Concepts of User-centered Interaction Design

Glenn Booker 5/7/2012

Introduction

This overview of user-centered design consists of three sections.

- A summary of chapter 7 from The Design of Everyday Things.
- A summary of chapters 9 and 10 from Interaction Design: beyond HCI.
- Jakob Nielsen's ten usability heuristics.

The Design of Everyday Things (DOET) by Donald A. Norman

Basic principles

The goal of design is to make sure 1) the user can figure out what to do, and 2) the user can tell what is going on. More specifically:

1. Make it easy to determine what actions are possible at any time through constraints.
2. Make things visible, e.g. the conceptual system model, alternative actions, and results of actions.
3. Make it easy to evaluate system state.
4. Use natural mappings between intentions and actions, actions and effects, and visible information and system state.

Deliberately make things difficult (for safety, security, etc. reasons) by breaking some rules for one part (not all). For examples:

- Make critical things invisible
- Use unnatural mappings for execution (controls seem inappropriate) or evaluation (system state is hard to tell)
- Make actions physically difficult to do (require strength, size, etc. That some users don’t have)
- Require precise timing and/or actions (launch nuke)
- Don't give feedback

The number of possible system actions and the number of system controls visible to the user affects the complexity of use of the system, and the complexity of appearance (does it look hard to use?). Apparent and actual complexity can be quite different (e.g. ice skates). Hide controls not needed to improve (reduce) apparent complexity.
General terminology

<table>
<thead>
<tr>
<th>Alternative actions</th>
<th>Choices the user could make at some point, other than the most common or intended action.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conceptual model</td>
<td>There are three conceptual models:</td>
</tr>
<tr>
<td></td>
<td>• The design model is how the designer imagined the system will work.</td>
</tr>
<tr>
<td></td>
<td>• The user's model is how the user thinks the system works. Should equal the design model.</td>
</tr>
<tr>
<td></td>
<td>• The system image is the visible part of system itself, which hopefully maps the design and user's models.</td>
</tr>
<tr>
<td>Constraints</td>
<td>Natural or artificial design factors that limit what options the user can choose.</td>
</tr>
<tr>
<td>Current state</td>
<td>The state of the system now.</td>
</tr>
<tr>
<td>Evaluation</td>
<td>User assessing whether the action had the desired effect.</td>
</tr>
<tr>
<td>Execution</td>
<td>User performing an action.</td>
</tr>
<tr>
<td>Natural Mappings</td>
<td>Part of the field of response compatibility. See step 4 in DOET design principles.</td>
</tr>
<tr>
<td>Natural relationships</td>
<td></td>
</tr>
<tr>
<td>Results</td>
<td>The outcome of some action, which might include some change in the system state.</td>
</tr>
<tr>
<td>User</td>
<td>The person who is using the system</td>
</tr>
</tbody>
</table>

DOET design principles

1. **Use knowledge in the world and knowledge in the head.** Novice users look to the system (the world) for guidance on what to do. Expert or frequent users know in their head what needs to be done. This includes understanding the three conceptual models, and writing good documentation.

2. **Simplify task structure.** Keep within limits on short term memory (5 unrelated things) and long term memory. Restructure tasks, and/or reduce the user’s mental load via mental aids (reminders). Four main approaches:
   - Keep tasks the same, but add mental aids (notepads, clocks)
   - Use technology to make invisible things visible for more feedback and control (car instruments)
   - Automate existing tasks (automatic transmission, TV dinners)
   - Change the nature of the task (Velcro instead of shoelaces)

Avoid overautomation (get too dependent upon it).

3. **Visible things bridge execution and evaluation.** User should know what is possible to do, how to do them, and their effects on system state. Action outcomes should be obvious. Make sure visible things are meaningfully explained (loop on remote control).

4. **Map things right.** Exploit natural mappings. There should be a connection or cause-and-effect relationship between many pairs of ideas or things.
• User intentions to actions (if I want to turn on the cruise control, what action do I need to take?)
• Actions and system effects (if I turn on the cruise control, what does that do to the system?)
• System state and user perception (if the cruise control is on, can I tell that?) To achieve this, exploit natural relationships and constraints.
• Perceived system state and user needs, intentions, and expectations (does the state I think the system is in fulfill my needs, etc.? if the cruise control is on, does it do the job I think a cruise control should do?)

5. **Exploit natural and artificial constraints.** Good constraints make sure the user can only do the right thing!

6. **Design for error.** Users will make all possible errors, so allow for reversing actions, error detection and recovery, etc. Support correct user responses. Make it hard to do irreversible things.

7. **When all else fails, standardize.** Use this when world information and natural mapping can't be used. E.g. car controls, keyboards, analog clock faces.
Interaction Design by Rogers, Sharp, and Preece (3e)

The Process of Interaction Design

Interaction design has four main approaches (Saffer, 2010):

- **User-centered design**, where the user **needs and goals** are the driving force behind design.
- **Activity-centered design**, where the user's behavior around particular tasks is the focal point.
- **Systems design**, a structured rigorous approach focuses on the context of a complex problem.
- **Genius design**, the Apple-like approach to use the experience and creativity of the lead designer (Steve Jobs?) to drive design, based on user validation of designer ideas.

All kinds of design involve 1) defining the requirements, 2) producing a design that meets those requirements, and 3) evaluating the design. Key trade-offs to resolve include (here) how much choice is given to the user, and how much direction the system should offer.

Generating alternatives is key for most kinds of design; brainstorming is a common way to do this. Need to capture ideas in a way the users can understand them (i.e. sketches, prototypes). Ultimately want to exceed user's expectations for the final product than fall below them. Training users, and user involvement in the design process also contribute to expectation management.

Users may have a wide range of involvement in the development process, from managing it entirely, to briefly appearing a few times, to getting occasional newsletters, to evaluating the product after release. Too much user involvement can result in 1) too many new features introduced late in the project, 2) users fearing the product would threaten their jobs, 3) goals may become too high, leading to stress, conflicts, and increased reworking of the product. In short, have just enough user involvement!

**User-centered approach for development**

Real users and their goals, not technology, drive product development. Basic principles include

- **Early focus on users and tasks** - observe users doing tasks, assess user cognitive, behavioral, etc. characteristics.
- **Empirical measurement** - of users' reactions and performance, both of existing system and new prototypes or simulations. Identify usability and user experience goals from the start. Use them to evaluate alternatives and monitor project progress.
- **Iterative design** - (design, test, measure, fix) repeat. Key is to get user feedback to refine the design choices. Assess what is needed, helpful, and feasible. Iteration is always needed.

'Early focus on users and tasks' can be refined with:

- What technologies are available to provide better support for users' goals?
- Study users' behavior and context of use. How do people perform their tasks? Behavior leads to priorities, preferences, and intentions. We want to improve work, not automate bad habits.
• Users' cognitive and physical aspects are captured and designed for. Attention, memory, perception, color vision, and many other issues can affect design choices and may differ for some types of users.
• Users may be consulted throughout development until after release, as discussed above. Respect user input!
• All design decisions are made within the context of users' work and environment. Using personas, or keeping gathered data or brainstorm ideas readily available, can help do this.

Four basic activities of interaction design

The same activities apply to many forms of design, e.g. architecture. They are:

• **Establishing requirements**, based on user and task analysis
• **Designing alternatives**, includes conceptual design (what can you do with the product, and how do you interact with it) and physical design (details of product design, colors, etc.), with alternatives investigated throughout.
• **Prototyping**, which could be digital or paper-based.
• **Evaluating**, to determine usability and acceptability, based on user errors, product appeal, requirements compliance, etc. This step eventually produces a final acceptable product.

Interaction design life cycle model

These basic activities of interaction design easily become a lifecycle model for interaction design, such as ISO 13407, Star. See also [http://www.usabilitynet.org/tools/13407stds.htm](http://www.usabilitynet.org/tools/13407stds.htm), [http://cml.hci.uni-bamberg.de/~gross/publ/jaihc_jrnl10_gross_hcc.pdf](http://cml.hci.uni-bamberg.de/~gross/publ/jaihc_jrnl10_gross_hcc.pdf) The lifecycle model should be tailored to meet the culture, size and complexity needs of the project at hand.

![Figure 2. Activities of user-centered design](http://michaelyeap.blogspot.com/2009/08/aug-29-jokela-et-al-standard-definition.html)
Interaction design activities can be integrated into other life cycle models. Costabile shows below how it can be added to the waterfall life cycle model, though it adapts better to agile development methods (XP, Crystal, Scrum, ASD, DSDM). A case study of ASD development used parallel tracks for code development and interaction design, with code and design data exchanged at the start of each cycle of development. Some argue that field study of user needs should even precede the start of the product develop lifecycle.


**Who are the users?**

Users might include:

- People who interact directly with the product or system
- Their managers
- Those who receive products from the system
- System testers
- Those who purchase the system
- Those who make competitors for the system
• Those who will lose their jobs because of the system
• Lots of possible stakeholders!

Can divide users into primary users (use system all the time), secondary (use it sometimes), and tertiary (are affected by system output). Could use an onion diagram to show layers of users.

What are 'Needs'?

People often don't know what they need; they don't know what is possible. There are many 'un-dreamed-of' requirements. Hence have to understand the characteristics and capabilities of users, what they want to achieve, how they do so now, and if they could achieve it more effectively or enjoyably.

User characteristics can include physical ones: motor skills, height, strength. Cultural needs should be accounted for, especially in terminology, attitudes toward technology. New inventions might be challenging to identify requirements, since no user can imagine using it yet.

Be aware of designer biases toward experiences and expectations. You want to design something you would want to use, and something like what you have designed before! Look for similar existing behavior. Focus on people's goals, usability, and user experience goals.

How do you generate alternative designs?

Don't settle for a familiar 'good enough' design. Often innovation comes from cross-breeding ideas from different areas, perspectives, applications, etc. Browse other design collections for inspiration. Prior experience, even from others, can feed ideas. Look for analogies in other fields. Look at similar systems (competitors) and how to improve upon them. Design balances constraints and requirements, so sometimes that will limit your vocabulary of design options. Play with visual props when possible (TechBox). Consider not just things, but also processes, activities, tasks. Draw from known designs that work, whether yours or others' (design patterns).

How do you choose among alternative designs?

Design decisions fall into two categories, those visible to the user, and those not visible. We focus on the externally measurable characteristics, though they are influenced by internal design choices.

Choose among alternatives by letting the users express experiences, preferences, and suggestions. Traditional design documents don't convey the user experience with the product. Prototyping does!

Can also assess design quality, but need to define that clearly. It might relate to product performance, or usability efficiency, or other traits. Quantify the characteristics of interest to make a fair comparison. Having verifiable usability criteria in a usability specification is part of usability engineering.

Basic usability goals include effectiveness, efficiency, safety, utility, learnability, and memorability. How do you measure user experience? Satisfaction, fun, motivation, aesthetics, entertaining?
Establishing Requirements

Users achieve their goals through activities done in some context. The requirements analysis seeks to understand all of those dimensions. It is far cheaper to establish a set of stable requirements right early in the project than fix them later on. We establish requirements based on user needs and associated data. Cultural concerns make it a key issue for global companies whether to have one website (Pepsi), or regional ones (Coke).

A requirement is a statement about a product that specifies what it should do (functional requirements) or how it should perform (non-functional requirements). Requirements are often decomposed into more specific requirements under some kind of hierarchy. Constraints can be specified, including design constraints or project management concerns. The Volere template for requirements includes product scope and project management issues:

- Project drivers - purpose of product, stakeholders
- Project constraints - mandated constraints, conventions & definitions, facts and assumptions
- Functional requirements
- Non-functional requirements - including look and feel, usability, performance, environment, cultural, and legal issues
- Project issues - risks, product migration, costs, documentation, open issues.

User profiles can be expanded into personas to describe realistic imaginary users, including their skills, attitudes, training, tasks, and environment. These help develop usability goals and user experience goals. Measures of these can be objective and quantitative, or subjective and qualitative.

Data gathering for requirements

Lots of methods can be used for gathering requirements.

- A novel one described is a cultural probe package, which seeks insight into the lives and aspirations of people. Other variations on this probe concept have been quite successful.
- Interviews - Good to explore issues, elicit scenarios. Can be structured or semi-structured or unstructured.
- Focus groups - good for consensus view and identifying areas of conflict. Tend to be well structured to get specific deliverables.
- Questionnaires or surveys - good for initial responses, then pick people to interview later. Gets impressions, opinions.
- Direct observation - good to understand the nature of tasks as they are currently done, and their context.
- Indirect observation - diaries and interaction logging can be done, often for product evolution ideas or improving websites.
- Studying documentation - good for task or activity details, relevant regulations. Never use as sole method!
• Researching similar products - good for generating alternative designs, prompting requirements.

Generally best to use more than one data gathering approach; which ones depends on the product, context, etc. Often do prototyping to help evaluate the requirements obtained so far.

Data gathering for requirements should:

• Focus on identifying the stakeholder’s needs
• Involve all stakeholder groups, maybe more than one person from each group for large groups
• Support data gathering with props - task descriptions, prototypes, etc.

**Contextual inquiry**

This is one seventh of *contextual design* to collect and interpret field data for software development. Based on ethnographic data gathering where the designer acts as an apprentice to the user. Is based on four principles:

• Context principle - go to the workplace and see what happens while the user is working
• Partnership principle - developer and user should collaborate in understanding the work.
• Interpretation principle - observations must be interpreted before use in design, and that's also done collaboratively.
• Focus principle - keep data gathering focused on your goals. Don't wander!

Data collection and interpretation sessions are usually separate activities, the latter generating models of work flows, artifacts, culture, etc. Contextual inquiry is part unstructured interview, part observation, plus collaborative interpretation of data.

**Data analysis, interpretation, and presentation**

A formal structure can help ensure that requirements gathering is complete (e.g. the Volere shell). Address traceability of requirements - who or where did it come from and when?

Functional requirements can be investigated with class diagrams, state charts, sequence diagrams, entity-relationship diagrams, or data flow diagrams. Here focus on four user-centered techniques.

Expect that requirements will grow and change as a deeper understanding of them is developed. Iteration is the norm! Start with less formal documentation, and formalize it when it becomes more solidified.

Brainstorming can help identify requirements too. Suggestions include

• Have a broad range of participants
• Don’t ban silly ideas
• Let ideas build on each other or combine
• Keep records
• Keep focus on a well defined problem
• Use warm-up exercises (word games, physical props, etc.)

Task Description

Common approaches for describing tasks include:

• **Scenarios** - an informal narrative description, technology-free, in the vocabulary and phrasing of users. Often about a paragraph long, and based on telling stories of how a task occurs and why it’s important (goals). Forms a great place to explore constraints, context, etc. Level of detail in scenario might increase during development. Can be in the current environment, or a future wishful scenario.

• **Use cases** based on object oriented description of a user (actor) involvement with a system. Each use case is a goal of an actor in using the system (a functional requirement). Provides a detailed narrative of actor and system interaction. Also describes *alternate courses* of the use case. Can be described in massive detail later on (Cockburn).

• **Essential use cases** (task cases) are more abstract descriptions of *user intentions* and the corresponding *system responsibilities*. Similar in format to a two-column use case.

Task Analysis

Is done to investigate the rationale and purpose of an existing situation or practice. What are they trying to achieve? Why are they trying to achieve it? How are they going about it?

**Hierarchical Task Analysis** (HTA) is the most commonly used version, and was originally used for training need identification. Break a task into subtasks, then sub-subtasks, etc. Focus is on *physical and observable actions*, starting with a user goal.

Documenting HTA can be in numbered steps (textually), or graphically like an org chart. The latter can be notated with the flow of steps, indicating repeated sets of tasks, decisions made, etc.

Problems with HTA include that real tasks are often too complex to show this way, and parallel tasks or task interruptions can’t be modeled. OTOH it lets you compare alternative designs easily, provides understanding of tasks at various levels of abstraction, and supports design reuse.
Ten Usability Heuristics by Jakob Nielsen

These are ten general principles for user interface design. They are called "heuristics" because they are more in the nature of rules of thumb than specific usability guidelines.

Visibility of system status
The system should always keep users informed about what is going on, through appropriate feedback within reasonable time.

Match between system and the real world
The system should speak the users' language, with words, phrases and concepts familiar to the user, rather than system-oriented terms. Follow real-world conventions, making information appear in a natural and logical order.

User control and freedom
Users often choose system functions by mistake and will need a clearly marked "emergency exit" to leave the unwanted state without having to go through an extended dialogue. Support undo and redo.

Consistency and standards
Users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform conventions.

Error prevention
Even better than good error messages is a careful design which prevents a problem from occurring in the first place. Either eliminate error-prone conditions or check for them and present users with a confirmation option before they commit to the action.

Recognition rather than recall
Minimize the user's memory load by making objects, actions, and options visible. The user should not have to remember information from one part of the dialogue to another. Instructions for use of the system should be visible or easily retrievable whenever appropriate.

Flexibility and efficiency of use
Accelerators -- unseen by the novice user -- may often speed up the interaction for the expert user such that the system can cater to both inexperienced and experienced users. Allow users to tailor frequent actions.

Aesthetic and minimalist design
Dialogues should not contain information which is irrelevant or rarely needed. Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility.

Help users recognize, diagnose, and recover from errors
Error messages should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution.

Help and documentation
Even though it is better if the system can be used without documentation, it may be necessary to provide help and documentation. Any such information should be easy to search, focused on the user's task, list concrete steps to be carried out, and not be too large.

I originally developed the heuristics for heuristic evaluation in collaboration with Rolf Molich in 1990 [Molich and Nielsen 1990; Nielsen and Molich 1990]. I since refined the heuristics based on a factor analysis of 249 usability problems [Nielsen 1994a] to derive a set of heuristics with maximum explanatory power, resulting in this revised set of heuristics [Nielsen 1994b].

From http://www.useit.com/papers/heuristic/heuristic_list.html