Introductory Comments

Interactive behavior arises from the people using devices to accomplish tasks. How do you determine if an interface is a “good” interface? An interface can only be judged based on how the interaction among the human, task, and device unfold over a sequence of events.

No matter how devices and interfaces evolve, you are still locked into a cycle of establishing a goal, breaking it into subgoals, hypothesizing an action plan to meet those goals and subgoals, issuing commands to the computer, waiting for the computer to respond, deciding whether the response takes you closer to a subgoal, and determining when you have met your top-level goal.

In this class, you should learn a vocabulary, concepts, and methods for building useful and usable human-computer interfaces.
Possible Examples to Prepare:
“Is this a good interface?” slide show (on iPad).
“PayPal is Rude in the Lunch Line” (1-page printed out, to read to students.)
CHI 2008 Presentation.

Translation of SBD-Specific Vocabulary Words

The textbook does an excellent job illustrating a classic user-centered design (UCD) process but recasts and renames it as “Scenario-Based Design” (SBD) in an attempt to formalize UCD. Formalizing processes is a good thing to do. It is necessary to convert science into engineering, which is a good thing to do. Hence the title of the book: Usability Engineering.

Nearly all of the terms introduced in the book, such as in boldface, are important standard terms used in human-computer interaction (HCI) research and practice, and in user interface (UI) design, and will help you to understand and be understood by HCI researchers and practitioners. There are a small number of terms, however, that are specific to SBD and, even though they represent important and usually well-established concepts, the specific terms will not be understood by most HCI and UI researchers and practitioners. The list below is an attempt to translate the SBD-specific words to more generally-accepted terminology, or the plain words that be more commonly used.

<table>
<thead>
<tr>
<th>SBD term</th>
<th>UCD term</th>
</tr>
</thead>
<tbody>
<tr>
<td>scenario-based design</td>
<td>≈ user-centered design</td>
</tr>
<tr>
<td>root concept</td>
<td>≈ a statement of the project vision and rationale, stakeholders, and starting assumptions (p.49)</td>
</tr>
<tr>
<td>scenario</td>
<td>≈ a story about people carrying out an activity</td>
</tr>
<tr>
<td>problem scenario</td>
<td>≈ a scenario prior to the introduction of a new system</td>
</tr>
<tr>
<td>activity scenario</td>
<td>≈ how the current use case could change with the introduction of a new system</td>
</tr>
<tr>
<td>claims</td>
<td>≈ statements of how different aspects of a system will impact users’ experiences</td>
</tr>
<tr>
<td>claims analysis</td>
<td>≈ a consideration of the pros and cons of design alternatives</td>
</tr>
<tr>
<td>metaphors</td>
<td>≈ one of many possible ways to creatively explore new ideas</td>
</tr>
</tbody>
</table>
Chapter 1 - Scenario-Based Usability Engineering

Scenario-based design is an approach to developing software systems that are easier to use, easier to learn, and more satisfying to users.

**Scenario-Based Design**

The chapter starts with an example scenario: Marissa’s gravity question.

A *scenario* is simply a story about people carrying out an activity (p.2).

A *problem scenario* is a story about a problem domain as it exists prior to the introduction of new digital technology.

A *design scenario* describes a new vision for how a human need might be addressed. Note how it is abstract. It does not discuss any details of how any technology might work.

The design scenario is not the system that you build. Exploration is critical.

**Managing Software Development**

The process of software development (the full life cycle) is difficult to see but it is easy to modify, sometimes for the better and sometimes for the worse. (Example: Deciding to defer requirements analysis.)

**No Silver Bullet** (p.6)

Software development is a very complex and wicked problem and there is “no silver bullet”, no magical solution that will change that fact. (Fred Brooks, 1987)

There is no single innovation that will make software development costs drop in the same way that hardware costs will continue to drop.

Software is complex in part because it is so complicated and yet so flexible. It gets conformed and modified at the expense of undermining its own conceptual integrity.

One of the keys to solving the problem—though not a silver bullet—is iterative development with prototyping.

**Software Engineering**

The goal is to gain and maintain control of the problem, both the process (by following a life cycle model) and the product (with established approaches to organizing problems, such as requirements documents and modular design).
**Waterfall model** (p.5) - an early and influential model for software development that is organized into a series of modular phases, including analysis, design, implementation, testing, and maintenance.

The waterfall model (approximately):

- Requirements Analysis
- Design
- Implementation
- Testing
- Maintenance
- Project Management

**Tradeoffs in Design**

**Tradeoffs** (p.7) - alternatives, and the benefits and problems associated with each alternative. Tradeoffs exist throughout a design process. For example: nail down a design, or keep your options open.

In design, there is almost never a single correct answer. (This is a truism, a statement that is so true that it basically says nothing new.) Some solutions are better than others in different ways, but there are always tradeoffs. (Simplicity versus more features and functionality, iPhone versus Android, electronic calendar versus paper calendar).

A design rationale is a written document that captures the decision-making process, and the tradeoffs, and documents why ideas are accepted or rejected.

There are tradeoffs not only in products but in processes. A waterfall model focuses on getting things done, whereas a prototyping approach focuses on revisiting ideas.

**Prototyping**

A **prototype** is a concrete but partial implementation of a system design (p.198). A prototype converts abstract ideas into concrete examples. In system design, they must somehow capture the dynamic aspect of the system.

Prototype (p.8) - an operational model to demonstrate a design idea.

**Prototyping** can be incorporated into a software development process lifecycle, perhaps best to contribute to the requirements analysis. (p.8)

**Iterative development** focuses on looping through

requirements analysis -> design -> prototyping -> evaluating -> (loop)

as a means of developing and refining a system design.

Iterative Development (p.9) - A process in which designs and design documents are produced as an output of each phase in the design process, but continually modified through prototyping and testing.
Usability in Software Development

Usability is a measure of a system in terms of ease of use, ease of learning, and user satisfaction. (It is also a “software quality” such as reliability, portability, maintainability, security,...)

Usability can be addressed throughout the process lifecycle, but most importantly at the beginning and the end.

(skipping pp. 9-14, though you should read them)

Usability Engineering

Usability engineering is a somewhat formalized approach to planning, achieving, and verifying the usability of a system, in which usability refers to ease of use, ease of learning, and user satisfaction. The key idea is that measurable usability goals can be defined early in the process, and pursued and assessed during the process.

Since its inception the mid-1980s, usability engineering focused on scenarios, though initially the stories related more to specific task instances, in which the classic human performance measures of speed and accuracy were measured, along with user attitudes. Designers focused on display features and commands. For example, in a command-line text editor, which commands are easiest to learn and use. (sort of like vi, pico, emacs)

Scenario-Based Usability Engineering

Computers do more than just provide information and services to people. They create new opportunities for human activity, and change the way we live. The basic idea of scenario-based usability engineering is that descriptions of people using technology are essential in discussing and analyzing how the technology is (or could be) reshaping these human activities. (pp. 15-16)

A user interaction scenario is a story about people and activities.

(skipping pp. 17-23)

Participatory design (p.23) - design work that takes place as a collaboration between developers and the people who will use the system. Very popular in Scandinavian countries. Used by architects, too.

Activity scenarios (p.26) - narratives of typical or critical services that people will seek from the system, deliberately focusing on functionality, refraining from specific details of how the system will work.
Chapter 2 - Analyzing Requirements

Requirements Analysis
The phase of software development in which the needs of clients with respect to a proposed project or technology are analyzed. (p.37)

Analyzing Work Practice
Activities - The personal goals, activities to achieve them.
Artifacts - The physical objects used to carry out the activities.
Social context - The people their relationships, motivations.

Hierarchical task analysis is an approach to studying activities. Individual tasks (pieces of work) and subtasks are organized into a hierarchy, a tree-like structure. (p.39) HTA helps to transform a complex activity into increasingly easier-to-understand steps, so the entire task can be better understood.

But HTA often does not capture the emotional aspects of a task or experience, the important social relationships, the motivations, the psychological experiences.

Ethnography is an analytical technique derived from anthropology in which the analyst becomes intensely involved in a group’s activities as a participant/observer, and collects data about that group, to understand it richly and deeply from the inside out. A debate: Can the ethnographer start with a hypothesis, or must the hypotheses be data-driven? Are they scientists?

Getting Users Involved (p.43)
Different users have different levels of expertise.
The manager often does not know how the employees on the front line really do their job.
(Mom’s Tektronix story.)
You must talk to many different stakeholders.

Observed on the bus: One person in a wheelchair says to people on a seat “There’s another wheelchair coming on.... I raised the seat. I offered to help the guy strap in. He said no. The first guy greets the second guy. Who are all the stakeholders there? I’ve wheeled chairs onto Max in the past. It was mentally exhausting, trying to think about all the social relationships.

Tacit Knowledge - What is happening that you cannot see? The people in an organization, or community, or common circumstance have some common understandings. How do you as an outsider dig into that tacit knowledge? Did those two guys in wheelchairs know each other? How can you find out?

I used to deejay at clubs in NYC, ages ago. Sometimes when I go and visit NYC I want to get a sense of what the scene is like now. It is totally different. One summer (2010), I spent about two hours in a store called Turntable Labs, listening to music but just as much listening to the
conversations that customers had with the staff. Such as where people are working, what the crowd is like there.... One guy said how much he liked deejaying at a crummy bar where kids from Long Island (looked down on) just get drunk and have a good time, disparaging a seemingly cooler crowd that analyzes every song that you play. The summer of 2008, I was in Berlin on vacation and I went to a bunch of clubs. A question formed: Why do all of the deejays in Berlin still use vinyl? I asked this question to a lot of people but got no good answers until I talked to a record store owner who explained that there are basically three reasons: (a) as a tribute to the past (“Vinyl is the mother. You don’t kill your mother.”), (b) ergonomics (“you can just look at the vinyl and you are done.”), (c) socio-economic reasons (government-controlled rent increases make it possible for small record stores to survive).

You can get to the tacit knowledge if you dig and probe, and if you are persistent and patient.

Contextual Inquiry: A requirements analysis method in which people are observed in their normal task environment, and the observer is permitted to interrupt the work with questions. I probably learned a lot about deejaying by conducting some of these analyses with famous deejays such as Frankie Knuckles, Larry T, Justin Strauss, and Dmitri of Delite. It is an apprenticeship. In olden times, you learned by apprenticing yourself to a master.

**Preparing for a Field Study**

Prepare for Interviews

It is important to generate a number of specific questions that you will ask. (Page 52 has an example.) But depending on the circumstance, you might want to start with one kind of question or another.

Such as start with open-ended questions, or easy questions. Then get into the harder or more personal questions.

Try to not ask leading question, such as “Were you annoyed that you had to give up your seat?” but rather “How did you feel...?”

Perhaps one person asks while the other takes notes. Videotaping and tape recording can be useful, but it can be a little intimidating for people, and it can be very time-consuming for you to transcribe it later.

**In-Class Exercise**

What are some specific activities that you can partake in to learn more about the context of use of people going to church for fellowship and community, and the experience of all people involved in people in wheelchairs riding on public transportation?

What are some specific sets of questions that you can ask, and contexts for asking them?
You are not designing computer programs. You are designing human activities. You are not designing a little app that shows a video of a Bic lighter burning. You are designing the activity of many people in a crowd firing up that app and showing it to the band. And you are also encouraging the delving into the virtual space while trying to help people show appreciation for a “live” event.

“Activity Design” “emphasizes the broad scope of what is being designed: people carrying out activities with the support of computer software. It is essential to design software systems in a usage context, always considering whether and how they will support human goals and activities.

What makes a user interface effective? (p.81) Or good? Or useful? One measure is efficiency, but just because the easiest and most efficient way to get students to fill out teaching evaluations is to withhold their grades if they do not fill them out, does not mean that this is a good user interface.

Being able to dial “0” from any campus phone, and say the name of a faculty, and get connected, is efficient but it only works about 67% of the time.

There are plusses and minuses to every idea for a new activity. Hence you do a cost/benefit analysis, or a “claims analysis”.

**Claims Analysis** - An analytic evaluation method that identifies the positive and negative consequences to a design. (from glossary on p.374)

Claims analysis plays an important role in SBD. See p.72.

Good examples throughout book, such as page 73.

**Designing Comprehensible Activities** (p.84)
The user has a "mental model," a set of expectations about how to do a task, and how to use use a system to do that task. This phrase is quite common in interface design, but it means slightly different things to different people, so you should define it when you use it. The user's mental model is not going to be the same as the designer's mental model, which the book calls the "designer's model," though this phrase is less commonly used in interface design.

The key idea is that the user's and designer's mental model are very rarely the same. The designer tries to create the user's mental model but it is very difficult to do, and difficult to know if you have succeeded.

Example: Current users have a mental model of a physical store that does not include finding post-it notes on shelves saying what previous customers think of the products on the shelves, but they have a mental model in which they do expect to find such notes posted on online stores.
**Metaphors** are analogies that illustrate one thing by relating it to a second different that is not typically related to the first thing, such as "love is a rose". Analogies provide a framework for creative exploration. Examples: Customer comments on online stores is like putting post-it notes on shelves at stores. A library is like a warehouse, or like a café (or a big office).

There are good examples of metaphors in the book on p.93-94.

**Designing Satisfying Activities** (p.89) How do you motivate people to do their work, and to help them to enjoy their job? Example in book: Librarians want to help people. Automating library systems might actually make it **harder** for librarians to have special knowledge and abilities to help others. You must examine the tradeoffs between “automation and personal control.” Designers must carefully consider the implications of redesigning a business practice. Example: Requiring students to fill out course evals in order to see their grades.

Using Personal Social Networks to Tailor News to Family and Friends
Can't Attend? Send Your Robot Instead - Video Library - The New York Times
The Kindle 3 - Video Library - The New York Times
Future example: Compare the iPad commercial to my iPad software purchase experience.

**Chapter 4 - Information Design**

“During information design, the objects and actions possible in a system are represented and arranged in a way that facilitates perception and understanding.” (p.109)
This information can be visual, auditory, or tactile.

Some terminology.
**Gestalt principles** -- characteristics of human visual information processing that define how individual components get grouped together to create a figure. Examples: Grouping, continuation, shapes.

**White space** is a very good way to create visual structure. The gaze does not land on it, but perceptual grouping can be perceived from across the display.
The **squint test** can be used to check interface groupings.
Realism versus abstract symbols.
**Consistency** permits the **transfer of learning**.
A **visual design program** or **system** can contribute to consistent displays.
**Internal consistency** and **external consistency**.
**Visual metaphors**, such as a calculator. (Though they can break down because no divide-by or multiply-by key on the keyboard).

The **plan-execute-perceive cycle of human vision**
The plan-execute-perceive cycle of human-computer interaction is presented in the next chapter. The same cycle, just on a faster time scale, is followed for moving the eyes around a visual scene to perceive visual information.
A plan is formed and updated in real time about how to interact with a visual scene to reach a goal. The plan draws uses prior knowledge about the scene or similar scenes (and thus expectations), from visual information in the periphery, and from visual information gathered during fixations.

Why do the eyeballs of a human rotate in their sockets?
See the slides for “Why the eyes rotate” on the course website.
The eyeballs rotate so that a person can orient the small region of high-resolution (or high acuity, but not "high focus") vision towards objects in the visual scene that a person wants to examine more closely. This region of high-resolution vision is roughly 2° of visual angle, roughly the size of your thumbnail at arm's length. (It corresponds to the foveal region on the retina, where the "rod" nerve cells are most dense.)
The eyes typically move with quick jumps called saccades that last roughly 20-40 ms, and then hold steady at locations with fixations that last typically 250-500 ms. If there is a moving object that the eyes can follow, then smooth pursuit eye movements can be made.

During each fixation, the saccade to the next fixation is planned, and then it is executed. For example: Turn on the lights, or open a window, or find an individual in the classroom.

There is a close relationship between good visual design guidelines (such as Proximity, Alignment, Repetition, and Contrast) and these basic characteristics of human vision. In short, a well-designed visual layout permits a person to see the structure and organization of the layout in the near periphery (similar to what you will see in a squint test) and use this structure and organization to determine appropriate destinations to saccade to, such that the high resolution vision is always placed on relevant information.

Information design in auditory displays
Perceiving information (data) is almost always an interactive process. Contemporary auditory interfaces do not adequately consider this.
Chapter 5 - Interaction Design

*Information* design focused on figuring out what task objects and actions to show, and how to represent them. The goal of *interaction* design is to specify the mechanisms for accessing and manipulating task information.  
(Don Norman’s example of a wall of doors with identical handles.) Interaction design tries to make sure that people can do the right things at the right time. The interaction design that you build into a system will determine the activities that your users can engage in. The human-computer interaction cycle: Establish a human goal, translate it into a system goal, develop an action plan, execute the plan, perceive the results of the execution, interpret the results, and decide whether the goal has been accomplished.

**The plan-execute-perceive cycle of human-computer interaction**

![Diagram of the plan-execute-perceive cycle](image)

Interaction design relates to how easy it is for a user to (a) translate his or her goals into the procedures for using a system to accomplish those goals, (b) carry out those procedures, and (c) determine that he or she is making progress towards his or her goals. (Example: Installing Keynote on iPad.)

### 5.1 Selecting a System Goal

People approach a computer with a human goal. They translate it into a *system goal* and determine and execute an appropriate *task strategy* to accomplish the human goal using the system.
An **affordance** refers to perceivable characteristics of an object that helps a person to know (not “that makes it obvious” as the book defines) what the object can do, and how it can be manipulated. It relates to “stimulus-response compatibility”, which is a measure of the speed and accuracy with which a person can learn, execute, and retain knowledge of the mappings between stimuli and responses. Such as, the mappings between four lights and four buttons.

When one of these lights turn on: \[1 \rightarrow 2 \rightarrow 3 \rightarrow 4\]

Press the button it is mapped to: \[J \rightarrow K \rightarrow L \rightarrow ;\]

**Stimulus-response compatible mappings:** 1J 2K 3L 4;

**Stimulus-response incompatible mappings:** 1L 2J 3; 4K

The **gulf of execution** refers to the difficulty that people have in determining the physical actions needed to accomplish a task with an interface. The **gulf of evaluation** refers to the difficulty that people have in determining whether they are making progress towards those goals after executing an action. (Neither is the “psychological distance” of anything as the book states because there is no such measure.) These two “gulf of” terms are not actually used very often, but they are terms from an 1980s book popularizing human factors (Norman’s *POET* book). And the fundamental concepts are very important in UI design and analysis (but I prefer plain words).

**Direct manipulation** is thought to make computers easy to use by introducing graphical user interfaces (GUIs) rather than command-line interfaces because GUIs perhaps reduce the gulf of execution by making screen objects look and sort-of behave like things in the world. And because it makes it difficult for programmers to get away with assigning radically different functions to the same actions. Though they sometimes do, such as how dragging a file or a folder to the trash deletes it, but dragging a floppy disk to the trash ejects it.

Direct manipulation started with WIMP interfaces: Windows, Icons, Menus, Pointers. Touchscreen displays, such as with tablets and smartphones, take direct manipulation to a greater extreme. But all kinds of inconsistencies are introduced. For example, what is “clickable” still needs to be made very clear, and often is not. Direct manipulation is not a magical way to make interfaces easier to use. For example, on a touchscreen, there is no “right click” to see a number of potential commands for an object. And you cannot rest your finger on a button while deciding whether to press it, or touch type. And it introduces many, many modes.

5.2 **Planning an action sequence:** People develop and execute task strategies. When interacting with computers, these typically include perceptual and motor. They can also be purely cognitive. They can be planned ahead, prepared. So consistency matters a lot, because they permit a user to plan a few steps in advance based on how they expect the functionality to be accessed, and how the computer will behave. (Such as, when you encounter a couple fields that say “username” and “password,” to be able to type your username, tab, your password, and enter. This was not the case on DuckWeb a few years ago.)
Action sequences, or cognitive strategies, are planned and executed on the micro level (tasks that last a few seconds, such as above) as well as the macro level (tasks that last minutes, such as connecting to a network and sending a print job to a printer).

The UI designer’s challenge is to support the user at every step in their action plan, and to make it clear to them what functionality is available so that the users can map that functionality to their tasks and goals. Such as, if a user wants to print double-sided, make it clear whether that functionality is available, and if it is how to access it.

Consistency is important. People can **chunk** interaction sequences such as typing in a username and password, copying and pasting, opening applications. To “chunk” is to join several interrelated pieces of data into a single piece of data. Such as how encoding LBT WCP ULO may require more than just three chunks, but other arrangements should take just three chunks. You can also chunk procedural knowledge, such as how scrolling down in a document should be consistent across all applications, and the same actions should always accomplish it (whether it be two fingers up—or down—on a trackpad, moving your hand to and rolling the scroll wheel on the mouse in a manner that can be prepared before your hand arrives.

A expert-user command sequence for....

- Opening a program: Command-Space and the first few chars of the application name.
- Turning off unwanted “help” in PyCharm: See “+Notes on Using PyCharm IDE.pages”

Action sequences—or task strategies—should be consistent across applications, and should not conflict. This permits the user to plan and prepare the execution of the strategy before initiating the task. When the system fails to support the prepared and executed action sequence, not only does the user have to diagnose, troubleshoot, and experiment to figure out how to do it; but all of the preparation for the initial execution is also wasted. And the interaction with the device becomes the primary task, not the human-centered goal that initiated the interaction. For example: You go to print or scan a document, and it doesn’t work.

Mistakes: An inappropriate intention is established and pursued. More common among novice users. Buying a copy of “Garage Band” because you want to start a band in your garage.

Slip: The correct goal is attempted, but a problem arises along the way. More common among experts. Example: The goal is to get cash from an ATM; you do it but you leave your ATM card. Can often be avoided by improving the interaction design, such as by giving back the card before the cash.

More examples on page 169, with design approaches to avoid the problems.

**Modes** should, in general, be avoided in UI design. Modes are restricted interaction states in which only certain actions are possible. Such as a “modal” dialog box that requires a response before you can do anything else with your computer; some reminders software work this way, such as to alert you of a scheduled event. A pop-up window on a web page asking you to take a survey is a modal dialog box within the context of that web page. Smartphones use modes...
extensively; it contributes to their reconfigurable flexibility, but it also requires lots and lots of extra button presses and swipes to switch from one mode to another.

Articulatory directness—how directly a device maps to its input requirements—is interesting to think about in terms of touch-displays. Spreading two fingers is surely like stretching something, to zoom, but a four-finger versus a three-finger swipe does not seem to have articulatory directness with anything in particular.

Interpreting System Feedback
Give the user feedback with regards to how they are progressing towards their goals, at multiple time scales, including responding to any input within 100 ms, just to show that the system received your command, but also on the time scale of seconds, showing progress towards the goal. (Unix does not give great feedback. Many direct manipulation interfaces do.)

Storyboards
A storyboard is an event-by-event description of a sequence of interactions between a user and a device. They are named after the comic-book-like sequences that are used to plan movie shots.

A storyboard of the start of the bank robbery in Batman - The Dark Knight (2008)

http://s3images.coroflot.com/user_files/individual_files/152129_WOEkopMM6ezXqt52G9vrtE758.jpg

From Figure 5.7 in Rosson & Carroll
Storyboards are not interfaces but they capture, in a static representation, the time-based element of the interface, which makes it easier to consider alternative designs side-by-side. (Perhaps show the EyeMusic storyboards, annotating sound file, and the NIME promo video.)

How can you represent interaction sequences? Remember, a screenshot is not an interface. You must show how an interface evolves over time, such as with a storyboard. “Here is what the user sees. If they click here, then they see this....” The challenge is to represent a dynamic artifact.

**Action sequences can be studied, and improved, at different time scales, including the fractions of a second needed to move the mouse to click on a target, or press keystrokes.**

Fitts’ law predicts pointing time as a function of distance (d) and width (w). There is a logarithmic relationship between d/w and pointing time. \( MT = a + b \log(d/w) + 1 \). The main point is that tiny targets are very slow and difficult to click on, and the edges of the screen have certain advantages. But overall pointing-and-clicking is quite slow for time-pressured practiced tasks. You should learn keyboard shortcuts, even for responding to dialog boxes. (It is sort of foolish not to.) A good interface design should support keyboard shortcuts. One of the big differences between software for the masses like iPhoto and software for the pros such as Lightroom is that the pro versions support *lots* of keyboard shortcuts, such as to rate a photo and advance to the next photo with a single keystroke. (My friend Mark in NYC took my advice.)
Chapter 6 - Prototyping

A prototype is a concrete but partial implementation of a system. A prototype can evaluate many different aspects of a system design, such as speed, interconnections among components, or usability with a user interface prototype.

A prototype can be high fidelity and expensive (in time and money) or low fidelity and inexpensive.

The design questions that you want to answer will influence what level of fidelity is needed. If you just want to make sure people will be able to understand the commands that are available, and how to use them, a low fidelity prototype may suffice. But if you need to understand precisely how a device would perform in a high performance task environment, a high fidelity prototype may be needed. For example, a low fidelity prototype could be used to evaluate the arrangement and labeling of buttons on a home phone, or on AV equipment in a classroom. But a high fidelity prototype may be needed to evaluate how a new fire fighter communication device would work in a high-stress emergency situation, or how a device intended to track nutritional intake would be used in a real-life day-to-day setting.

A prototype should focus on some aspect of a system design. It is a partial implementation. Focus on the aspects that are most important to explore, such as what is novel. For EyeDraw, our first prototype was to just get the cursor to appear where the eyes are looking. If we couldn't do this, then we couldn't do anything.

Types of prototypes (p.199): Storyboard, paper, Wizard of Oz, computer animation (such as with Macromedia Director), scenario machine, rapid prototype, working partial system.

Storyboard: One or more pictures narrate a scenario. Figure 5.7 on p.191 shows the dynamic aspects of a system. Storyboards can also be hand-drawn (like Figures 4.16 and 4.17 on pp. 150-151), or like a comic strip using stick figures. (They are used when making movies.)

Low-fidelity prototypes are more likely to generate more ideas (Wendy Marsh from Intel). In the virtual science fair example: "The roughness of these prototypes convey that the design ideas are tentative and that wide-ranging input is needed." (p.210)

High-fidelity prototypes are more likely to mislead clients to think that major components of a system are completed, or to cause premature commitment to specific design ideas.

Paper-based mockups can work great. (See the DuckTix example.) In one form of a paper prototype, a separate sheet of paper is used for each screen and mode, and you add an extra little label to each button telling the person running the prototype which page to turn to after the user "clicks" on that button.

In a Wizard of Oz prototype, the user is presented what looks and acts like a real system, but in fact a human on the design team triggers the system responses, like the man behind the curtain in
the movie. But the team member triggering the responses needs to follow a strictly-defined set of rules, so as to not pretend that impossible-to-program human intelligence would be built into the system.

**Usability Testing**

"Usability testing is the core of usability engineering practice: Representative users are asked to interact with system prototypes, and their behavior and subjective reactions are studied." (p.204)

The ideal usability test would use the actual completed system but this is not the best use of resources if the goal is to explore potential interface alternatives.
Chapter 7 - Usability Evaluation

7.1 - Usability Specification for Evaluation

A usability evaluation is a study to determine the ease of use and ease of learning of a system. Ease of use is a measure of how well a system supports users accomplishing tasks.

Formative Evaluation vs. Summative Evaluation

takes place during the design process—how are we doing?
takes place after the design process—how did we do?

How usability evaluation fits into a software development process model:

Analytic Evaluation vs. Empirical Evaluation

Studying or modeling the interface without users. Cheaper, faster, sometimes can help to show what is wrong.

Examples
1. Usability inspection (Heuristic Evaluation)
   Ten guidelines on p.233. (Read them.)
2. Cognitive Modeling (KLM/GOMS)
3. Cognitive Walkthrough

KLM Operators Adapted From Card, Moran and Newell (1983)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Home hand to mouse (H)</td>
<td>0.4s</td>
</tr>
<tr>
<td>Press&lt;keyname&gt;(K)</td>
<td>0.28s</td>
</tr>
<tr>
<td>Press or release mouse button (B)</td>
<td>0.1s</td>
</tr>
<tr>
<td>Click mouse button (BB)</td>
<td>0.2s</td>
</tr>
<tr>
<td>Type &lt;string of characters&gt; (T(n))</td>
<td>0.3n s</td>
</tr>
</tbody>
</table>
Cognitive Walkthrough

Slightly biased towards ease of learning (not routine use, as is KLM). But ease of use and ease of learning are probably correlated.

Good for: ATMs, web sites with lots of new users (e.g., IRS, USPS, etc.), games, any “walk up and use” software.

Not good for: Air traffic control consoles, power plan control rooms, telephone switchboards.

Interesting detail: Cog. Walkthrough seems to assume the user will not be reading the manual.

Relates to the cycle of human information processing: Can users easily (1) perceive the state of the system, (2) determine the commands that are available for forming an action plan, (3) execute those commands, (4) perceive the results of those actions, (5) determine whether the user moved closer to their task goals, and (6) if the task is not completed, understand the next subgoal that needs to be accomplished?

Preparation Phase
Who are the users of the system?
What are the tasks?
What is the correct action sequence for each task?
How is the interface defined?

Analysis Phase: Walk through the interaction telling a credible story.
At every step or prompt, consider:
Will the user know the correct subgoal or subtask?
Will the user know that the correct action is available?
Will the user associate the correct action with the subgoal?
If the correct action is performed, will the user know progress is being made toward the goal?

How to fix the breakdowns:
If the user does not know...
Which subgoal to accomplish
  • Eliminate the required action
  • Prompt the use to make the action
  • Re-organize the interface to more closely support the users’ anticipated task hierarchy

The action is available
  • Make the controls more obvious, as with a prompt or a menu

The action is appropriate
  • Provide labels and descriptions for actions that incorporate the users’ vocabulary
  • Reword labels selected in error

Progress is being made
  • Prompt for the next correct action.
  • Provide feedback regarding what happened, ideally in the users’ vocabulary.

7.3 - Empirical Methods

This section in the textbook provides a very accurate and relevant discussion of how to conduct a usability study. This is perhaps the most important section in the textbook for you to learn. All of the terms that are in bold in this section are very important terms. (The Appendix on "Inferential Statistics" is also very good, on p.363.)

“The gold standard for usability evaluation is empirical data.”
Empirical: Based on observation (not theory or conjecture).

You are looking to establish a cause-and-effect relationship between characteristics of the system and ease of use. You want to claim that your interface causes a task to be easy to perform for a population.

... You want your experiment to have good “validity”.
Validity (Figure 7.3 gives examples of questions that relate to validity.)
Validity refers to the best available approximation to the truth of propositions.

External validity is the extent to which the experiment measures and shows something that is true about the world.
Internal validity is the extent to which the experiment truly measures what it tries to measure; that is, within the context of this particular experiment.

Example script for a usability study

Roles: Test monitor, technicians, users or “participants”.

Recruitment criteria (for this example):
1. Users who have never used an iPad, iPhone, or iPod touch.
2. Users who have used the iPhone (or iPod touch) calendar (at least once a day for at least a year? month? And who find it relatively easy to use). And who love their iPhone?

Purpose of observation: I am trying to learn how people might use the iPod Touch (or iPhone) to enter an appointment in a calendar.
This study should take about five to ten minutes. Feel free to quit any time.
I would like to ask you to think-aloud while you do the task. By this I mean to say what comes to your mind as you are working. To help you do this, I am going to ask the two of you to work together and to agree on every action that you take, and to make sure that both of you understand what is happening all the time. (The think-aloud protocol can be facilitated by two users doing “co-discovery.”)
Your first task is to create an appointment this Saturday from noon to 4PM to grade papers. Your second task is create an appointment on June 13 to attend commencement.

Debriefing questions:
1. What did you think?
2. How did you do the tasks?
3. How did you figure out how the calendar worked?
4. Did the system respond as you expected? Always?
5. Was there anything about the task that seemed particularly easy or difficult?
6. What were some of the feelings that you had as you did the task?
7. Do you think the calendar is easy or difficult to use?
8. Is there anything else that you would like to share about this?
9. Those are all of my questions. The study was designed for the hypothesis below. Do you have any questions for me?

My hypothesis, as you know, is that the iPhone calendar interface causes difficulty in recording appointments. (I have told you before that this is an unnecessarily difficult task.) I will operational my hypotheses to be:

H1: In order to enter an appointment, a novice user will require at least twenty screen touches (in which a swipe will count as two screen touches) in addition to the appointment’s text string.

H2: In order to enter an appointment, even an expert user will require at least twenty screen touches (in which a swipe will count as two screen touches) in addition to the appointment’s text string. And at least one error will occur for every appointment entered, in which an “error” is any undesired system response that requires the user to make an extra movement.

Important Topics in Empirical Methods

Think-aloud protocol - prompting a user to verbalize what they are doing as they proceed.
Co-discovery - having two users work together and agree on each step aloud.

Controlled experiments versus field studies.

Independent variable - characteristic that is manipulated to create different experimental conditions.
Dependent variable - an experimental outcome.
Hypotheses - predictions of causal relationships between dependent and independent variables.
Classic human performance measures are speed and accuracy. There is a tradeoff between them.
Experimental design - the details of how a cause-and-effect relationship is explored between independent and dependent variables.
Within-subject - all participants see all conditions.
Between-subject - different groups see different conditions.
Random assignment to remove order effects.
A major goal in experimental design is remove alternative explanations as to why the dependent variables changed when you changed the independent variables.
Informed consent - confidentiality, can quit any time without penalty. This is to protect participants.

The VSF examples are very good. The assistance policy, for example.
Chapter 8 - User Documentation

User documentation is stored information about how to use a system.

It takes many forms: paper or PDF document, online help, online tutors, and user forums.

User forums, as I am sure you have observed, are very mixed. Yes, Googling your question will often help you find an answer. But it also often just helps you to learn that lots of other people have the same question, and sometimes someone suggested an answer. But you do not know if it is correct. Such as how to uninstall a piece of software. How do you know if it is right?

"Socially mediated documentation can be fun and empowering, BUT the offered advice may be wrong or suboptimal...." (p.288)

Video documentation makes no sense to me even if just for the single reason that its content cannot be visually skimmed anywhere nearly as easy as pages of text and pictures to find what you need, or to figure out that your question is not answered here. Part of the problem is that the videos often start with company logos or talking heads rather than just jumping as quickly as possible to showing the task. They are modeled after TV shows or commercials, not a friend very quickly showing you how to do something.

Production Paradox: People want to be productive, to get things done, to achieve goals and subgoals. But in order to do this, they need to carry out activities that undermine their short-term progress towards the goal, that seem to prevent the production. (A paradox is a seemingly self-contradictory statement.)

Users often try to learn by doing. This might work for a photo organizing software like iPhoto, but not for the software that runs a space station. Most "power users" probably read manuals.

One solution is to try to put the documentation where the user will see it as they are doing a task with help that is built into the system so that users can "walk up and use" the system. But it is still difficult to know how much help to give, and whether they will even know that it is there. Tooltips can potentially help a user to learn by doing, but they can also get in the way of doing.

Documentation should support the human tasks, not just list the system functions.

User manuals should be organized by task. Hierarchical task analysis (HTA)—the same analysis that you did to help you understand user requirements and design your system—can also be used to organize your user manual.

Organize the manual hierarchically, following the HTA that evolves from (a) your initial requirements analysis, (b) the intended use of your system and how it fits into the intended use context, and perhaps also (c) how you observe participants using it during your user observation studies.
There are two kinds of knowledge (in many ways; this is just one):
1. How to use it—only what is needed to accomplish a task, such as how to use your turn signal.
2. How it works—the internal mechanics of the system, such as what makes the lights flash.
   (A lot of Apple's systems try to hide the second of these, such as in iPhoto, iMovie.)

Most users only need to know how to do a task. They only need to know how something works when they want to do an unusual task (such as synchronizing file systems), or to troubleshoot an unexpected system behavior (such as when a PDF got was deleted from an iPad—file transfers through iTunes works differently Pages than for ReaddleDoc).

Use the user's language. Don't explain the internal error codes or mechanisms. Tell them what they need to do to fix the problem and continue with their task. "You need to quit some applications to open another" rather than "Insufficient memory".

Advantages of Different Types of Documentation

<table>
<thead>
<tr>
<th>Paper-Based</th>
<th>On Hard Drive with App</th>
<th>Internet Forums</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Parallel viewing</td>
<td>• Searchable</td>
<td>• Sometimes easy to get suggestions.</td>
</tr>
<tr>
<td>• Can use anywhere</td>
<td>• Always there</td>
<td>• Can ask very obscure questions.</td>
</tr>
<tr>
<td>• Easy to annotate</td>
<td>• Can be at the function</td>
<td></td>
</tr>
<tr>
<td>• Can study deeply</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Creating a standalone application with a help system that requires an Internet connection seems very odd to me. At least some of these applications also permit a PDF to be downloaded as well. But you still need to plan ahead to make sure you have the PDF when you need it. It seems much more useful to just ship it with the application.

**Interactive tutorials** walk you through using the software. The Macintosh for a period of time had a help system that would open dialog boxes and circle appropriate buttons for the user to push to do a task.

**Scaffolding** is interface elements that try to guide new users through tasks, elements that are eventually removed as the user is able to do the project without the help, like scaffolding that is removed from a new building after it is built or maintained.

A "minimal manual" can be extremely helpful. It is action-oriented, builds on the user's task knowledge, and anticipates and manages errors. (Example: Codewarrior QuickStarts.) Many pieces of hardware come with both long comprehensive manuals as well as "Quick Starts".

**Mixed initiative** interfaces try to predict the user's task that is currently underway and tries to help them with it. They may have potential but, because of the colossal failure of Microsoft's Clippy, we may need to skip a generation to try it again. People *hated* it.

User documentation should be organized around, and should support, the user's task.
INTRODUCTION

HTA was developed by Annet and Duncan in the 1960s “in order to overcome the limitations of classical time-and-motion methods in analyzing complex nonrepetitive cognitively loaded tasks.”

Each subtask, or operation, includes a goal, input conditions, actions, and feedback. “The process of HTA is to decompose tasks into subtasks to any desired level of detail. Each subtask, or operation, is specified by a goal, the input conditions under which the goal is activated, the actions required to attain the goal, and the feedback indicating goal attainment.”

A “plan” defines the relationship between a task and subtasks. There are several types of plans. “The relationship between a set of subtasks and the superordinate task is termed a plan, and several plan types can be distinguished, including procedures, selective rules, and time-sharing tasks.”

The original purpose of HTA was to “identify actual or possible sources of performance failure and to propose suitable remedies, which may include modifying the task design and/or providing appropriate training.” This seems consistent with how it is currently used, such as to identify information needed within subtree B that was only available within an earlier subtree A (thus necessitating a new subtree B being created within subtree A). Such as the problem with Canvas shown at the end of this lecture. Or my OCCU Credit Card Payment slides.

Origins of HTA

In the 1950s, new technology and in particular automation was being introduced which, while reducing physical effort, increased “the mental effort required to monitor and control complex machinery”.

Time and motion studies became inadequate for describing mental content. Time and motion studies focused on visually observable behavior. (See Gilbreth’s “chronocyclograph” which uses small electric lights attached to a person’s hands, and a very long photographic exposure, to reveal patterns of physical motion.)

Miller (1953, 1962) developed a method for analyzing and describing human-machine tasks that “decomposed the main task functions into subtasks” that was evidently a precursor to HTA. (This was not the Miller of “The Magical Number Seven, Plus or Minus Two”.)

In the 1950s, information-processing theories of human performance were being “extended from the basic concept of limited capacity in the execution of simple perceptual-motor tasks (Fitts,
1954; Hick, 1952) to more general accounts of the acquisition of skill (Annett & Kay, 1956, 1957), selective attention and memory (Broadbent, 1958), and the principle of feedback control (Annett, 1969).” Miller et al. (1960; the “Magic Number Seven” author) asserted a theory of “goal-directed behaviour that was based on the concept of a nested hierarchy of feedback loops, referred to as TOTE (Test, Operate, Test, Exit) units, and it is this general conception that underlies the operation as the chosen unit of analysis in HTA.” The first test is whether the operation is needed; the second is whether the operation succeeded.

DEFINITIONS AND PRINCIPLES

Analysis Versus Description

“Analysis is not just a matter of listing the actions or the physical or cognitive processes involved in carrying out a task.... Analysis, as opposed to description, is a procedure aimed at identifying performance problems, that is, sources of error, and proposing solutions.” (Or, perhaps, avoiding potential performance problems in the case of task analysis done to assist in the design of a user interface.)

Tasks and Goals

A task is “‘any piece of work that has to be done’ and, as such, every task has a goal.”

“HTA differs radically from earlier methods of analysis by beginning, not with a list of activities, but by identifying the goals of the task.”

“In routine repetitive tasks, actions vary little.... In complex tasks, the same goals may be pursued by different routes and different means, depending on circumstances peculiar to each occasion. Hence, simply to list actions without understanding what they are for can be misleading. Complex systems are designed with goals in mind, and understanding how a system attains or fails to attain its designated goal is the primary purpose of the analysis.”

A goal can be stated formally as a goal state. (This is a term used in A.I.) The goal should be an event or a “physically observable value of one or more variables that act as criteria of goal attainment.” At any time, a goal can be either active or latent depending on whether it is currently being pursued or whether the person is waiting to pursue it later, when certain appropriate conditions arise. This distinction is important in the context of a “plan”.

Decomposition and Redescription

Complex goals can be defined by multiple events that must occur, or multiple variables that must be set. When multiple events or variables can be identified, the goal can be decomposed, or redescribed. That is, “successively unpacked to reveal a nested hierarchy of goals and subgoals.” This process provides an economical way of locating sources of system error (actual or potential) when trying to accomplish goals.
**Operations: Input, Action, and Feedback**

“An *operation* is the fundamental unit of analysis. An operation is specified by a goal, the circumstances in which the goal is activated (the *Input*), the activities (*Action*) that contribute to goal attainment, and the conditions indicating goal attainment (*Feedback*); hence operations are sometimes referred to as *IAF* units. Operations are synonymous with TOTE units (Miller et al., 1960) and are feedback-controlled servomechanisms.” A servomechanism is a device that permits a person to exert small forces to control large forces, such as when braking a car.

Operations can be decomposed into *nested hierarchies* of suboperations. Each suboperation should make a unique contribution to the attainment of its superordinate goal, and each suboperation should be mutually exclusive. Each set of suboperations should exhaustively state the subgoals needed to complete their superordinate goal.

An *action* is an instruction of what to do in a particular circumstance, such as the TOTE for hammering a nail:

- If the nail is not flush, then hammer.
- If the nail is flush, then exit.

Input and feedback both represent *states or tests*. The states register either an error, which triggers the action, or the cancellation of an error, which halts the action.

Each action is a *transformation rule*, which specifies how a servo responds to an error signal and its cancellation. For example, in a manual tracking task, a *transfer function* can quantify the control output required to correct for an error signal of a given direction and magnitude. However, transformation rules are often defined not mathematically but instead with commonly understood verbs such as lather, peel, sand, Google, or email. Often each these verbs represents a transformation or operation that itself could be further defined by a set of sub-operations.

**Plans**

The *plan* specifies the order in which suboperations (subtasks) should be carried out. “When a goal becomes active, its subordinate goals become active according to the nature of the plan.”

The two most common types of plans:

1. *Fixed sequence*, or, *routine procedure*. Do this, and then this, and then this.
   - Example: Review the course syllabus in a browser.
2. *Selective rule*, or, *decision*. “If <a> then do <x>. If <b> then do <y>.”
   - Example: Get your laptop connected to a cafe’s wireless. A password might be needed.

Both types of plans “imply knowledge on the part of the operator. It may be simple procedural knowledge, or the plan may require extensive declarative knowledge of the environment, the limits and capabilities of the machine, safety rules, and much else besides.”

“A third distinct type of plan requires two or more operations to be pursued in parallel; that is, the superordinate goal cannot be attained unless two or more subordinate goals are attained at the same time.” The plan is referred to as a:
3. Time-sharing, or, dual task plan.

Example: Troubleshoot a computer while doing other work. (I write subgoals on Post-Its.)

“This type of plan also has significant cognitive implications in terms of the division of attention,
or, in the case of team operations, the distribution of information between team members acting
together.”

Stop Rules

Stop rules determine how deep an analysis should go in a task decomposition (and not to the
stopping of an operation within a plan because the operation has been successfully completed).

The ultimate stop rule is simply “Stop when you have all the information you need to meet the
purposes of the analysis.”

A more engineering-oriented stop rule is “Stop when the product of the probability of failure and
the cost of failure is judged acceptable.” This is known as the $p \times c$ criterion. The goal is to
minimize unneeded analytical work, and to focus the analyst on the aspects of the design that are
critical to system success. It is, however, often difficult to estimate $p$ and $c$ due to a lack of
empirical data.

“The obvious reason for stopping is that the source of error has been identified and the analyst
can propose a plausible remedy in terms of either system design, operating procedures, or
operator training, that is, by redesigning the cognitive task.”

“HTA may be used for design purposes and need only be carried down to the level of equipment
specificity.”

“Team training often follows training in individual operator skills, and the analysis of team tasks
may be usefully taken down to the level at which operations involve communication and
collaboration between individuals.”

Some Reflections

I am quite struck by the similarities between HTA and GOMS. For example, I did not
know that HTA had these three different kinds of plans, the first two of which are in GOMS (but
not the third one).

I believe that I am slowly coming to understand the value of HTA to design. One is that it
motivates the analyst or the designer to group related operations into mid-level, mid-sized
chunks of work, and to thus try to identify how they believe people will think about the system.
It’s kind of like one very specific instantiation of a “mental model”. Once these assumptions are
articulated, they can be critiqued.

Dave Kieras adds: It reflects this ancient intuitively obvious notion that tasks consist of
subtasks. It permits you to identify methods that can be reused in the analysis, and by users.

Q&A

• Stopping criterion? It is presumed that each operation provides some sort of “feedback”
indicating the operation has been completed.
• Why better than flowchart? Because, although simple tasks can be defined by a string of actions, complex tasks can be accomplished by many different ways. It is thus important to cluster subsets of actions into meaningful groups that represent subsets of actions that are closely related to each other because they relate to a single subgoal within the hierarchy. It is not only important to know what is the sequence of actions that a person takes, but also why a person takes (or would take) such a set of actions. It sort of relates to the modular design of functions, and the programming heuristic of DRY, or, “don’t repeat yourself”.

Hence, simply to list actions without understanding what they are for can be misleading.

HOW TO CARRY OUT AN HTA

...  

Step 4: Acquire Data and Draft a Decomposition Table or Diagram

...
One of the many interface problems that will result from an interface built without adequate consideration of the user’s task hierarchy.

Canvas arbitrarily constrains dates that can be entered, permits you to enter dates that it will not accept, and then simply blocks you from completing the task if it determines the date is invalid. It does not use the user’s language to describe the error. But more importantly, (a) it fails to provide information that is needed at this point in the task execution hierarchy and (b) does not provide a means of getting that information. Entering the last possible end date would be a particularly difficult task to accomplish on a touchscreen device with no side-by-side viewing of windows or applications.