CSC 484 Lecture Notes Week 4, Part 1

Understanding and Conceptualizing Interaction

I. **Relevant Reading** -- chapter 2 of the book.

II. **Introduction to Chapter 2 (Section 2.1).**

A. Speaking on behalf of software engineers, I think this chapter displays a pretty significant misunderstanding of what modern software engineering is about.

B. The introductory question/answer sequence sets up a would-be dichotomy that does not really exist.

1. They start by posing the (paraphrased) questions,

   "In designing a new application, would you start by coding? Or, would you start by talking to users and seeing what else is out there?"

2. Their answer

   "Interaction designers would do the latter."

   begs the question

   "Who wouldn't?"

3. A well-trained software engineer, even the most extreme of the extreme programmers, would answer the questions the same way as the ID folks answer.

4. I.e., they'd start by talking to users and seeing what else is out there.

C. This introduction sets the stage for a largely false dichotomy between what the authors tend to see as the following actors:

1. the rather clueless software engineers, versus
2. the inspired interaction designers,

the latter seeing it as their mission to enlighten the former on the importance of involving users in a design.

D. In my experience, software engineers really aren't as clueless as they're made out to be.

1. There have been more than thirty years of research in software requirements and modeling.
2. This research addresses many of the same issues discussed in this chapter.
3. The research has many good ideas on how effectively to solve problems, and to be creative doing it.

E. With particular regard to Agile development, the following statement on Page 44 is antithetical:

   "... Once ideas are committed to code, they become much harder to throw away".

1. Agile developers say precisely the opposite, since their methodology revolves around writing small increments of code, which are entirely disposable if necessary.
2. Even for fully traditional SE, the idea of "committed to code" does not apply to throw-away prototypes, since the very name suggests they are designed to be thrown away.

F. All that having been said, the chapter does provide some useful information; here's my list of hits and misses:

1. ***Hits:***

   a. Provoking thought, by challenging the assumptions and beliefs of engineers.

   b. The importance of understanding the problem space, in Section 2.2.

   c. Analysis of the *interaction types*, in Section 2.3.4.

   d. The interview with Terry Winograd, at the end.

      i. Note that he talks a lot about of "people", "products", and "examples".

      ii. He never mentions "conceptual models", "metaphors", or "analogies".

2. ***Misses:***
   a. The misunderstanding of SE.
   b. The maltreatment of conceptual modeling -- high on aphorism, deficient on useful suggestions.

III. **Understanding the problem space (Section 2.2).**

A. What you should take away from this section are two things:
   1. The importance of having a problem to solve.
   2. The notion of identifying and challenging your design assumptions.

B. The engineer's common refrain -- *"What's the problem here?"*
   1. You ask this whether you're improving on an existing product, or coming up with a brand new idea.
   2. The bulleted list at the end of the section summarizes some useful questions in this regard.
      a. What problems are you trying to solve?
      b. Why do these problems exist?
      c. How is your new design going to solve the problems?
      d. If you've not identified any specific problems[1], but are designing for a brand new user experience, how do your ideas make things better than the current way of doing things?

C. As part of your work on Assignment 2, do the following in a team meeting:
   1. Answer each of the questions above, i.e., what's your problem, how are you going to make things better?
   2. Identify the assumptions you're making, and how you're going to validate them.
      a. For example, you may identify some underlying assumption about which you weren't specifically cognizant, and the team may determine on the spot that the assumption is not valid; this means you should not base any of your ideas on it.
      b. Alternatively, you may identify an assumption you think is true, but you cannot validate immediately; in this case, come up with some ways to do validation, such as asking users appropriate questions.

IV. **Conceptualizing the design space (Section 2.3).**

A. The authors need to get straight what we're trying to do. Is it
   1. trying to solve a particular problem, or
   2. trying to build a product that will appeal to the largest possible user community, or
   3. trying to invent some yet undreamed of new form of interaction

B. If it's the first of these, then the conceptual model is based on a specific understanding of real and concrete user problems.

C. If the second, then the conceptual model is based on a generalized understanding of potential users.

D. If it's the last, then the conceptual model is whatever a design team might dream of.

V. **Conceptual models, and examples thereof (Sections 2.3.1 and 2.3.2).**

A. To start with, the 1996 Liddle quote (on Page 51) is seriously out of date:

> *"The most important thing to design is the user's conceptual model. ... That is almost exactly the opposite of how most software is designed."*

[1] The engineer would say, *"If you don't have a specific problem, fuhgeddaboudit."* OK, so engineers do have their limitations.

B. In modern (post-1996) SE, conceptual models are an important part of even mundane software products.

   1. They start with introductory sections of the requirements document, or in a stand-alone document such as a "Vision and Scope".

   2. Models then pervade the rest of the development, including formalizing the requirements, specifying software behavior, and defining the program architecture.

C. A major question that the book does not address is the following:

   Is the conceptual model a specific, concrete artifact of the ID process, or is it embodied in other process artifact(s)?

   1. If the latter, what exactly does it look like?

   2. In what language or notation is it stated?

   3. The best the book can come up with is to suggest that it's expressed in the "*lingua franca* used by the design team".

D. What the book fails to recognize is how that "lingua franca" of the conceptual design is expressed specifically in the *the storyboards and scenarios*.

   1. If you believe in affordance, a concrete user interface should fully *embody and convey* the conceptual design.

   2. It does so to the end users, as well as to design team.

   3. If a concrete interface is not so affordant, i.e., if it fails to make clear the underlying conceptual model, then

      a. it's a bad interface, or

      b. it's a bad concept

   4. Either way, the interface itself is the artifact that represents the underlying conceptual model.

E. Direct evidence for this is provided book by authors themselves, in Section 2.3.2.

   1. What is the "lingua franca" they choose to convey the conceptual models of the "best practice" examples?

   2. It's pictures of concrete user interfaces!

F. So -- a well-designed user interface *fully affords*, and hence defines the underlying conceptual design.

   1. The *metaphors* and *analogies* should be readily apparent in the interface layout

      a. For example, an electronic spreadsheet looks like the paper ledger it models.

      b. The Xerox Star UI looks like a desktop.

         i. The screen elements are familiar items from a non-electronic desktop, such as paper documents.

         ii. There are desk accessories, such as clocks and calendars, easily recognizable as such.

   2. The conceptual model *lexicon* is comprised of the words used skillfully in the display of information.

G. Other aspects of a conceptual model are conveyed most effectively by an operational prototype.

   1. The Xerox Star concept of drag-and-drop is very difficult to convey in any form other than prototypical demonstration.

   2. The direct manipulation behavior of the Star interface far better illustrated by example than any form of ontology.

H. None of the preceding discussion means that early versions of a concrete interface need look specifically like the final product.

   1. Winograd and others say that designers should not be tied to specific forms of interface "widgets".

   2. E.g., don't get stuck in a conventional menu-based UI, when designing something that should have a natural language interface.

   3. Through an evolutionary process, the conceptual model is conveyed in a succession of interfaces, from

storyboard sketches, to illustrated scenarios, to interface prototypes.

I. As a thought experiment -- consider how Steven Spielberg presents his conceptual model for a new picture.
   1. Is it with some storyboards and a plot outline?
   2. Or is it with some tome about metaphor and analogies?
   3. From what I've read, it's the storyboards
   4. He leaves it to the movie critics to wax idiotic about metaphor.

J. Or, consider how Frank Lloyd Wright conveyed his conceputal model for a building.
   1. He did it with sketches of the building and its surroundings.
   2. It was the civil engineers who did the modeling.

VI. **Some specific comments on the Johnson and Henderson paper that forms the basis of Section 2.3.1.**

A. Henderson and Johnson profoundly misunderstand conceptual modeling, in two fundamental ways:
   1. the role played by software engineers and others in the development of conceptual modeling principles;
   2. the way that concrete examples *define and convey* a conceptual model via affordance.

B. Regarding the first misunderstanding, the authors make the following statement, in the introduction:

   "... our experience with our clients indicates that conceptual models of this sort are almost completely unknown outside of the HCI community, especially among Web designers and software programmers."

   1. I respectfully submit that their experience is bogus.
   2. Conceptual models "of this sort" have been the subject of SE research for well over 30 years; to whit
      a. The use of metaphors and analogies are part of what SEs have long called *domain analysis*.
      b. Representing concepts as objects and operations dates back as far as the early 1970s, in languages like PSL and SADT, and extends straight through to UML 2.0 today.
      c. Relationships and mappings were also fundamental parts of PSL and SADT, as well as ER diagrams.
      d. Artificial intelligence people have been working on similar concepts and notations for ontologies, for just as long.
   3. Johnson and Henderson may find the SE notations stodgy and too "engineered", but they offer absolutely nothing as a constructive alternative.
   4. In the area of modeling, the Johnson and Henderson have more to learn from software engineers, than the other way around.

C. On the second misunderstanding, they consistently miss the point about the power of concrete examples.
   1. Consider one of the first "concepts" they site -- whether to represent data as a flat list or hierarchy.
   2. If this is a fundamental and high-level concept, then it should be immediately obvious by looking at some aspect of the high-level user interface, not in some other "lingua franca" of the designers.

D. In the beginning of the paper, the authors say that "How the system *presents itself* to users" is does not convey a conceptual design.
   1. If it's does not, then the users have been left out, and by the authors' own admission, they cannot be.
   2. Further, if the authors want to say that a storyboard is not in fact a means to convey a conceptual design, then they should give some other specific way to do so.
   3. They do not do so, nor do I think they can.

VII. **Interface metaphors and analogies.**

A. They're fine, however:
   1. Choose ones that are understandable and compelling to users.

2. Don’t over do it.

B. In the book, the part on "opposition to using metaphors" is longer than the other parts; they got that right.

VIII. **Interaction types (Section 2.3.4).**

A. This is some useful information.

B. They present four specific types:

1. ***Instructing*** -- users instruct, i.e, command the system to do things.
2. ***Conversing*** -- users have a two-way dialog with the system.
3. ***Manipulating*** -- users open, close, move, edit data provided by the system.
4. ***Exploring*** -- users move through a large space or virtual environment.

C. These types are definitely not mutually exclusive.

1. An interactive system can, and often does provide more than one of these types of interaction.
2. Allowing users to seamlessly progress among the different types is an important part of a well-integrated user experience.

IX. **Instructing Interfaces (Pages 65-67)**

A. Iconic UIs are generally easier to use and less error prone than command-language.

B. Good reasons to use a command-language UI:

1. The number of instructions is too large to map well to icons.
2. The interface needs to be scriptable, e.g., users can write scripts to perform repetitive tasks

C. Questionable reasons to use a command language UI:

1. It’s easier to implement, e.g.,
   a. a small vending machine keypad, instead of a large back-lit iconic keypad
   b. a program with a simple text UI, instead of a more complicated GUI
2. It’s easer to maintain, e.g.,
   a. it’s easier to re-map the vending machine code "B2" to a different product;
   b. it’s easier to maintain a non-GUI program when it’s deployed on multiple platforms.

D. Here "questionable" means that the UI itself is not optimal, but there may be other trade-offs involved that lead to the selection of a sub-optimal UI.

X. **Conversing Interface (Pages 67-70).**

A. These UIs involve a two-way conversation, and assume therefore that the system has enough knowledge to communicate effectively with the human.

B. Good reasons to use a conversing UI.

1. The user has little or no knowledge of the available commands.
2. The system has enough data and intelligence to provide effective answers to general questions.
3. "Intelligent" agents are both good and bad examples, depending on how able the agent is to answer a particular user’s questions effectively.

C. Bad reasons to use a conversing UI:

1. It’s cheap, e.g., it’s cheaper to have an automated phone menu than a person or AI system give an answer.
2. It looks cute and *seems* intelligent, e.g., numerous online examples that provide UIs that appear to understand some form of natural language, but actually do not.

XI. **Manipulating Interfaces (Pages 70-75).**

A. These UIs involve manipulating "real-world" representations of objects, performing operations with actions that represent real-world manipulations. E.g.,

1. dragging a file icon into a trash can
2. commanding a robotic device using a joystick

B. The term *direct manipulation* refers to the form of interaction that is now ubiquitous in window-based computer UIs.

C. Good reasons to use a direct manipulation UI.

1. An action can more efficiently or effectively be performed, e.g., drawing a graphic shape with a mouse, rather than typing in it's geometric coordinates.
2. A direct-manipulation UI can be easier to learn.

D. Reasons not to use a direct manipulation UI.

1. It takes (substantially) longer than a simple command, e.g., manual search-and-replaces versus a "change-all" command.
2. A sufficiently expert user community can be more productive with a command-language UI.

XII. **Exploring Interfaces (Pages 75-83).**

A. These include virtual environments, or physical context-aware environments.

B. These are relatively new forms of interaction, about which designers still have much to learn.

C. Virtual UIs have been effective in a number of areas, including games, architectural exploration, and larger-scale exploration of geographic regions.

D. They have yet to take off in other areas, e.g., "smart homes".

XIII. **Theories, models, and frameworks (Section 2.4).**

A. A ***theory*** is a high level explanation of human-computer interaction, based in particular on theories of human behavior and cognition.

B. A ***model*** is an abstraction of human-computer interaction, typically of a specific aspect, designed to provide the basis for design and evaluation.

C. A ***framework*** is a prescriptive set of principles and organizational guidelines, designed to provide a broader view of how to approach design and evaluation.

D. Subsequent book chapters cover these subjects in further detail.