Application Design

21W.780 – Class 4
March 7, 2006
Frank Bentley

Some thoughts on design…

Usability is one of the biggest problems in interactive systems/environments (and usually an afterthought)

Software, like buildings and physical devices should be designed to support the tasks of people who will be using them

You are not the user (most of the time)

It's much easier to change design early in the process compared to after code is written

Application Design is just not about a pretty front end

Overview

Deciding what to build

- Contextual Inquiry / Ethnographic Observation
- Contextual Inquiry User Models
- Task list
- Requirements list

Deciding how to build it

- User Environment Diagrams
- Paper Prototyping
- Heuristic Evaluation
- Usability Testing

Contextual Inquiry / Ethnographic-Style Observation

Contextual Design (Beyer and Holtzblatt, 1998)

- **A process of developing user requirements by understanding user behavior**
- **Involves observing users performing tasks similar to those they would be performing with your system**
  - People cannot be relied on to tell you what they think or how they approach tasks
  - In context, people can relate what they are currently thinking (“ think aloud” methods, probing questions)

Contextual Inquiry / Ethnographic-Style Observation

Who to involve

- **Users most similar to those who will be using your system**
- **As diverse a set of users as you can get (age, gender background, lifestyle, tech usage, etc.)**
- **7-10 users is typically enough, stop when you keep seeing the same things**

What to observe

- **Tasks people perform / steps performed in those tasks**
- **Critical Incidents (things that don’ t go as expected or things that are exceptionally good)**

Example – Music Use

Observe participants' music collections in their homes

- **Observe organization**
- **Probe for uses of music in different places**

Have them perform tasks that they perform in their daily lives:

- **Choosing work out music**
- **Choosing cleaning music**
- **Choosing music for a part**

Goal is to understand behavior and ground future ideation

CI Models

Models summarize user behavior

Aggregated models across all participants can help in design

Artifact Model

- Capture information/things user interacts with

Cultural Model

- Capture people users interact with to complete tasks

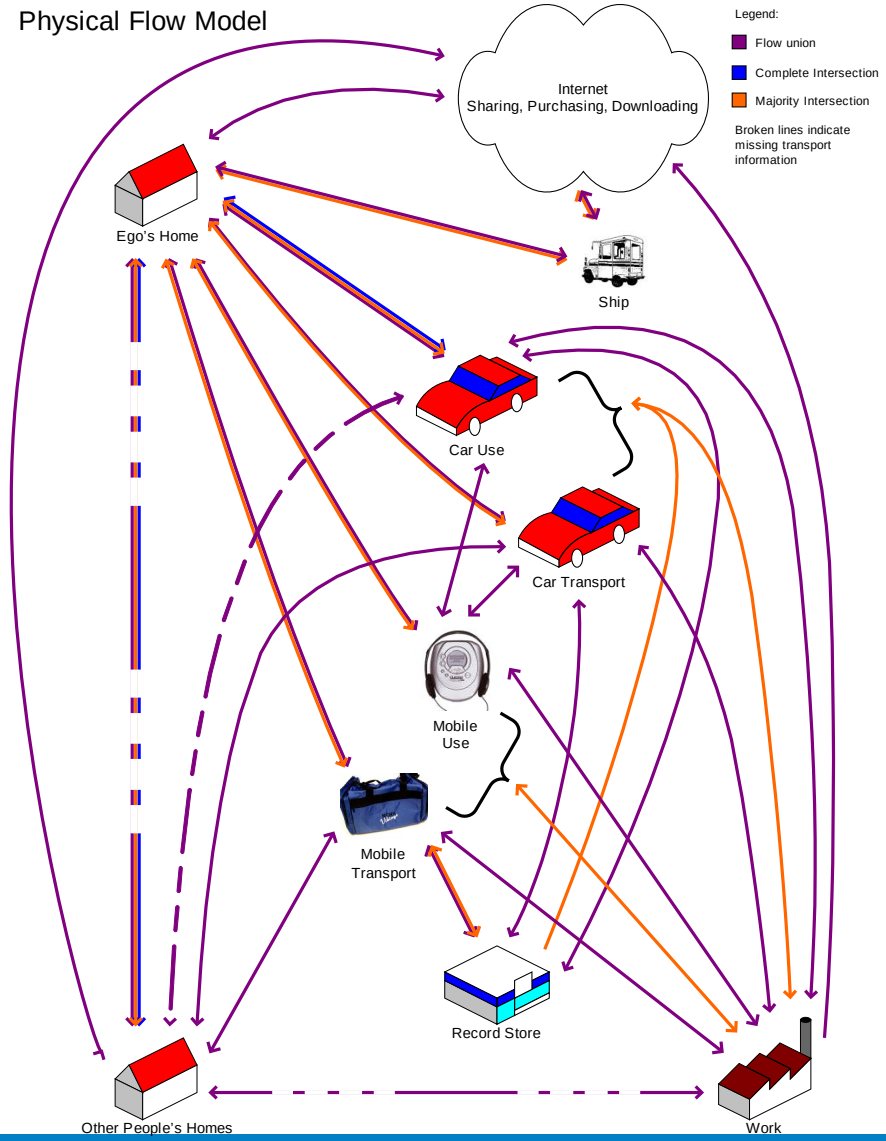
Physical Model

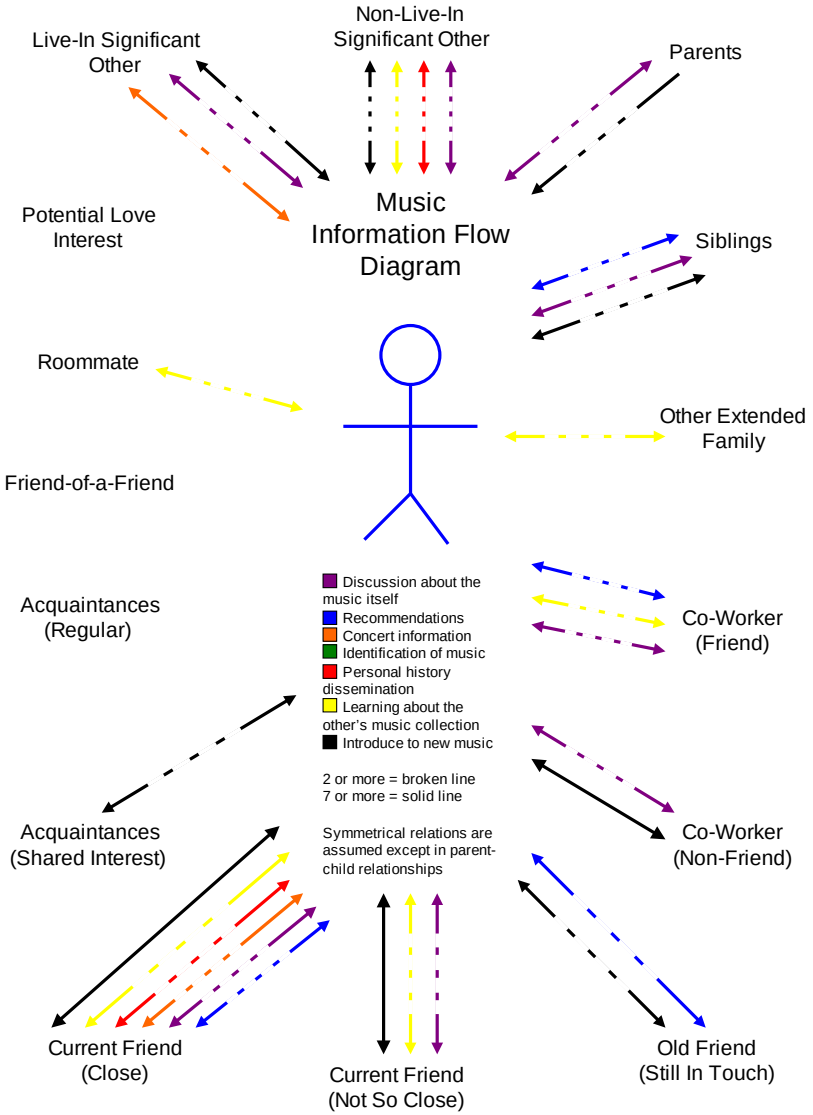
- Capture how people move and interact with a space

Sequence Model

- Capture tasks and steps performed to complete the tasks

Examples:





Task list

-Tasks are high level concepts of purposeful use

- Most revolve around end states the user would like to be in (e.g. “ select set of music with the intent to play it”)
- Not individual requirements for a system
- Ideally, tasks come from observations, user needs

Example

- Select a set of music to play
- Play music
- Control the playback of music
- Stop playing music

Requirements list

- **Functional requirements needed to accomplish tasks**
- **Can be user facing (visible) or system facing (hidden)**
- **Should exhaustively enumerate everything the application/system has to perform**
- **Prioritize list to determine what will be implemented / what can safely be omitted in early versions (common prioritizing is Core, Important, Nice to Have)**
- **Prioritize by use (Used by many, most, few) and expected frequency (Used often, sometimes, rarely/once)**
- **You' ll rarely be able to implement everything or cleanly fit it into a design**

Example

- Select a set of music to play
  - View available artists / albums / genres / playlists for all music in collection (U)
  - Select an artist / album / genre to play (U)
  - See song titles of music in that category (U)
  - Ability to catalog artist / album / genre for each song (S)
  - Ability to associate music into playlists (S)
- Play music
  - From a list of songs, be able to enter a playing mode (U)
  - Music should flow from one song to the next in the list (U)
  - Maintain a list of the current songs to play (S)
  - Play music through external speakers (U)
- Control the playback of music
  - Be able to pause music (U)
  - Be able to resume music (U)
  - Be able to skip over a song in a playlist (U)
  - Be able to return to browsing for different music to play (U)
  - Be able to fast forward / rewind in a track (U)

User Environment Diagrams

Represent groups of tasks / requirements that the user will perform into “ focus areas”

Shows links between areas

Begins to approximate user interface

Each area is meant to represent functions and objects of interaction required for a particular type of work

For each area list:

- **Purpose (summary of why the user would be in this state)**
- **Functions (list of available functionality in this state)**
- **Links (list of places the user can navigate to from here)**
- **Objects (things the user can see and interact with here)**

Hidden areas can represent tasks done by the system

Example

Selecting Music

Purpose: To allow the user to view all music available and to subset this music to create a list to play

Functions:

- View available artists / albums / genres / playlists for all music in collection
- Select an artist / album / genre to play
- See song titles of music in that category
- Play music

Objects: lists of available attributes, lists of available values, lists of songs matching vaules

Playing Music

Purpose: To control the playback of music

Functions:

- See what song is currently playing
- Pause music
- Resume music
- Skip over a song in a playlist
- Return to browsing for different music to play
- Fast forward / rewind in a track
- Maintain a list of the current songs to play
- Go back to selecting music

Objects: list of currently queued songs

Paper Prototyping

Serves as a mock up of design

Helps see potential trouble spots

Helps think out tough areas in detail

Provides model that you can interact with without spending time writing throw-away code

Allows for iteration with low-cost

Heuristic Evaluation

A list of common errors in user interfaces to check your interface against (sanity check)

Simple way to evaluate interface without involving formal user study

Can generally solve many initial usability issues

No replacement for usability testing

Nielsen’ s Heuristics

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.

Usability Testing

Best way to learn how interface will be used is to see it used

Choose tasks that users would actually perform (don’ t ask someone to do something they never intend to do)

Use 5-7 users to catch majority of major flaws

Tell user that interface is being tested, not them

Have users “ think aloud” verbalizing what is going through their heads, not reflections on what they are doing

Don’ t help users (only ask them to keep talking or move to the next task upon success / failure)

Determine ahead what constitutes a failure case, don’ t allow users to run amok in your UI aimlessly

Watch for critical incidents

References

Contextual Inquiry / CI Models / User Environment Diagrams

Beyer, H. and Holtzblatt, K. 1999. Contextual design. *interactions* 6, 1 (Jan. 1999), 32-42. DOI= http://doi.acm.org/10.1145/291224.291229

Beyer, H. and Holtzblatt, K. 1998 *Contextual Design: Defining Customer-Centered Systems*. Morgan Kaufmann Publishers Inc.

Tasks and Requirements Analysis

Ellen Isaacs , Alan Walendow , Alan Walendowski, Designing from Both Sides of the Screen: How Designers and Engineers Can Collaborate to Build Cooperative Technology, New Riders Publishing, Thousand Oaks, CA, 2001

Paper Prototyping

Rettig, M. 1994. Prototyping for tiny fingers. *Commun. ACM* 37, 4 (Apr. 1994), 21-27. DOI= http://doi.acm.org/10.1145/175276.175288

Heuristic Evaluation

Nielsen, J. 1994. Heuristic evaluation. In *Usability inspection Methods*, J. Nielsen and R. L. Mack, Eds. John Wiley & Sons, New York, NY, 25-62.