for the prototype. The image was loaded in Autodesk Sketchbook Pro, a tablet PC drawing application, and participants were given 15 minutes to draw modifications and comments on top of that image. While designers today often use paper for static annotation, comparing the digital d.note interface to another digital interface is a stronger minimal pairs design. Because fewer variables differ among the conditions, one can draw more confident conclusions from the results.

The caveat of our design is that ordering of conditions may have affected usage. For example, participants may have become more comfortable, or more fatigued, for the second condition. However, we judged this risk to be lower than the potential learning effect of becoming familiar with the

		Inked Annotations											D.Note Revisions				
Particpant	Task	Written Comments	Sketches on Canvas	Drawn Transition Arrows	Other Arrows	Drawing/Modifying Screen	Circled States	Circled Groups of States	Circled Transitions	Circled components	Circled State Graphics	Crossed out items	Created States	Created Alternatives	Created Transitions	Deleted Transitions	Deleted States
		Without D.Note															
4	C	8	2	7	4	1			1	2	2		N/A				
5	C	14		3	6	1	3	2	1								
7	C	10	1		3	2											
9	C	9		8	2		2				2						
10	C	14		4	3		1	2	1								
12	C	13		7	9	2		1	4		5						
1	K	5	4	6	1	1											
2	K	5		3	7		4	1		2	3	3					
3	K	10	1	1	2	2	2		2								
6	K	10		4	4	1			2			1					
8	K	13			10	1											
11	K	6		1	3				1								
		With D.note															
1	C					2							2	1	9		
2	C	3			1		1	1						1	12	3	
3*	C	4															
6	C					6							4	2	9		
8	C	1		2							1				8	7	
11	C	1				4		1					5	1	13	1	
4	K					2							1		4	5	
5	K	5			3			2									
7	K	4			1			1				7				4	2
9	K	1			1			1							10	2	
10	K	8		1	4		3	2							7	2	2
12	K	1			1										6	4	2

Table 1. Content analysis of d.tools diagrams reveals revision patterns: with d.note, participants wrote less and deleted more (Task K = keychain, C = camera).

Perceived advantages of d.note for expressing revisions	
Can test proposed changes	IIIIIII
Can make functional changes	IIIIII
Less cluttered than drawing	II
Notation easier to interpret	II
Can express alternatives	I

Perceived disadvantages of d.note for expressing revisions	
Commenting is more difficult	III
Steeper learning curve	II
Danger of getting stuck on details	II
Lack of rich drawing tools	I
Diagramms become too cluttered	I

Table 2. Most frequently mentioned advantages and disadvantages of using d.note to *express* revisions.

d.note annotation language and then applying it in the non-d.note condition. After the design reviews, participants completed a survey that elicited high-level summative feedback in free response format.

Results

Figure 18 shows two examples of diagrams produced by participants. We categorized all marks participants made; Table 1 summarizes the results. Most notably, participants wrote significantly more text comments *without* d.note than with it (mean: 9.8 without, 2.3 with; two-tailed t(22)=6.20, p<0.0001). In contrast, deletions were rare without d.note (4 occurrences); but common with d.note (34 occurrences; 8 out of 12 participants). Finally, revisions with d.note focused on changes to the information architecture, while freeform revisions often critiqued the prototype on a more abstract level. Our results thus corroborate Wojahn's finding that the choice of revision tool affects the number and type of revision actions [34].

The post-test survey asked participants to compare the relative merits of Sketchbook and d.note. We categorized their freeform written answers (Table 2). The two most frequently cited advantages of d.note were the ability *to make functional changes* (6 of 12 participants), and to then *test proposed changes right away* (7 of 12 participants).

Three participants suggested that commenting was more difficult with d.note; two wrote that the tool had a steeper learning curve. Two participants with a product design background wrote that using d.note led them to focus too much on the details of the design. In their view, the lack of functionality in the Sketchbook condition encouraged more holistic thinking.

Discussion

Why did participants write less with d.note? One possibility is that that users wrote more with Sketchbook because it was easier to do so (Sketchbook is a polished product, d.note a research prototype). To the extent this is true, it provides impetus to refine the d.note implementation, but tells us little about the relative efficacy of static and dynamic approaches to design revision.

More fundamentally, d.note may enable users to capture intended changes in a more succinct form than text comments. Four participants explicitly wrote that d.note reduced the need for long, explanatory text comments in their survey responses: "[with d.note] making a new state is a lot shorter than writing a comment explaining a new state"; "[without d.note] I felt I had to explain my sketches." d.note's rich semantics enable a user's input to be more economical: an added or deleted transition is

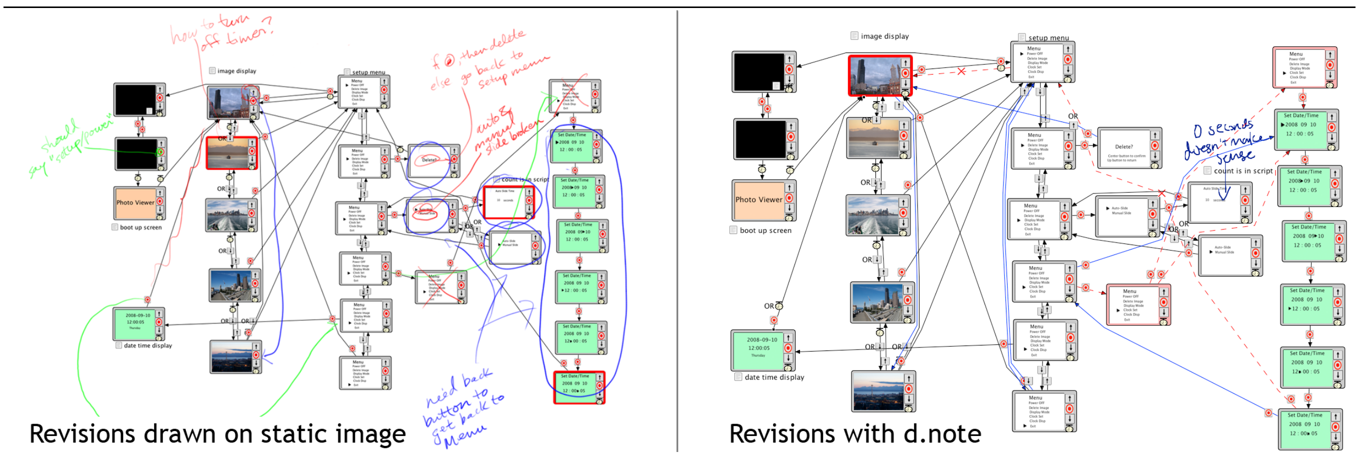

Figure 18. Two revision diagrams produced by our study participants for the keychain photo viewer task.

unambiguously visualized as such. In d.note, users can implement concrete changes interactively; only abstract or complex changes require comments. Without d.note, both these functions have to be performed through the same notation (drawing), and participants explained their graphic marks with additional text because of this ambiguity. In our data, inked transition arrows drawn without d.note (44 drawn transitions) were replaced with functional transitions with d.note (78 functional transitions added; only 3 drawn as comments).

Though participants could have disregarded the revision tools and only commented with ink, the mere option of having functional revision tools available had an effect on their activity. This tendency has been noted in other work [3,20]; understanding the tradeoff deserves future research.

Why did participants delete more with d.note? While participants created new states and transitions in both conditions, deletions were rare without d.note. Deletions may have been implied, *e.g.*, drawing a new transition to replace a previously existing one, but these substitutions were rarely noted explicitly. We suggest that deletions with d.note were encouraged by the ability to immediately test concrete changes. Quick revise-test cycles exposed areas in which diagrams had ambiguous control structure (more than one transition exiting a state on the same event).

Why were more changes to information architecture made with d.note? The majority of revision actions with d.note concerned the flow of control: adding and deleting transitions and states. In the Sketchbook condition, participants also revised the information architecture, but frequently focused on more abstract changes (Example comment: "Make [feedback] messages more apparent"). The scarcity of such comments with d.note is somewhat surprising, as freeform commenting was equally available. One possible explanation is that participants focused on revising information architecture because more powerful techniques were at hand to do so. Each tool embodies a preferred method of use; even if other styles of work remain possible, users are driven to favor the style for which the tool offers the most leverage.

Study 2: Interpreting Revisions

The first study uncovered differences in expressing revisions. Are there similar characteristic differences in interpreting revisions created with the two tools?

Method

Eight (different) participants interpreted the revisions created by participants of the first study. After a demonstration and warm-up task (as in study 1), participants were shown the two working prototypes and given time to explore them. Next, participants were shown screenshots of annotated diagrams (see Figure 18) on a second display. Participants were asked to prepare two lists in a word processor: one that enumerated all revision suggestions that were clear and understandable to them; and a second list with questions for clarification about suggestions they did not understand. Participants completed this task four times:

S2 Participant	Task	S1 Participant	With D.Note: Clear Annotations	With D.Note: Unclear Annotations	S1 Participant	Without D.Note: Clear Annotations	Without D.Note: Unclear
1	C	6	IIIIII	I	7	IIIIII	
	K	4	IIII		3	IIIII	IIIIII
2	C	8	IIII	III	9	IIIIIIII	I
	K	12	IIIIII		2	IIIIII	IIII
3	C	11	IIIIIIIII	IIIIII	4	IIIIIIII	IIIII
	K	9	IIIII	I	8	IIIIIIIII	IIIIII
4	C	2	IIIIIIIIIIIIIII	I	10	IIIIIIIIII	II
	K	10	IIIIIIIIII		11	IIII	II
5	C	1	IIII	I	5	IIIIIII	III
	K	4	III	I	6	IIIII	I
6	C	11	IIIIII	III	12	IIIIIIII	IIII
	K	10	IIII	IIIII	5	IIIII	II
7	C	6	IIIIII	I	7	IIIIIIII	
	K	12	IIIIIII		3	IIIII	IIIIII
8	C	2	IIIIIIIIIIIIII	II	4	IIIIII	IIII
	K	9	IIII	I	8	IIIIIIIII	I

Table 3. How well could study 2 participants interpret the revisions created by others? Each vertical bar is one instance.

Perceived advantages of d.note for interpreting revisions		Perceived disadvantages of d.note for interpreting revisions	
Changes are concrete and specific	IIII	Visual clutter in regions of dense changes	IIIII
Contains proposed solutions	III	Hard to glean motivation for changes	IIII
Can automatically apply changes	I	Hard to keep track which changes were already examined	II

Table 4. Perceived advantages and disadvantages of using d.note to *interpret* revisions as reported by study participants.

one d.note and one freeform diagram were chosen at random for each of the two prototypes.

Results

The cumulative counts of clear and unclear revision suggestions for all participants are shown in Table 3. Participants, on average, requested 1.3 fewer clarifications on revisions when using d.note than when sketching on static images (two-sample $t(29)=1.90$, $p=0.03$).

The post-test survey asked participants to compare the relative merits of interpreting diagrams revised with d.note and Sketchbook. The most frequently mentioned benefits arose from having a notation with specified semantics (Table 4): revisions were more concrete, specific, and actionable. Frequently mentioned drawbacks were visual complexity and problems discerning high-level motivation in d.note diagrams.

Discussion

Why did participants ask for fewer clarifications with d.note? When interpreting revised diagrams, participants are faced with three questions: First, what is the proposed change? Second, why was this change proposed? Third, how would I realize that change? The structure of this study asked participants to explicitly answer the first question by transcribing all proposed changes. We suggest that the formal notation in d.note decreased the need for clarification for two reasons. First, the presence of a formal notation resulted in a smaller number of handwritten comments, and hence less problems with legibility (Example without d.note: "Change 6 - unreadable"). Second, because of the

ad-hoc nature of handwritten annotation schemes in absence of a formal system, even if comments were legible, participants frequently had trouble tying the comments to concrete items in the interface (Example: "I have no idea what it means to 'make it clear that there is a manual mode from the hierarchy'. What particular hierarchy are we talking about?").

How might we improve capturing the motivation for changes? In the survey, participants commented that it was harder to understand why certain changes were proposed in d.note. While handwritten comments focused on high-level goals without specifying implementations, tracked changes make the opposite tradeoff: the implementation is obvious since it is already specified, but the motivation behind the change can remain opaque. We see two possible avenues to address this challenge. First, when using change tracking, multiple individual changes may be semantically related. For example, deleting one state and adding a new state in its stead are two actions that express a desired single intent of replacement. The authoring tool should detect such related actions automatically or enable users to specify groups of related changes manually. Second, even though freeform commenting was available in d.note, it was not used frequently. Techniques that proactively encourage users to capture the rationale for changes may be useful.

How might we reduce the visual complexity of annotated diagrams? Visual programs become harder to read as node and link density increases. Showing added and deleted elements simultaneously in the diagram sometimes yielded "visual spaghetti": a high density of transition lines that made it hard to distinguish one line from another. The connection density problem becomes worse when state alternatives are introduced because each alternative for a state has an independent set of outbound transitions.

In response, we already modified the drawing algorithm for state alternatives to only show outgoing connections for the currently active alternative within an alternative container. Additional simplification techniques are needed though. The Topiary system [21] highlights incoming and outgoing transitions to make them visually prominent. A further step in this direction would be to only render incoming and outgoing transitions for a highlighted state and hide all other transitions on demand.

A Design Space of User Interface Revision Tools

What can be revised?	Information Architecture	Static Screen Content	Dynamic Behavior	
How concrete are revisions?	Suggest Problem	Suggest Change	Demonstrate Change	Implement Change
Where are revisions captured?	Diagrams of UI Structure	Source Code	Static Screen Images (Runtime artifacts)	Recording of Running Application (Video)
	Source artifacts			
What modalities are used for input?	Digital Ink	Direct Manipulation	Text	Voice Annotation / Video Annotation
When are changes computed?	Incrementally, Online	Between two separate versions, Offline		

Table 5. A design space of user interface revision tools. The sub-space d.note explored is outlined in green.

THE DESIGN SPACE OF REVISION TOOLS

The particular implementation of revision techniques in d.note represents only one point solution in a larger design space of possible user interface revision tools. The main salient dimensions we considered during our work are summarized in Table 5. d.note focuses on revision of information architecture and screen content of user interfaces through sketching of comments and modifications on top of UI state diagrams and screen images. In our study, these functions were used to point out problems and to suggest as well as implement changes. The design space reveals additional areas of exploration we have not touched upon so far. For example, it is not yet possible to directly *modify* dynamic behaviors such as animations, as those are defined in source code. In fact, it is not even feasible to efficiently *comment* on dynamic behaviors either, as there is no visual record of them in the interaction diagram. Recording and annotating video of runtime behavior is one promising avenue to enable commenting on dynamic aspects. Many usability testing tools already support video annotation. How to tie comments and annotations back to the source representation of the UI is an open question.

The particular revision actions of d.note are based on a visual language that shows both user interface content and information architecture in the same environment. d.note techniques directly transfer to other authoring environments that use UI states as their primary abstraction. Such tools exist both in research (*e.g.*, DENIM [22], SUEDE [17]) and industry (*e.g.*, Adobe Flash Catalyst, which is based on states and transitions, though with different visual layout). In addition, change visualization for node-link diagrams of interactive systems can also apply to popular commercial data flow authoring environments such as MaX/MSP and Apple Quartz Composer. But how might we express revisions for user interfaces specified entirely in source code? Existing source revision techniques do not permit designers to comment or revise the output of their application. Future research should investigate if sketch-based input and annotation in the output domain of a program can be applied to UIs expressed in textual source code.

CONCLUSION

This paper introduced the d.note revision notation for interaction design. It contributed an analysis of how to transfer principles of document revision to the domain of interaction design and introduced concerns unique to the revision of interaction designs: design alternatives as a revision operation; and immediate testing of proposed functional revisions.

The paper also evaluated d.note against freeform sketched comments in two studies. The first study on revision production found that the type of revision tool used had an impact on the type and number of revisions: participants wrote less, deleted more, and focused their changes on

information architecture when using d.note. The second study on revision interpretation found that participants asked for fewer clarifications about revisions, but had less insight into the motivations behind revisions when using d.note. Our study pointed out that optimally balancing both structured and informal feedback may not be straightforward. Fundamentally, the presence of functional revision tools appeared to discourage participants from freeform commenting. Future work should address how to structure a revision tool so that it leads to more balanced suggestions.

REFERENCES

1. Eclipse Graphical Editing Framework (GEF). http://www.eclipse.org/gef/.
2. Bailey, B.P., Konstan, J.A., and Carlis, J.V. DEMAIS: designing multimedia applications with interactive storyboards. *Proceedings of the ACM international conference on Multimedia*, ACM (2001), 241-250.
3. Buxton, B. *Sketching User Experiences: Getting the Design Right and the Right Design, Chapter on Visual Story Telling*. Morgan Kaufmann, 2007.
4. Cross, N. *Designerly Ways of Knowing*. Springer, 2006.
5. Drucker, S.M., Petschnigg, G., and Agrawala, M. Comparing and managing multiple versions of slide presentations. *Proceedings of UIST 2006*, ACM (2006), 47-56.
6. Girschick, M. *Difference detection and visualization in UML class diagrams*. Report TUD-2006-05, TU Darmstadt, 2006.
7. Guimbretière, F. Paper augmented digital documents. *Proceedings of UIST 2003*, ACM (2003), 51-60.
8. Hailpern, J., Hinterbichler, E., Leppert, C., Cook, D., and Bailey, B.P. TEAM STORM: demonstrating an interaction model for working with multiple ideas during creative group work. *Proceedings of Creativity and Cognition 2007*, ACM (2007), 193-202.
9. Harel, D. Statecharts: A Visual Formalism For Complex Systems. *Sci. of Computer Programming 8*, (1987), 231-274.
10. Hartmann, B., Yu, L., Allison, A., Yang, Y., and Klemmer, S.R. Design As Exploration: Creating Interface Alternatives through Parallel Authoring and Runtime Tuning. *Proceedings of UIST 2008*, ACM (2008).
11. Hartmann, B., Klemmer, S.R., Bernstein, M., et al. Reflective physical prototyping through integrated design, test, and analysis. *Proceedings of UIST 2006*, ACM (2006), 299-308.
12. Heckel, P. A technique for isolating differences between files. *Communications of the ACM 21*, 4 (1978), 264-268.
13. Heer, J., Mackinlay, J.D., Stolte, C., and Agrawala, M. Graphical Histories for Visualization: Supporting Analysis, Communication, and Evaluation. *Proceedings of IEEE Information Visualization 2008*, IEEE (2008).
14. Hunt, J.W. and McIlroy, M.D. An Algorithm for Differential File Comparison. *Computing Science Technichal Report #41, Bell Laboratories*, (1976).
15. Jones, J.C. *Design Methods*. Wiley, 1992.
16. Klemmer, S. Integrating physical and digital interactions. *Computer 38*, 10 (2005), 111-113.
17. Klemmer, S.R., Sinha, A.K., Chen, J., Landay, J.A., Aboobaker, N., and Wang, A. Suede: a Wizard of Oz prototyping tool for speech user interfaces. *Proceedings of UIST 2000*, ACM (2000), 1-10.
18. Klemmer, S.R., Thomsen, M., Phelps-Goodman, E., Lee, R., and Landay, J.A. Where do web sites come from?: capturing and interacting with design history. *Proceedings of CHI 2002*, ACM (2002), 1-8.
19. Kurlander, D. and Feiner, S. Editable Graphical Histories. *Workshop on Visual Languages*, IEEE (1988), 127-134.
20. Landay, J. and Myers, B. Sketching Interfaces: Toward More Human Interface Design. *Computer 34*, 3 (2001), 56-64.
21. Li, Y., Hong, J.I., and Landay, J.A. Topiary: a tool for prototyping location-enhanced applications. *Proceedings of UIST 2004*, ACM (2004), 217-226.
22. Lin, J., Newman, M., Hong, J., and Landay, J. DENIM: finding a tighter fit between tools and practice for Web site design. *Proceedings of CHI 2000*, ACM (2000), 510-517.
23. MacIntyre, B., Gandy, M., Dow, S., and Bolter, J. DART: a toolkit for rapid design exploration of augmented reality experiences. *Proceedings of CHI 2004*, ACM (2004), 197-206.
24. Mehra, A., Grundy, J., and Hosking, J. A generic approach to supporting diagram differencing and merging for collaborative design. *Proceedings of the International Conference on Automated software engineering*, ACM (2005), 204-213.
25. Moggridge, B. *Designing Interactions*. The MIT Press, 2007.
26. Neuwirth, C.M., Kaufer, D.S., Chandhok, R., and Morris, J.H. Issues in the design of computer support for co-authoring and commenting. *Proceedings of CSCW 1990*, ACM (1990), 183-195.
27. Perry, M. and Sanderson, D. Coordinating joint design work: the role of communication and artefacts. *Design Studies 19*, 3 (1998), 273-288.
28. Schipper, A., Fuhrmann, H., and Hanxleden, R.V. Visual Comparison of Graphical Models. *Proceedings of the IEEE Int'l Conference on Engineering of Complex Computer Systems*, IEEE Computer Society (2009), 335-340.
29. Sharp, H., Rogers, Y., and Preece, J. *Interaction Design: Beyond Human-Computer Interaction*. Wiley, 2007.
30. Song, H., Guimbretière, F., Hu, C., and Lipson, H. ModelCraft: capturing freehand annotations and edits on physical 3D models. *Proceedings of UIST 2006*, ACM (2006), 13-22.
31. Su, S. Visualizing, Editing, and Inferring Structure in 2D Graphics. *Adjunct Proceedings of UIST 2007*, ACM (2007).
32. Terry, M., Mynatt, E.D., Nakakoji, K., and Yamamoto, Y. Variation in element and action: supporting simultaneous development of alternative solutions. *Proceedings of CHI 2004*, ACM (2004), 711-718.
33. Wobbrock, J.O., Wilson, A.D., and Li, Y. Gestures without libraries, toolkits or training: a $1 recognizer for user interface prototypes. *Proceedings of UIST 2007*, ACM (2007), 159-168.
34. Wojahn, P.G., Neuwirth, C.M., and Bullock, B. Effects of interfaces for annotation on communication in a collaborative task. *Proceedings of CHI 1998*, ACM (1998), 456-463.
35. Yeh, R., Paepcke, A., and Klemmer, S.R. Iterative Design and Evaluation of an Event Architecture for Pen-and-Paper Interfaces. *Proceedings of UIST 2008*, ACM (2008).