Agile Usage-Centered Design of a Social Networking Contact Book

J. David Beutel
Spring 2009 Semester
ICS 667
Dr. Dan Suthers

Abstract

I evaluated the utility of Constantine & Lockwood's (2002) streamlined and simplified variant of the Usage-Centered Design method over a period of 10 weeks, by applying it to a web application for sharing and updating contact information between family and friends. My goal was to gain familiarity with the design method for my own potential use in the future, and to start creating an application that I had wanted for years. The design method proved useful.

Introduction

Constantine (2002) describes agile usage-centered design as a middle ground "between the world of hack-and-slash programming and that of obsessive-compulsive documania." The agile method (Fowler, 2005) of organically growing (Alexander, 1979) and refactoring implementation code (Kerievsky, 2005) is a healthy approach to small development projects, but there are drawbacks to changing the user interface in the same way. It breaks any "benign habituation" (Raskin, 2000) that users have for the interface, the "reliance" mode of attention that Krippendorff (2006) defines as usability from a human-centered perspective. Changes to the user interface created "anger and frustration" in users upgraded from Microsoft Office version 2003 to 2007 (Cummings, 2007), and a revolt when Facebook redesigned user home pages (Gaudin, March 2009). Minimizing such disruption is worth a little up-front design of the user interface. Another potential advantage is to start with a design that would be difficult to reach via the incremental improvements of agile iterations.

I followed the method outlined in "Usage-Centered Engineering for Web Applications" (Constantine & Lockwood, 2002) in particular, with some elaboration from "Process Agility and Software Usability: Toward Lightweight Usage-Centered Design" (Constantine, 2002). This method balances design with easy integration of agile development.

Method

Agile usage-centered design varies in a number of ways from conventional UCD (Constantine & Lockwood, 1999, 2000). It is lighter weight:

- It is based entirely on index cards.
- It has no operational model, domain model, or user role map.
- The user role model is deemphasized as a bridge to the task model. (Constantine, 2000)
- Instead of a use case map, the task case cards are just clustered by affinity, to approximate

interaction contexts. (see Illustration 1)

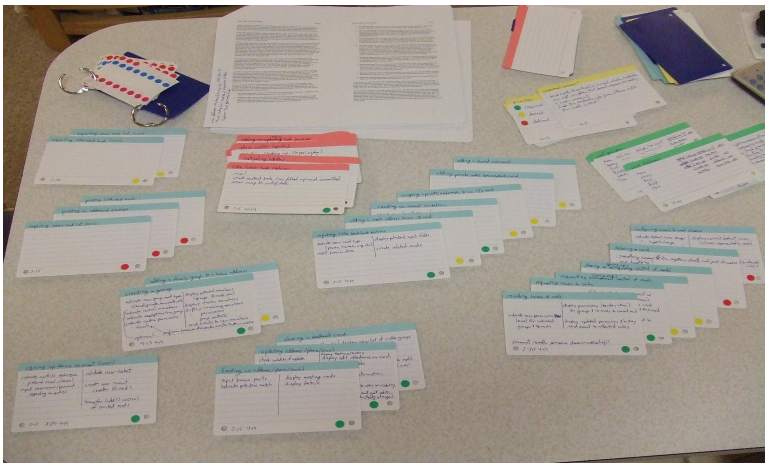

Illustration 1: Task clustering.

- Essential use cases (see Illustration 2) are done on only the task cards which are critical, complex, unclear, or interesting, not routine or obvious.

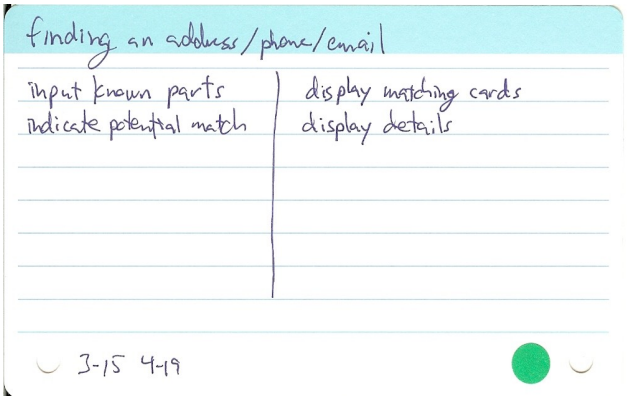

Illustration 2: Task - finding an address/phone/email.

- The detail and amount of modeling is reduced by "just-in-time requirements" allowed by easy access to users and domain experts.
- Instead of a navigation map, a navigation architecture is based on the task clusters and provides a framework for the anticipated interaction contexts.
- Instead of a final visual and interaction design, a visual and interaction *scheme* is based on all the tasks. It's a sort of abstract style guide, briefly describing the basic recurrent visual elements and common templates that apply to various interaction contexts. The purpose of this and the navigation architecture is to avoid radical refactoring or inconsistency across design increments, by anticipating the overall application, but without premature details.
- The content model (aka abstract prototype) is optional. Under pressure, a designer "may move directly into sketching realistic paper prototypes." (Constantine & Lockwood, 2002)
- Paper prototypes are created for only the interaction contexts and task cases which were selected as required for the next iteration.

Agile UCD also adds two preliminary steps (Constantine & Lockwood, 2002):

- "Essential purpose and preconception: clarify business and user purposes then fantasize and set aside preconceptions of features, facilities, content and capabilities."
- "Exploratory modeling: identify questions, ambiguities, and areas of risk and uncertainty."

Finally, Agile UCD's focus is not on producing design artifacts, but on helping the designer gain the knowledge and understanding he needs to do the design, what Krippendorff (2006) described as second-order understanding. The navigation architecture, visual and interaction scheme, essential use cases, and paper prototypes are useful for development, but they do not fully externalize the designer's knowledge. Instead, this method uses frequent meetings and continual consultation with the rest of the development team and the customers (domain experts and representative end users). (Constantine & Lockwood, 2002) This is consistent with the Manifesto for Agile Software Development (Beck et al., 2001):

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

Process

Participants

This lightweight method was a good match for the limited resources of my project: about 1 graduate credit-hour. Since I chose an application that I had been wanting to build for years, I was the primary domain expert and representative end user as well as designer. That's unlike a commercial project, but it avoids the agile pitfall of not having enough access to the customer, and it would be unrealistic to expect anyone else to dedicate the necessary time to my project.

Nevertheless, as instructed by the Agile UCD method, I did get input from other potential users. I discussed the essential purpose and preconception, exploratory modeling, role modeling, and task modeling:

- with my mother over the phone twice during the first two weeks
- with my ICS 667 classmates and Dr. Suthers during class at weeks 1, 2, and 6
- with Dr. Suthers via disCourse from his helpful guidance and questions in weeks 3 and 5 in response to the notes and issues I posted in my project section

Finally, Agile UCD calls for a collaborative usability inspection, which we did during the second-to-last class (project week 9) using some of the paper prototypes and visual and interaction schemes.

Agile UCD does not call for the use of an asynchronous collaboration system like disCourse, since it specifies face-to-face meetings with index cards. However, my application's purpose is remote collaboration, so most of the stakeholders cannot meet face-to-face. In the early stages of the project, I hoped that some would participate via disCourse, so I used it, especially for the exploratory modeling. Unfortunately, Dr. Suthers was the only other person who contributed to the discussion on disCourse. This suggests that the face-to-face method specified by Agile UCD has an advantage, although also limiting where it can be used.

My project's section on disCourse contains:

contents	***description***
11 pages	corresponding to UCD models

contents	*description*
5 discussions	totaling 35 messages, mostly from myself
48 scanned index cards	mainly paper prototypes and interaction schemes for the collaborative usability inspection in class

I also recruited stakeholders from my Facebook friends, hoping some would be interested because I may put their addresses in the application. I value their input in particular because of their experience on Facebook, and because I think it makes them more likely to use my application. Three of them, all relatives of mine, actually got disCourse accounts in my project space, logged in, and looked at some of the discussion. However, none posted any feedback. I suspect the discussion was too abstract or dense for them. Perhaps they can participate later with usability inspections or testing, when a prototype is available.

I tried to involve a couple other friends and coworkers as stakeholders face-to-face, but they did not have the time or interest to discuss the design in any depth. This demonstrates that face-to-face access does not necessarily solve the issue of participation.

Essential purpose and preconception

Agile UCD calls for kicking off the process by involving stakeholders in framing the purposes of the application from the perspective of external users, as well as from the business perspective (which doesn't apply to my non-commercial project). It suggests brainstorming these purposes onto index cards, which can be sorted by priority. And, it encourages participants to share fantasies about how the application will be, but label them as preconceptions and set them aside.

Because of this step in the process, I reached out to my mother, classmates, and instructor. I started by making notes on paper during the phone call and class discussion, instead of using index cards, because I then transcribed the notes onto disCourse for use as described in the section above.

This preliminary step had several positive results:

- It got me to involve other stakeholders and gave me a point at which to start the discussion with them.
- My preconceptions helped the other stakeholders understand what application we were discussing, since I was initiating it, so they had not yet conceived of it.
- The other stakeholders provided some fundamental requirements at an early stage. For example:
    - With the first phone call to my mother, I got the importance of fine-grained privacy controls. I realized that users will need to be able to mask any part of their contact info, either globally or to certain groups or individuals. I hadn't considered letting users control access to every little piece of data, since I assumed that would be too complicated. This also raised questions for the following step, exploratory modeling.
    - From the week 2 class discussion, a classmate, Salvatore Aurigemma, made me realize the importance of private notes which cannot be accidentally shared or mixed up with notes of higher visibility.

Exploratory modeling

The second preliminary step of Agile UCS, exploratory modeling, is to identify questions, ambiguities,

and areas of risk and uncertainty, by sketching out the role and task models. For this step, I identified issues, noted them in disCourse, and discussed with Mom, in class, and on-line.

This step took a lot more time than I had scheduled. I started it in week 1, but got stuck for almost half the ten weeks of the project. The issues that were holding up the completion of the next steps, the role and task models proper, were basically requirements. For example, should the access rights to shared contact information be transitive, so downstream users loose their access when their upstream user is cut off? UCD assumes that the users already know their tasks or domain and can explain them to the designer. However, with my application, there is no boss telling the users their job. The usage is optional. Nobody knows yet exactly how or whether my application will be used.

Usage-centered design did not help me cope with this kind of ambiguity about certain task cases. Its contrasting method, *user*-centered design, seemed to offer more help for this step– focusing on user experience and satisfaction via user studies, participatory design, and feedback. Inspired by Krippendorff's methods, I turned towards recruiting more stakeholders, such as from Facebook, and looking harder for answers from them. However, I was never able to actually resolve the issues via this user-centered detour.

I finally got past this step by using two different strategies. One was to follow Dr. Suthers' hint about providing tools for the users to find their own answers. Each question became a meta-requirement to provide the users with a way to choose their own answer. The other strategy was to rely on myself as the primary representative end user, instead of trying to guess what the other users are going to want. I think that will provide a reasonable starting point for this kind of project.

Nevertheless, this exploratory modeling was an important step. Without it, I wouldn't have arrived at transitive permissions, or groups that are copied by reference and difference. These issues had a significant impact on the design, avoiding later disruption.

Another issue highlighted by this step was terminology. As Krippendorff (2006) explained, metaphors are important for design. However, so far it has proved less important to the design itself than to the discussion about the design, because not all of the terms appeared in the design.

In (Constantine, 2002), these two preliminary steps are not distinguished from the role and task modeling steps, but I think it's helpful to consider them as something different. Dealing with the uncertainty at this point would have frustrated me more if it weren't specifically allowed by having these steps dedicated to it.

Role modeling

Agile UCD's first iterative step is modeling roles that users can play in relation to the application: brainstorming roles directly onto index cards, then briefly describing salient aspects, and ranking them in order of priority for project success. I did this from weeks 3 to 8, starting from the exploratory modeling sketch of the previous step, using cards with a red color-coded top. Dr. Suthers also added the roles to my disCourse project space after I reviewed them in class during week 3.

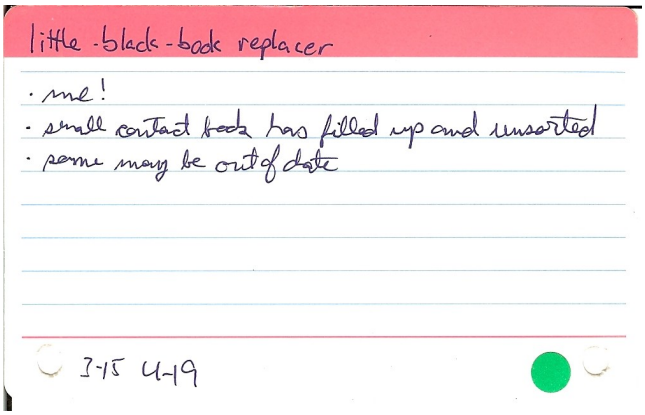

Illustration 3: Role - little black book replacer

This step helped me recall one of my primary objectives for this application, replacing my little-black-book, which filled up years ago into an unsorted tangle of partly obsolete information. (Illustration 3) When I started this project in February, the role on my mind was instead someone maintaining an Xmas card list, so I might have forgotten about the other role. It's good to start from this level, before focusing on the trees which may obscure the forest. There were 23 roles in the end.

Having the roles on cards and relegating some of them down to a deferred priority was also useful. I don't need everyone to accept and use this applications, just enough to help maintain the information. This let me exclude difficult users, such as people without computers or familiarity with the web, who would be difficult to include in the design. (Illustration 4)

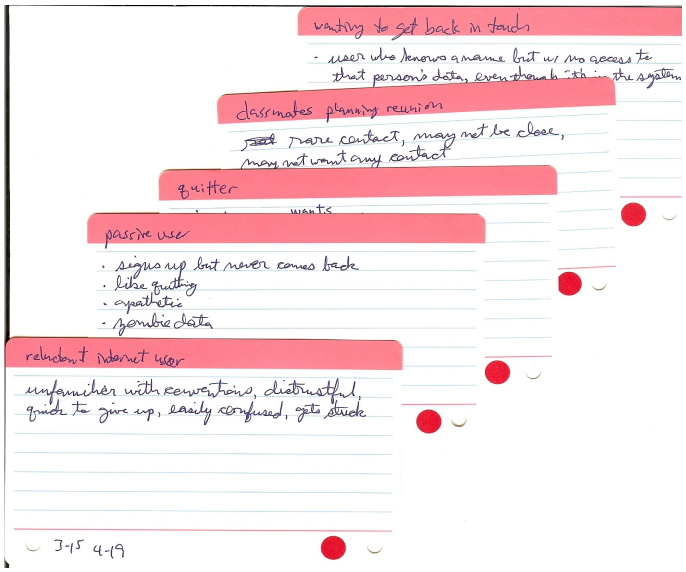

Illustration 4: deferred roles

In the week 3 class discussion, a continuum of abstraction emerged. At the essential end of the spectrum were 4 roles with respect to data access rights: controller (owner), maintainer, accessor, and other (no access). That may be closer to the domain model than what UCD considers to be roles. At the other end of the spectrum were roles so narrow that I reclassified one of them as a task (look at a calendar). Most roles were in between, having some context and characteristic patterns of interaction, as UCD intends, to help inform the task model.

Task modeling

Agile UCD's second iterative step is task modeling: inventory task cases directly onto index cards, as anticipated by the roles. I developed these task cards, 25 in all, from weeks 3 to 10, generally after the roles. UCD calls for sorting the cards by expected frequency and overall importance, and then triaging into 3 piles: required, desired (if time), and deferred (next time). Using cards made them easy to sort and split up. I used color-coded stickers for the triage results (green, yellow, and red, respectively), so it's not lost in the following step, task clustering. I used task cards with a teal color-coded top.

The next part of this step is to fill in essential use cases on the task cards which are not deferred, obvious, or routine. The limited space on the cards helped keep the use cases at the right level of abstraction. The cards also afforded simple modifications and some history. I noted the month-day at the bottom of each card when I modified it. (Illustration 5)

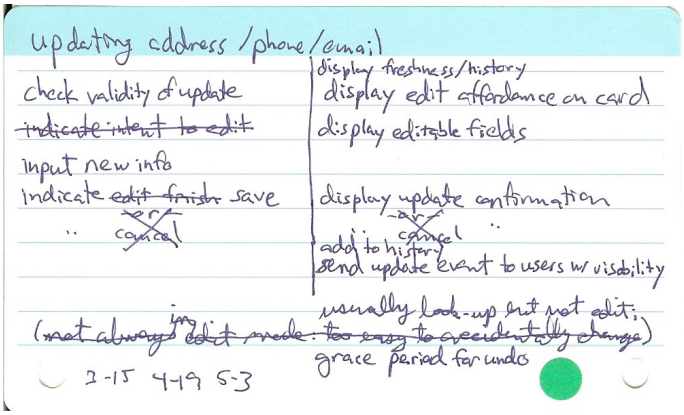

Illustration 5: Task - updating address/phone/email

Task clustering

The Agile UCD method's next step, which I did in week 8, is to group the task cards into affinity clusters based on how strongly they seem to be related or how likely they are of being performed together. The cards and my familiarity with them made this easy to do. (Illustration 1)

Each cluster represents a set of capabilities that need to be in the same, or closely connected, interaction context (such as a page). Based on this, Agile UCD calls for drafting an overall navigation architecture, which I did on one index card. (Illustration 6)

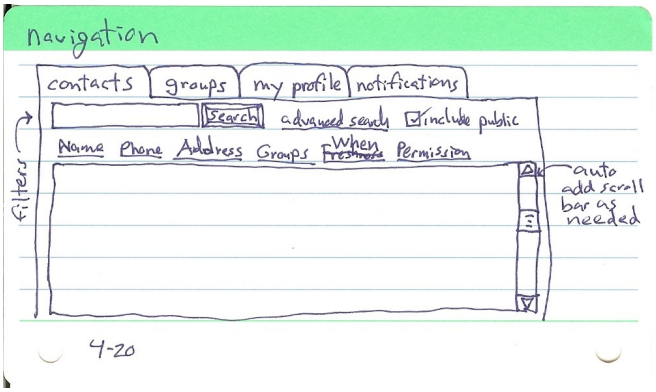

Illustration 6: navigation architecture

The Agile variant of UCD does not require a content model (aka abstract prototype), and I did not construct one. Instead of traditional UCD's navigation map, which is a part of the content model, I used the tasks and clusters to create the navigation architecture and the subsequent steps: visual and interaction scheme, and detailed user interface design. The task cards in this case are serving as rough proxies for the content model. This reminds me of a strategy game, San Juan, which is a simplified card version of Puerto Rico. San Juan overloads its cards to represent money, products, plantations, buildings, and points, which are represented in the original game as separate kinds of tokens.

Visual and interaction scheme

In addition to a navigation architecture, Agile UCD also calls for a visual and interaction scheme, to round out the overall user interface design without getting into premature details. I did this step next, from week 8 to 10, on ten green color-coded index cards.

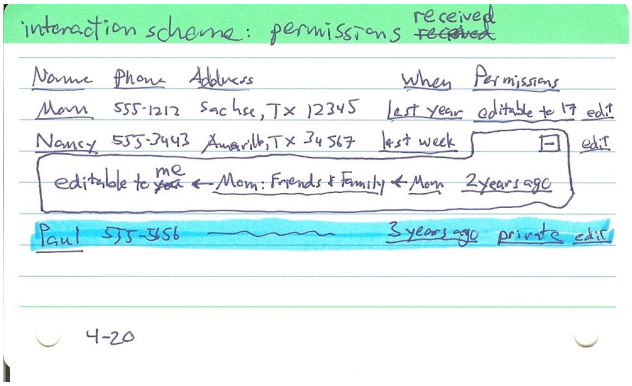

Illustration 7: Interaction Scheme – permissions received

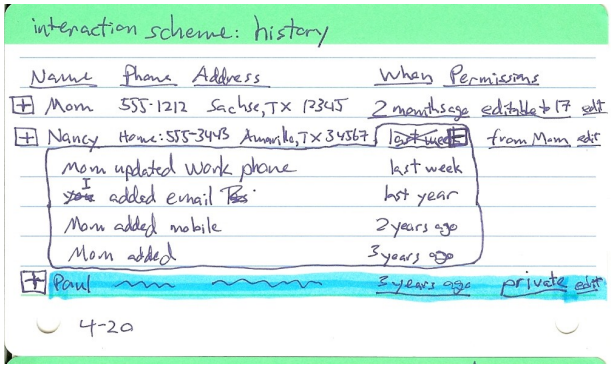

Illustration 8: Interaction Scheme - history

I drew schemes of permissions (Illustration 7), history (Illustration 8), details, and editing. Constantine & Lockwood (2002) do not specify using index cards for the paper prototypes, but I did anyway, and the cards seem to have worked well. Although their small size required continuation cards and limited the detail to unrealistic levels, it also forced me to focus on the parts important to each card, like an essential use case. This made the card contents more orthogonal, and combined with their small size, limited the amount I had to redraw when revising the design.

The permissions scheme is based on a suggestion from Dr. Suthers in the project's Exploratory Modeling disCourse. He wrote that Nathan Dwyer's dissertation abstracted three major dimensions that we can manipulate in the communicative properties of media: perceptability (visibility), mutability (who can edit it), and relationship. In my application, the information is the media. The permissions column shows a summary of those properties, at the row's current level of detail, if it's different from its container. The user can drill down into the permissions detail to see who has what via what relationship. With this level of control, the users can set the permissions to their individual needs.

Detailed user interface design

The last step of the design iteration is sketching realistic paper prototypes, augmented as necessary with notes about how various elements behave, for select interaction contexts. I drew 33 such cards, similar to the visual and interaction scheme, from weeks 8 to 10. Many of them I named in line with their corresponding task name.

Eight of those cards were revisions or extensions following a collaborative usability inspection, described below, which is recommended as part of this step to identify usability problems and areas for improvement. Compare the original paper prototype of "updating address/phone/email" (Illustration 9)

with the result of the collaborative usability inspection described below.

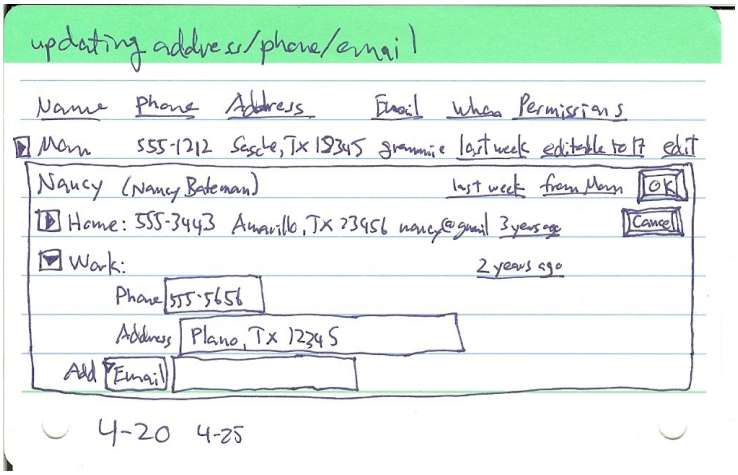

Illustration 9: Paper Prototype - updating address/phone/email

Collaborative usability inspection

I led a collaborative usability inspection (Lockwood, 1999) in class at week 9. I played the roles of Lead Reviewer and Developer, preparing use case scenarios and enacting the software with the paper prototypes scanned into disCourse. Dr. Suthers played the role of Recorder, while my classmates played the role of Users. We all played the roles of Usability Specialists and Designers.

We did not have time in that class for a continuity review or even to look at most of the use cases. We got through a couple of the scenarios had I prepared, starting with a single use case (signing up for an account), and moving on to some key, typical interactions (signing in, finding an address, and updating details). Leading the scenarios and enacting the software reminded me of playing Dungeons and Dragons. Nevertheless, I received useful feedback from everyone.

For example, regarding the "When" column and its links to the past (last week, 2 years ago, etc), there was confusion over their meaning. I revised the history interaction scheme (Illustration 10 & 11) to try to avoid that problem, by changing the "When" column to "History", and by using consistent expanding drill-down arrow buttons instead of links.

As another example, on the "updating" prototype, there were problems with the locus of attention when needing to click "edit" to get into edit mode, the ambiguity of the "OK" button to save changes, and the issue of having any edit mode at all (as Raskin decried modes in general). I tried to avoid those problems by adding an interaction scheme for editing (Illustration 12, 13, 14, & 15). It's always in edit mode, with a color leading the way through drill-downs to editable fields, to avoid the frustration of drilling down just to find that something isn't editable to you. When you change a field, its background color changes to a darker editable color (as Dr. Suthers suggested), and a "save" button appears (as Brian suggested, instead of "OK"). There's no "cancel" button anymore, but "undo" is implemented with a grace period, to reconcile the challenge of events that cannot be undone, such as sending a notification email.

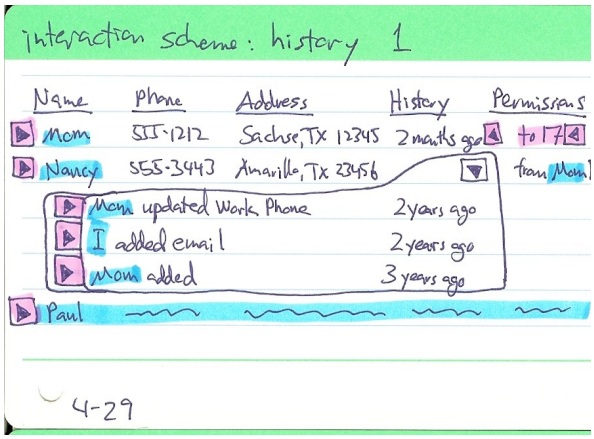

Illustration 10: Interaction scheme - history 1.

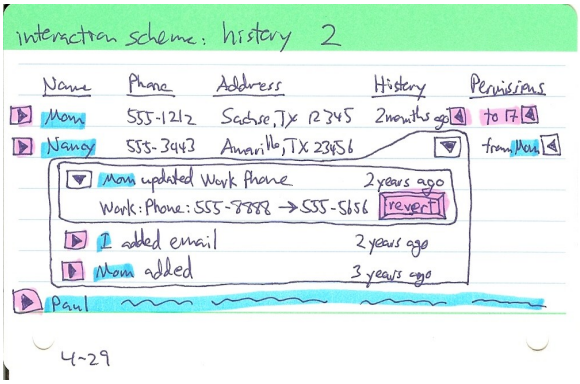

Illustration 11: Interaction scheme - history 2.

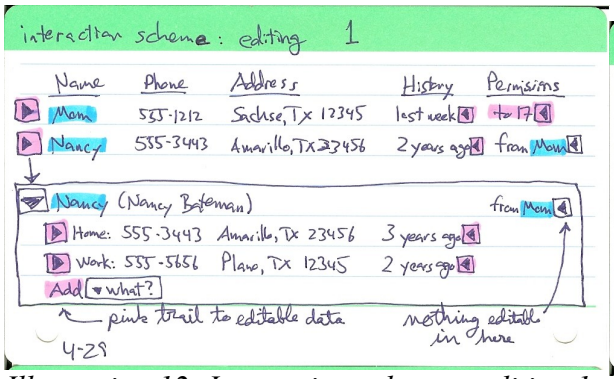

Illustration 12: Interaction scheme - editing 1.

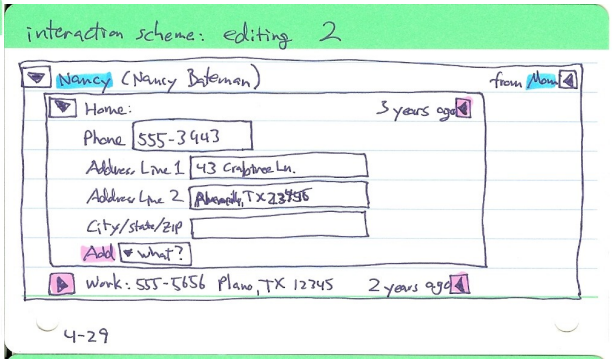

Illustration 13: Interaction scheme - editing 2.

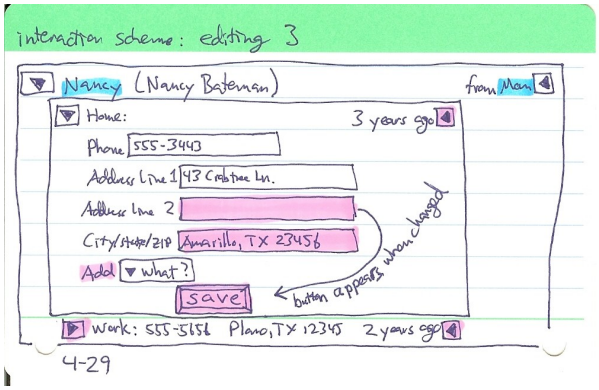

Illustration 14: Interaction scheme - editing 3.

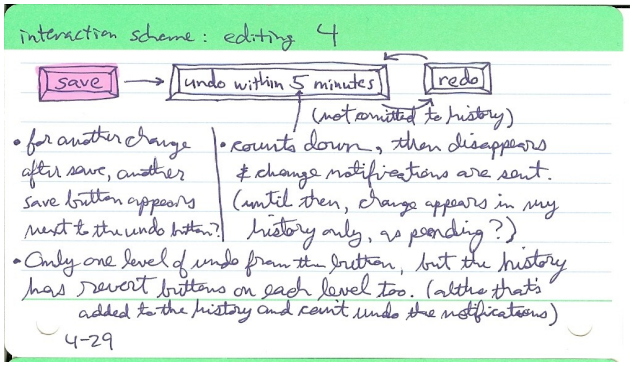

Illustration 15: Interaction scheme - editing 4.

Analysis

This Agile UCD method was useful to me. While it's hard to predict the user experience based on static paper prototypes, there are some problems that they can certainly reveal. Conducting a collaborative usability inspection and revising an index card is much faster than developing code for that user interface and conducting a usability test. By thinking a few steps ahead at this higher level, I realized new parts of the interface and corrected related areas before wasting time on a bad implementation.

If I had started this application with just an agile method, no UCD, I would have been reinventing chunks of the interface from scratch on every task. I can picture a lot of the user interface now that I could not at the start of this project. On the other hand, if I had used the traditional UCD, I would not have gotten as far with the design as I have in the last 10 weeks. Agile UCD is a good way to start a new project. I expect to use it again in the future.

Further research

Although the Agile UCD method has proven useful so far, several questions remain. I need to proceed to agile development, with implementation iterations and further Agile UCD iterations, to see:

- How useful or complete are the current design artifacts for the first implementation iteration?
- How well does the scheme anticipate needs of tasks selected in the future for implementation?
- How flexible or wasteful are the artifacts and methods for unanticipated future tasks or requirement changes?
- What are the usability testing results on a working prototype or implementation?
- What kind of user acceptance and feedback comes from a deployment?

References

Alexander, C. (1979). *The Timeless Way of Building.* New York: Oxford University Press.

Beck, K. et al. (2001). "Manifesto for Agile Software Development," retrieved May 4, 2009 from http://www.agilemanifesto.org/.

Constantine, L. (2000). "Cutting Corners: Shortcuts in Model-Driven Web Design," experience report [January 2000, rev. February 2002], reprinted as a column in The Management Forum, *Software Development,* February 2000, reprinted in L. Constantine (ed.), *Beyond Chaos: The Expert Edge in Managing Software Development.* Boston: Addison-Wesley, 2001. Retrieved February 19, 2009 from http://foruse.com/articles/shortcuts.htm.

Constantine, L. (2002). "Process Agility and Software Usability: Toward Lightweight Usage-Centered Design," reprinted from *Information Age,* August/September, 2002, revised and expanded version of a column from The Management Forum, *Software Development, 9,* (6), June 2001, retrieved February 19, 2009, from http://foruse.com/articles/agiledesign.htm.

Constantine, L. & Lockwood, L. (1999). *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design.* New York: Addison-Wesley.

Constantine, L. & Lockwood, L. (2000). "Structure and Style in Use Cases for User Interface Design," preprint [February 2000, rev. November 2000] published in M. van Harmelen (ed.), O*bject-Modeling and User Interface Design,* Addison-Wesley, 2001, retrieved February 19, 2009 from

http://foruse.com/articles/structurestyle2.htm.

Constantine, L. & Lockwood, L. (2002). "Usage-Centered Engineering for Web Applications," Unabridged original draft of an article accepted in condensed form for publication in *IEEE Software, 19* (2), March/April 2002, retrieved February 19, 2009, from http://foruse.com/articles/webapplications.htm.

Cummings, J. (Oct. 2007). "Word 2007: Not Exactly a Must-Have," *Redmond Magazine,* retrieved May 11, 2009, from http://redmondmag.com/features/article.asp?editorialsid=2346.

Fowler, M. (2005). "The New Methodology," earlier abridged version published in *Software Development, 8* (12), December, 2000 under the title of "Put Your Process on a Diet," retrieved May 4, 2009, from http://martinfowler.com/articles/newMethodology.html.

Gaudin, S. (March 25, 2009). "Facebook caves in to users after revolt over redesign," *Computerworld,* retrieved May 11, 2009, from http://www.computerworld.com/action/article.do?command=viewArticleBasic&articleId=9130467.

Kerievsky, J. (2005). *Refactoring to Patterns.* Boston: Addison-Wesley.

Krippendorff, K. (2006). *The semantic turn: a new foundation for design.* Boca Raton: Taylor & Francis.

Lockwood, L. (1999). "Collaborative Usability Inspecting," presented at Software Development East, Washington, DC, November 1999. Retrieved May 15, 2009, from http://www.foruse.com/presentations/inspections.htm.