

# Human Computer Interaction

## Orchestration, Flow, and Eliminating Excise

Prof. Andrew D. Bagdanov

Dipartimento di Ingegneria dell'Informazione  
Università degli Studi di Firenze  
`andrew.bagdanov AT unifi.it`

October 3, 2017

- 1 Exam schedule
- 2 Attention, Thought, and Action
- 3 Orchestration and Flow
- 4 The way forward
- 5 Homework

## Exam schedule

- This is the final exam schedule up through Easter:

<i>Appello</i>	<i>Date</i>	<i>Verbalizzazione (entro)</i>
1	15/01/2018	19/01/2018
2	19/02/2018	23/02/2018
3	09/04/2018	13/04/2018

- My goal is to finalize all exams **within the week they are scheduled**.
- That is, by the **Friday** after the official date of the exam.

# Attention, Thought, and Action

- When people interact purposefully with the world around them, including computer systems, some aspects of their behavior result from the **limited capacity of attention and short-term memory**.
- When interactive systems are designed to **recognize and support** those patterns, they fit better with the way people operate.
- Some user interface design rules are based directly on the patterns and thus indirectly on the limits of short-term memory and attention.
- Here we will look at six important examples of these patterns.

- When people are doing a task – trying to accomplish a goal – most of their attention is focused on the **goals** and **data** related to that task.
- Normally, **people devote very little attention to the tools** they are using to perform a task.
- When people refocus their attention on their tools, it is **pulled away from the details of the task**.
- This shift increases the chances of users losing track of what they were doing or exactly where they were in doing it.
- That is why most software design guidelines state that software applications and most websites **should not call attention to themselves**.

- Because our short-term memory and attention are so limited, we learn not to rely on them.
- Instead, we mark up our environment to show us where we are in a task:
  - **Counting objects:** if possible, we move already counted objects into a different pile to indicate which objects have already been counted; to keep track of the number we are on, we count on our fingers, draw marks, or write numbers.
  - **Reading books:** When we stop reading, we insert bookmarks to show what page we were on.
  - **Checklists:** We use checklists to aid both our long-term and short-term memory: in critical or rarely performed tasks, checklists help us remember everything that needs to be done.



# We use external aids to keep track

- One implication of this pattern is that interactive systems should **indicate what users have done versus what they have not yet done**.
- Most email applications do this by marking already-read versus unread messages, and most websites do it by marking visited versus unvisited links, and many applications do it by **marking completed steps of a multipart task**.



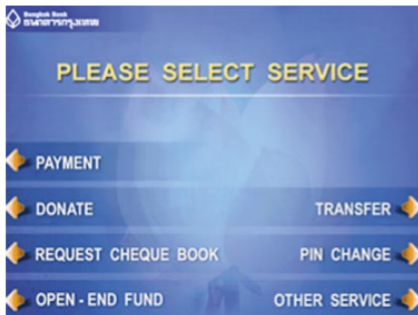
# We use external aids to keep track

- Another design implication is that applications should let users **mark** or **move objects** to group and indicate which they have used before:

▶	Can't Edit Prev Entered Data	1/2/07
▶	Cancel doesn't cancel	12/29/06
▶	Missing options	1/13/09
▶	Moving controls	9/14/09
▶	Trapping User	9/14/09

- Focusing our attention on our goals makes us interpret what we see on a display in a **very literal way**.
- **People don't think deeply** about instructions, command names, option labels, icons, navigation bar items, or any other aspect of the user interface of computer-based tools.
- This tendency of people to notice only things on a computer display that match their goal has been called *following the scent of information toward the goal*.

- Goals can lead us to the wrong places if their names aren't **chosen carefully**:

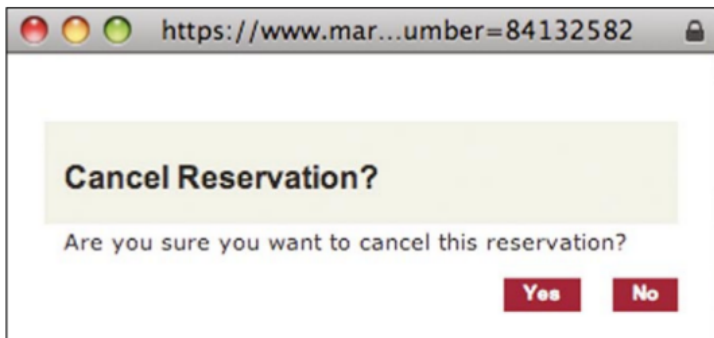


For each goal below, what on the screen would attract your attention?

- Pay a bill
- Transfer money to your savings account
- Pay your dentist by funds transfer
- Change your PIN
- Open a new account
- Purchase travelers' cheques

- The goal-seeking strategy of following information scent suggests that interactive systems should be designed so that the scent is strong and **really does lead users to their goals**.
- To do that, designers need to understand the goals that users are likely to have at each **decision point** in a task.
- For example, imagine that you want to **cancel** a reservation you made or a payment you scheduled.
- You tell the system to cancel it, and a confirmation dialog box appears asking if you really want to do that.
- **How should the options be labeled?**


- Unambiguous language and choices:



- Confusing information scent:

**Cancel Payment**

Wells Fargo - Direct Connect



Payee: [blurred]  
Amount: [blurred]  
Delivery Date: [blurred]  
Transmission Date: [blurred]  
Account Number: [blurred]  
Check Number: [blurred]

This payment can still be canceled.

Request Cancel Payment?

- People **know** that their attention is limited, and they act accordingly.
- While pursuing a goal, they take familiar paths whenever possible rather than exploring new ones, **especially when working under deadlines**.
- Taking familiar, well-learned routes can be done fairly automatically and does not consume attention and short-term memory.
- Once we learn one way to perform a certain task using a software application, we may **continue** to do it that way and **never discover a more efficient way**.
- Even if we discover or are told that there is a “better” way, we may stick with the old way because it is **familiar and requires little thought**.
- Avoiding thought when using computers is important: **people are willing to type more in order to think less**.



- This preference for familiar, relatively mindless paths has several design implications for interactive systems:
  - **Sometimes mindlessness trumps keystrokes.** With software intended for casual use or infrequent use, allowing users to become productive quickly is more important than saving keystrokes. Such software simply isn't used enough for key-strokes per task to matter much.
  - **Guide users to the best paths.** From its first screen or home page, software should show users the way to their goals. This is basically the guideline that software should provide clear information scent.
  - **Help experienced users speed up.** Make it easy for users to switch to faster paths after they have gained experience. The slower paths for newcomers should show users faster paths if there are any. This is why most applications show the keyboard accelerators for frequently used functions in the menu bar menus.

- Over many decades, scientists studying human behavior have found a cyclical pattern that seems to hold across a wide variety of activities:
  - 1 **Form a goal**, e.g. open a bank account or delete a word from a document.
  - 2 **Choose and execute actions** to try to make progress toward the goal.
  - 3 **Evaluate whether the actions worked**, i.e., whether the goal has been reached or is nearer than before.
  - 4 **Repeat** until the goal is reached (or appears unreachable).
- We saw a similar analysis in our discussion of Don Norman's model of thought in action.

- How can software support users in carrying out the goal-execute-evaluate cycle?
- Any of these ways:
  - **Goal:** Provide clear paths—including initial steps—for the user goals that the software is intended to support.
  - **Execute:** Software concepts (objects and actions) should be based on the **task** rather than the implementation. Provide clear information scent at choice points to guide users to their goals. Don't make them choose actions that seem to take them away from their goal in order to achieve it.
  - **Evaluate:** Provide feedback and status information to show users their progress toward the goal. Allow users to back out of tasks that didn't take them toward their goal.



## Orchestration and Flow

- Our goal is to make the people who use our products more productive, effective, and engaging.
- This means that we must ensure our systems are “mentally ergonomic”.
- That is, that they **support user intelligence and effectiveness** and avoid disrupting states of productive concentration.
- When people are able to concentrate wholeheartedly on an activity, they lose awareness of peripheral problems and distractions.
- The state is called **flow**, a concept first identified by Mihaly Csikszentmihalyi in *Flow: The Psychology of Optimal Experience*.

- In *Peopleware: Productive Projects and Teams*, Tom DeMarco and Timothy Lister describe flow as a “condition of deep, nearly meditative involvement.”
- A person in a state of flow can be **extremely** productive, especially when engaged in constructive activities such as engineering, design, or writing.
- To state the obvious: to make people more productive and happy, we should design interactive products that **promote and enhance flow**.
- If an application consistently rattles the user and **disrupts flow**, it becomes difficult to maintain that productive state.

- In most cases, interacting with software (especially business software) is a **pragmatic exercise**.
- If a user could achieve his goals magically, **without messing about with a user interface**, he would.
- Interacting with a lot of software will **never** be an entirely aesthetically pleasing experience (with some exceptions).
- Directing your attention to the interaction itself puts the emphasis on **side effects** rather than on goals.
- We must remember that the ultimate user interface for most purposes is **no interface at all**.

- To create a sense of **flow**, our interaction with software must become **transparent**.
- When a novelist writes well, the **craft of the writer becomes invisible**, and the reader sees the story and characters with clarity undisturbed by technique.
- When a product interacts well with a person, **interaction mechanics disappear**, leaving the person with his objectives and **unaware** of the intervening software.
- **Well-orchestrated** user interfaces are transparent, and the interaction designer must train himself to hear sour notes in the orchestration of software interaction.
- When an application's communication with a person is well orchestrated, **it becomes almost invisible**.



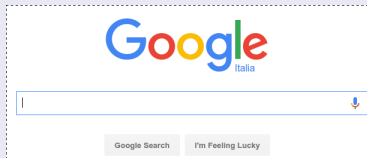
- There are no universal rules for designing **harmonious interactions**.
- However, there are some good **rules of thumb**:
  - 1 Follow users' mental models.
  - 2 Less is more.
  - 3 Enable users to direct, don't force them to discuss.
  - 4 Keep tools close at hand.
  - 5 Provide modeless feedback.
  - 6 Design for the probable; provide for the possible.
  - 7 Provide comparisons.
  - 8 Provide direct manipulation and graphical input.
  - 9 Reflect object and application status.
  - 10 Avoid unnecessary reporting.
  - 11 Avoid blank slates.
  - 12 Differentiate between command and configuration.
  - 13 Provide choices.
  - 14 Hide the ejector seat levers.
  - 15 Optimize for responsiveness; accommodate latency.

## 1. Follow users' mental models

- This one should be familiar to us by now.
- We should design interactions to **conform with how users think about objects and actions**.
- Can be highly **domain-specific**

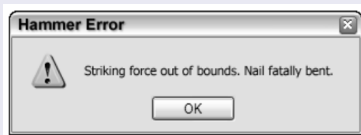
## 2. Less is more

- Reduce the number of elements in user interfaces without reducing the capabilities of the product.
- To do this, we must do **more** with **less**: we must control the power of the product without letting the interface become cluttered



## 3. Enable users to direct, don't force them to discuss

- This ideal interaction is not a dialogue – it's more like using a tool.
- When a carpenter hits nails, she doesn't discuss the nail with the hammer; she **directs the hammer** onto the nail.

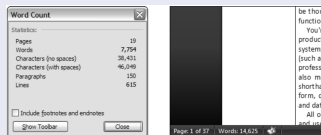


## 4. Keep tools close at hand.

- Most applications are too complex for **one** mode of direct manipulation.
- So applications offer a set of different tools to users – which is a **compromise with complexity**.
- Tools should be close at hand: on palettes or toolbars and accessible by keyboard command for expert users.

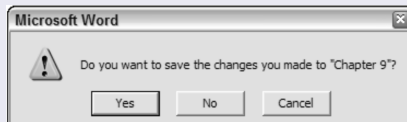
## 5. Provide modeless feedback.

- When interacting it's usually important to clearly present the status and effect of these manipulations.
- When possible, **feedback should be modeless**.



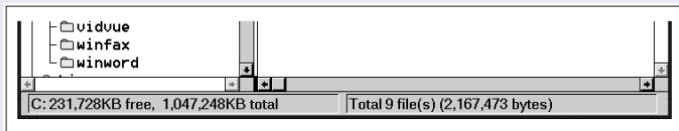
## 6. Design for the probable; provide for the possible.

- One of the most potent methods for better orchestrating your user interfaces is segregating the **possible** from the **probable**.



## 7. Provide comparisons

- Visual presentation expert Edward Tufte says that quantitative presentation should answer the question, “Compared to what?”
- Knowing that 231,728 KB are free on your disk is less useful than knowing that it is 22% of the **total** capacity.
- **Show the data**, rather than simply telling about it textually or numerically.



## 8. Provide direct manipulation and graphical input.

- Few interfaces let a user **draw** numerical input using appropriate graphical widgets.
- An exception: most modern word processors let you set tabs and indentations by dragging a marker on a ruler.

## 9. Reflect object and application status.

- When the application is engaged in some significant internal action, it should be **obvious** that it won't be as responsive as usual.
- If the interface is sending a large email, we should see a small representation of it uploading and sending (in a **modeless** progress bar).

## 10. Avoid unnecessary reporting.

- For programmers, it is important to know exactly what is happening in a program.
- Nontechnical people may be **alarmed** to hear that the database has been modified.
- It is better for the application to **just do what has to be done**, issue reassuring clues when all is well, and not burden users with the trivia of how it was accomplished.
- In a similar vein: **don't use dialogs to report normalcy**.

## 11. Avoid blank slates.

- It's easy to assume **nothing** about what your users want, and instead ask **questions** to determine desires.
- Users prefer application that give them what it **thinks** is right and are willing to manipulate settings to make it exactly right.
- When in doubt: **ask for forgiveness, not permission**.

## 12. Differentiate between command and configuration.

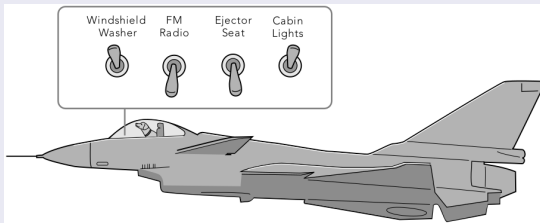
- Many applications lack differentiation between functions and **configuration** of those functions.
- If you ask an application to perform a function, the application should simply **perform** it and **not interrogate** you about configuration details.
- When you ask many applications to print a document, they launch complex dialog boxes asking how many copies, what the paper orientation is, what paper feeder to use, etc.

## 13. Provide choices.

- Contrary to what many software developers think, questions and choices don't necessarily make users feel **empowered**.
- More commonly, they make people feel **badgered** and **harassed**.
- Users don't like to be asked questions, it cues a user that the application is:
  - Ignorant
  - Forgetful
  - Weak
  - Lacking initiative
  - Unable to fend for itself
  - Fretful
  - Overly demanding



## 14. Hide the ejector seat levers.



## 15. Optimize for responsiveness; accommodate latency.

- There isn't much that is more disturbing to flow than staring at a screen **waiting** for the computer to respond.
- In a number of studies dating back to the late 1960s, it's been found that user perception of response times can be roughly categorized:
  - Up to 0.1 seconds, users perceive the system's response to be **instantaneous**. Here, they feel that they are directly manipulating the user interface and data.
  - Up to about 1 second, users feel that the system is responsive. Users will likely notice a delay, but it is small enough for their thought processes to stay **uninterrupted**.
  - Up to about 10 seconds, users clearly notice that the system is slow, and their mind is likely to wander, but they are capable of maintaining some amount of attention on the application. Providing a **progress bar** is critical here.
  - After about 10 seconds, you will lose user attention. Processes that take this long should be conducted **offline** or in the **background**, allowing users to continue with other work. Status and progress should be communicated, including estimated time remaining, and a cancel mechanism is critical.

## The way forward

- Today we saw some general **design principles** that we can use to **maintain flow**.
- In the next lecture we will finish this discussion with an overview how to **eliminate excise** in interfaces.
- We will then begin a discussion of metaphors and idioms (and what makes **good idioms**).
- And then we will see how to translate our **Gestalt** view of organization into rules of thumb for visual composition of graphical interfaces.
- We will also begin **brainstorming** ideas for final projects (including some presentations of past projects).
- In the **laboratory** on Wednesday, we will begin work on mini-project using (eventually) the **MVC** model.

# Homework

NONE