Human Computer Interaction Notes

Saiberspeis

June 7, 2020

Contents

1 HCI Foundation 1 - Human

The first, important thing we have to mention when it comes to Human-Computer Interaction is the **human**. We can talk about the human as a system where several information are sent outside and received inside (I/O like visual, auditory, movement...), stored (sensory, short and long term memory...) and processed and applied (reasoning, problem solving...). Emotions can influence human capabilities and such influence depend on the single human being as each person is different. Let's see some of these capabilities.

1.1 Vision

Vision is divided in two stages:

1. physical reception of stimulus
2. processing and interpretation of stimulus

The eye is the physical receptor for vision, obviously. It is a mechanism for receiving light and transforming it into electrical energy for the brain, in fact images are focused upside-down on retina and correctly processed by ganglion cells (present in our brains) which detect pattern and movement. Some parameters that define the characteristics of what we are seeing are:

- Size and depth (visual angle, acuity, cues like overlapping objects which help perception...);
- Brightness (subjective reaction to its levels, affected by luminance of objects...);
- Color (made up of hue, intensity, saturation, different acuity for colors, color-blind people problem...).

However, the visual system is also able to compensate for movement or changes in luminance, but sometimes there is overcompensation which causes optical illusions on some images or videos and so on. Context is resolved to solve visual ambiguity.

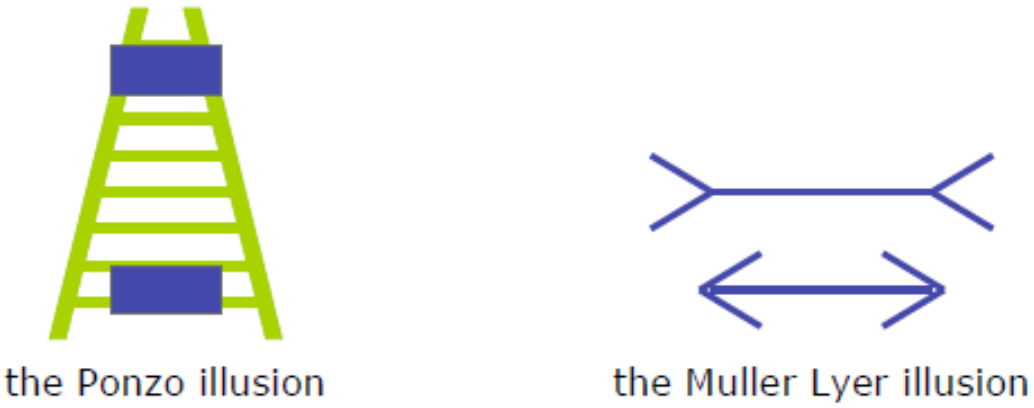


Figure 1: Optical illusions examples.

Reading is something specific we can do with our visual system: in fact we perceive a visual pattern then we give it a meaning with respect to a specific internal representation of language (knowledge of syntax, semantics, grammar...). Word shape is important to recognition and negative contrast is necessary to perceive words and symbols on a computer screen (in fact we always set a light or dark theme with respect to our tastes: the important is to be able to distinguish letters and numbers on the screen we are using!).

1.2 Hearing

Hearing is done thanks to our ears and provides information about environment, like distances and directions of the heared sound. The physical apparatus interested is so composed:

- outer ear, which protects inner and amplifies sound;
- middle ear, which transmits sound waves as vibrations to inner ear;
- inner ear, where chemical transmitters are released and cause impulses in auditory nerve.

The signal we receive is called *sound* and is composed by pitch (sound freq.), loudness and timbre. However, humans can hear frequencies only from 20Hz to 15kHz, so not all the frequencies. Auditory system is capable of filtering sound, distinguishing the sound we are interested in from background noise and other useless stuff (cocktail party phenomenon!).

1.3 Touch

Touch provides important feedback about environment, more than other senses. May be the main sense for environment for someone who is visually impaired! Stimulus are received via receptors in the skin and we have different sensibility with respect to the body part (fingers are more sensible!).

1.4 Movement

Time taken to respond to stimulus is composed by reaction time + movement time. This time depends on age, fitness and other factors and also on stimulus type (visual = 200ms, auditory = 150 ms, pain = 700 ms...). *Fitts' Law* describes the time taken to hit a screen target:

$$Mt = a + b \log_2(D/S + 1) \tag{1}$$

where:

- a and b are empirically determined constants;
- Mt is movement time
- D is distance
- S is size of target

That's why we have to project UIs with targets as large as possible and distances as small as possible.

1.5 Memory

There are three types of memory that we'll now examine:

- Sensory memory: buffers for stimuli received through senses (iconic, echoic, haptic...), continuously overwritten;
- Short-Term memory (STM): scratch-pad for temporary recall with rapid access and decay, and limited capacity;
- Long-Term memory (LTM): repository for all our knowledge (slow access and decay, huge or unlimited capacity). Two types: episodic and semantic. The latter derive from the former. Episodic is related to events we remember, semantic instead provides access to information and is something we use during exams, etc. Semantic LTM represents relationships between bits of information and follows a specific model, called semantic network presented in the next image.

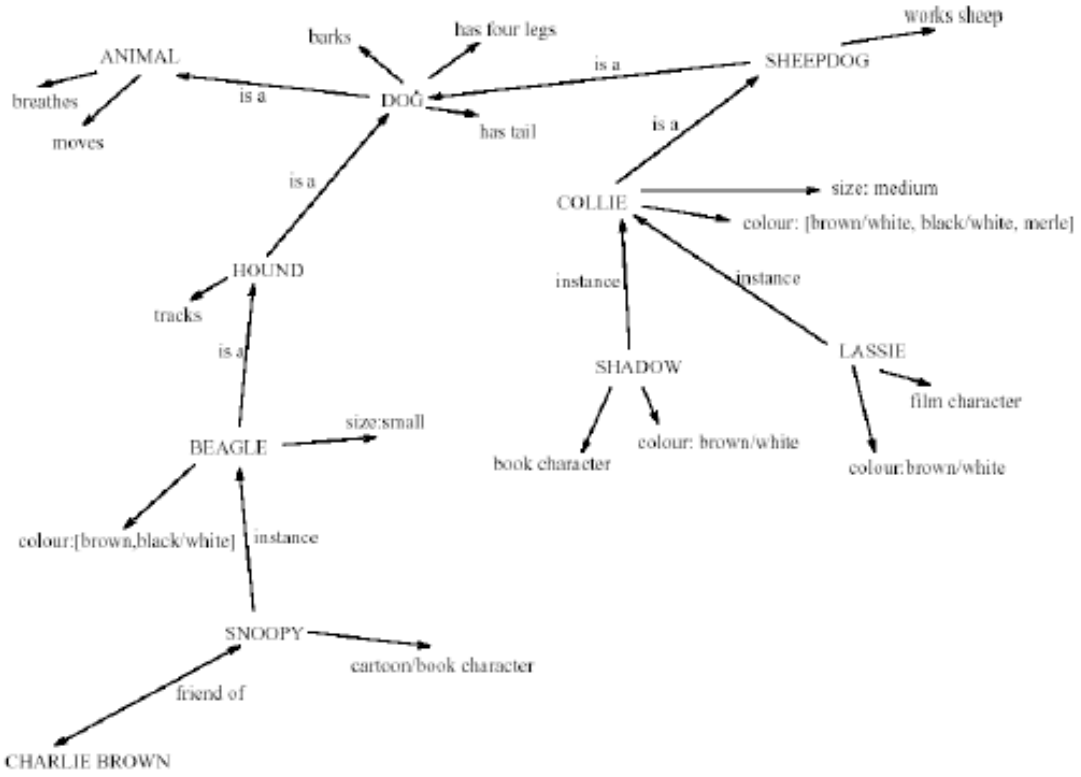


Figure 2: Semantic network.

Information in LTM are organized in data structures called *frames* with some fields as we can see here:

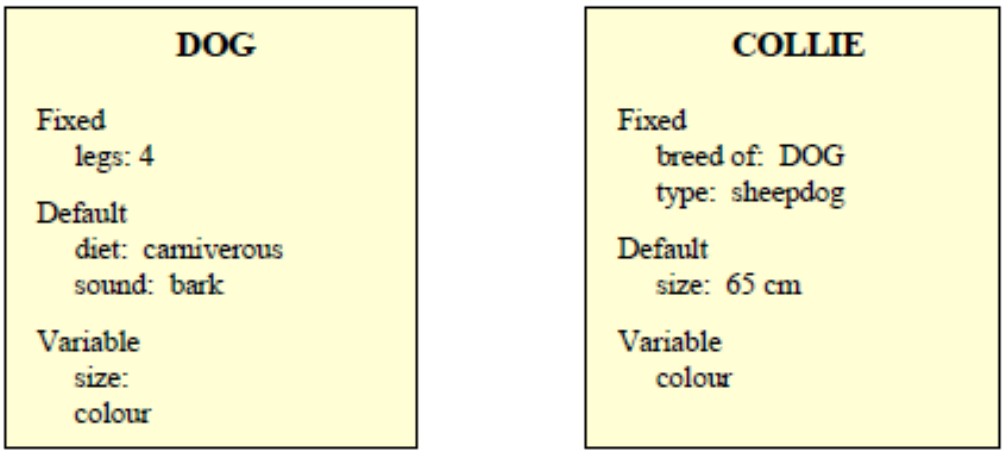


Figure 3: LTM frames.

We can also use *scripts*, which are richer than frames because they not only model knowledge but also the process in which its components are involved.

Script for a visit to the vet

Entry conditions:	*dog ill* *vet open* *owner has money*
Result:	*dog better* *owner poorer* *vet richer*
Props:	*examination table* *medicine* *instruments*
Roles:	*vet examines* *diagnoses* *treats* *owner brings dog in* *pays* *takes dog out*
Scenes:	*arriving at reception* *waiting in room* *examination* *paying*
Tracks:	*dog needs medicine* *dog needs operation*

Figure 4: LTM scripts.

Production rules are instead representation of procedural knowledge, i.e. condition/action

rules.

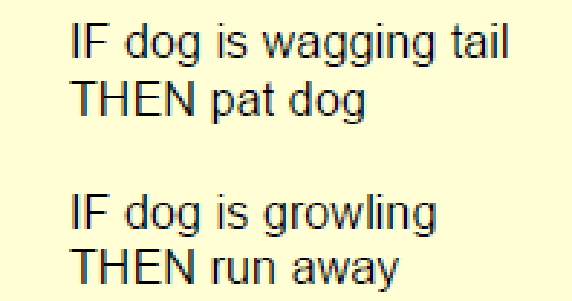


Figure 5: LTM production rule.

From the point of view of actions, what we can do with LTM? We can store information, moving them from STM to LTM and also distributing the learning over time, and giving it a structure to improve such a movement. The immediate consequent action is the retrieval of such information, through recall (remember) or recognition of something we see/hear/experience and so on. Last but not least, we can forget information, for decay (it happened long time ago) or interference (overwriting of info in our brain).

1.6 Thinking

Thinking involves reasoning (deduction, induction, abduction) and problem solving. Let's see them more in detail.

- *Deduction* means derive logically necessary conclusion from given premises/experiences. Logical conclusions not necessarily true (think about syllogisms...). As human knowledge is different from logical deduction algorithms, we bring such knowledge in order to understand if an info is real or not;
- *Induction* consists of generalizing from cases seen to cases unseen, i.e. from particular to general. However it can prove only false and not true, so it is unreliable but still useful. Humans are not good at using negative evidence (Wason cards example!).

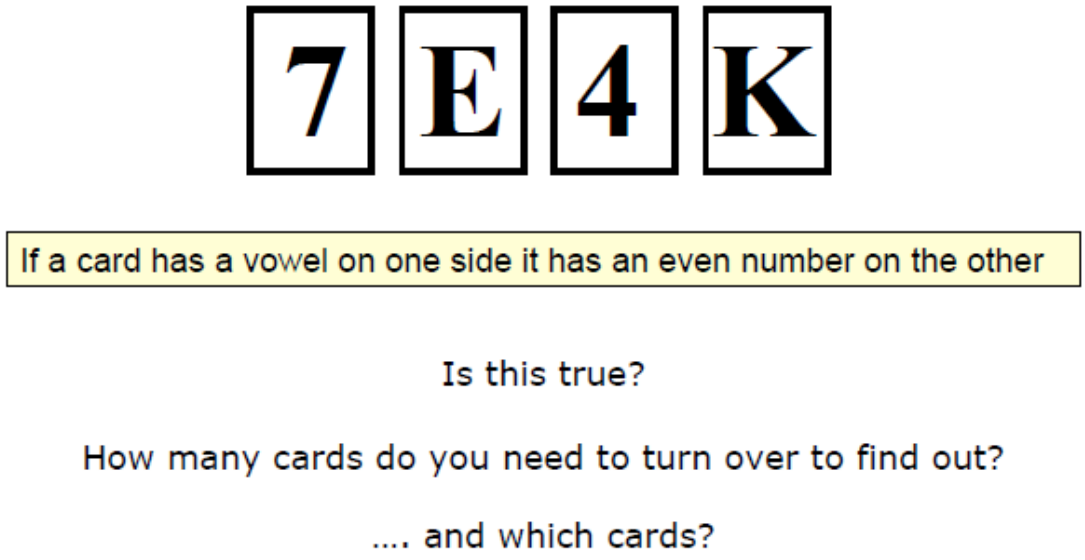


Figure 6: Wason's cards.

The answer is 7 and E, where 7 is the negative evidence. This test shows that humans are not good in using negative evidences (we would have turned over E only, probably).

- *Abduction* is reasoning from event to cause and it is unreliable because it can lead to false explanations.
- *Problem Solving* is the process of finding solution to unfamiliar tasks using knowledge and it is supported by several theories. The problem space theory says that the problem space comprises problem states and the process of problem solving involves generating states using legal operators. Heuristics may be employed to select operators and operates within human information processing system. This model is largely applied to problem solving in well-defined areas. We use *analogy* and *skill acquisition* when we have to solve a problem. However, this is not a perfect method: we can do errors (slips, i.e. right intention but failed attempt, or mistakes, i.e. wrong intention caused by incorrect understanding).

1.7 Emotion

Emotion largely affects human capabilities and it is also a capability itself. There are several theories on how emotion works, like "emotion is our interpretaion of a psychological response to stimuli" by James-Lange. Emotion clearly involves both cognitive and physical responses to stimuli so if we design a system for being emotion-resilient, we have to recreate these emotions also during testing phase. The biological response to stimuli is called *affect*, which influences how we respond to situations (positively or negatively). Stress will increase the difficulty for problem solving and relaxed users will be more forgiving of shortcomings in designs, so the more the user

will be stressed, the easier our UI should be in order to give a better UX (aestethically pleasing and rewarding interfaces...).

1.8 Conclusions

There are several differents in each human being like sex, physical and intellectual abilities, age, stress effect and so on. Hence, we have to consider the psychological aspect when designing a system, modeling it with respect to these differences.

2 HCI Foundation 2 - Computer

Nowadays, computers pervade our lives and we regularly interact with them. A computer is made up of various elements such as input devices (text entry and pointing), output devices (screen, digital paper...), memory (RAM, ROM, HDD...) and processing (speed of processing, networks...). In order to understand human-computer interaction we need to understand computers, i.e. what I can do with them. Computers we use nowadays are quite similar, in fact they have a screen, keyboard and pointing device. Some variations can affect the whole system concept such as desktop PCs, laptops and so on. However, each type supports different style of interaction and purpose. There are also small computers in our washing machines, microwaves and so on. Some years ago there was not so much interactivity, but nowadays almost every computer system is interactive. Let's see how we can interact with a computer.

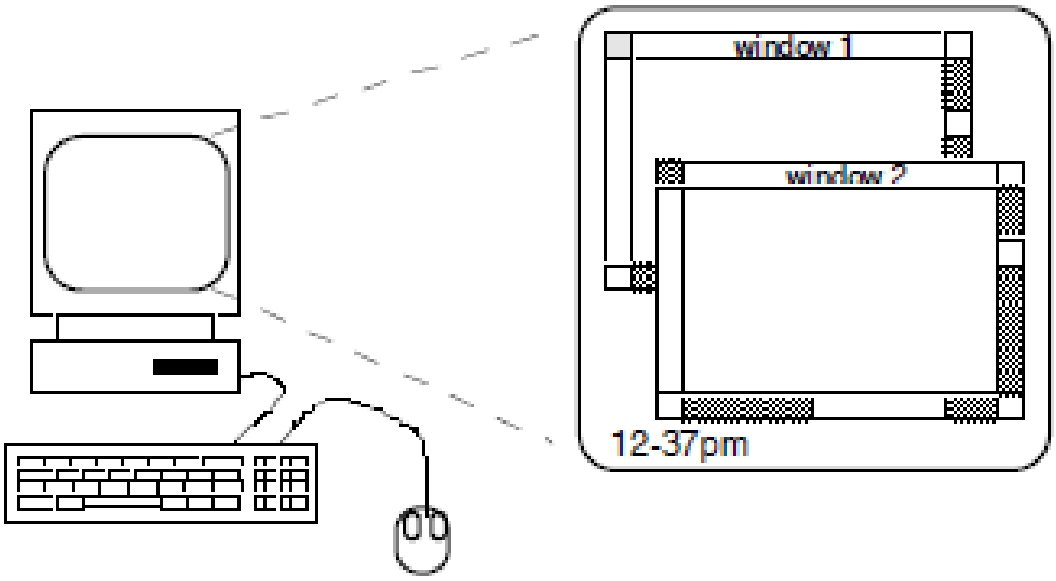


Figure 7: Computer scheme.

2.1 Writing Devices

2.1.1 Keyboards

Keyboards are the most common input devices and allow rapid entry of text by experienced users. Keypress closes connection, causing a character code to be sent and it is usually cable connected but can also be wireless. The most famous layout is the QWERTY one that changes with respect to the country. However, there are also other layouts such as alphabetic orders, etc. or special keyboards for left handed use, phone keyboards with T9 criterion, etc. There are also numeric keypads, i.e. with numbers only.

2.1.2 Handwriting Recognition

Text can be input into the computer, using a pen and a digesting tablet. There are technical problems in this approach, like capturing all useful informations, interpreting individual calligraphies and individual letters from words. This is an approach used on tablets and PDAs.

2.1.3 Speech Recognition

Speech recognition is most successful when there is a single user and limited vocabulary, for obvious reasons. Problems here are related to external noise interfering, pronunciation errors, large vocabularies and different voices.

2.2 Pointing Devices

2.2.1 Mouse

Mouses are handeld pointing devices, very common and easy to use. They permit a planar movement on a display and in the simplest version they have two buttons and a small wheel. They require little physical space and do not cause arm fatigue. Pointers also do not hide screen portions so it is something really useful for human-computer interaction. They work in two main methods:

- Mechanical, older mouses with a sphere on underside of mouse, but usable in most surfaces;
- Optical, with a light emitting diode on underside of mouse and less susceptible to dust and dirt, but sometimes causing issues with certain surfaces.

There have been also experiments with feet (footmouse) but it is something not really common, obviously :D however, foot controls are common elswhere, like in cars or piano pedals.

2.2.2 Touchpad

Touchpads are mostly present in laptop computers and they need a stroke to move mouse pointer. They are small touch sensitive tablets and need stroke settings to work properly (fast or slow).

2.2.3 Joystick

They are more common for computer games and they use pressure of stick for movement and buttons for selection.

2.2.4 Touch-Sensitive Screen

This kind of screen detects interactions through the presence of finger or stylus on the screen.
Advantages:

- fast
- requires no special pointer
- suitable for use in hostile environment

Disadvantages:

- finger can mark the screen
- imprecision
- lifting arm can be tiring

Stylus and light pens are devices which help this kind of interaction and are heavily used in tablets. These pens can also write (handwriting) and perform other interesting tasks, although they can obscure the screen a little bit.

2.3 Display Devices

2.3.1 Bitmap Displays

Bitmap displays are the more common, the ones composed by a vast number of colored dots. We list the main characteristics of such displays:

- Resolution (pixels)
- Aspect Ratio (16:9,...)
- Color Depth

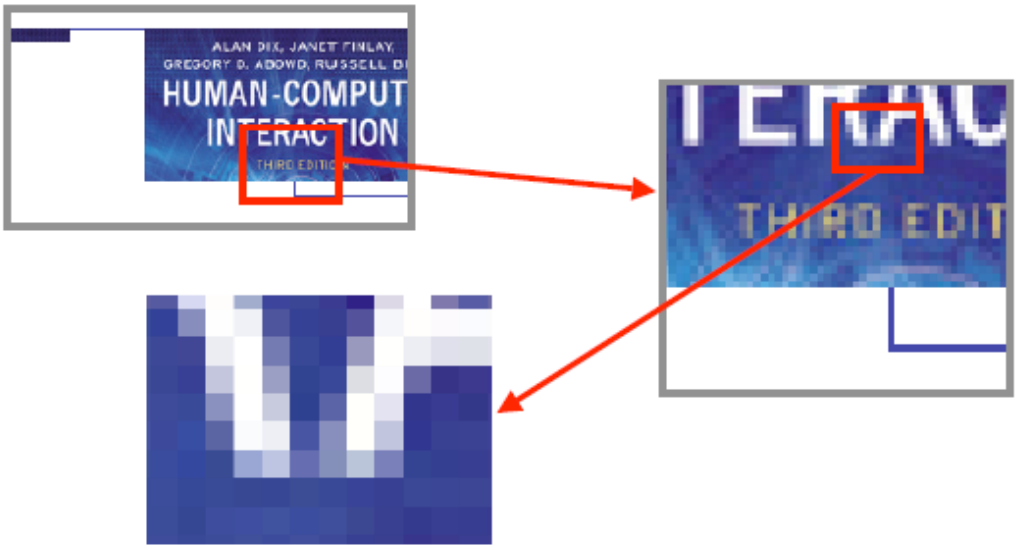


Figure 8: Bitmap display.

2.3.2 Cathode Ray Tube

The system of old TVs: stream of electrons emitted from electron gun, focused and directed by magnetic fiels, hit phosphor-coated screen which glows.

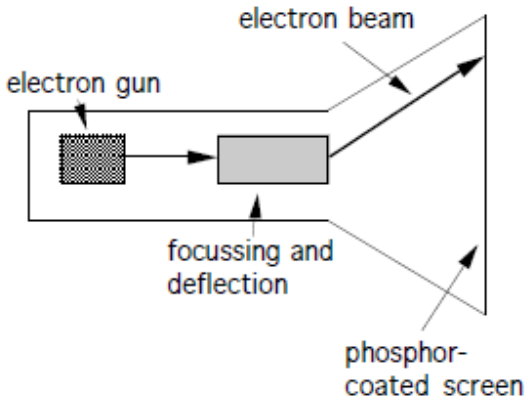


Figure 9: Cathode ray tube.

This technology declined because of the monitors size. It also had some health hazards, like radio frequency emissions and electrostatic and electromagnetic fields related problems.

2.3.3 Liquid Crystal Displays

They replaced CRT monitors because of the smaller size, lighter images and no radiation problems. We find them everywhere nowadays.

2.3.4 Large Displays

They are used for meetings, lectures and so on. Technologies for these displays range from video walls, projected, back-projected etc.

2.3.5 Situated Displays

These displays are put in public places and are sometimes interactive, sometimes not. In all the cases when implementing these displays, the location matters. Think about displays in train stations, etc.

2.4 VR and 3D Interaction

2.4.1 Positioning in 3D Space

In 3D space, positioning is done differently:

- cockpit and visual controls like steering wheels, knobs and dials, like in real life with cars and planes etc.
- the 3D mouse is a mouse allowing 6 directions of movement
- data glove uses fiber optics to detect finger position
- VR helmets
- whole body tracking

Movement is done in three directions through pitch, yaw and roll.

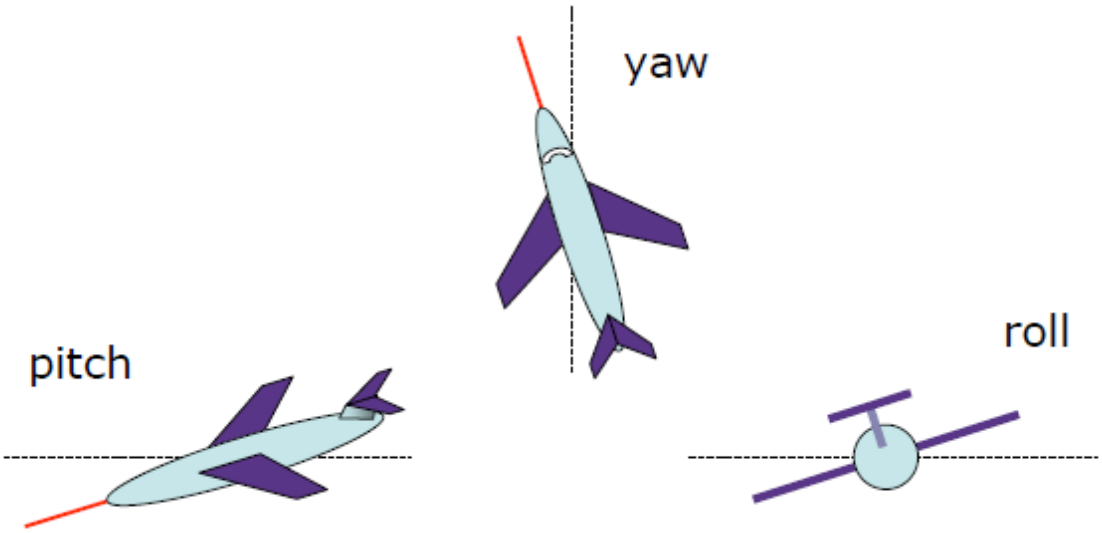


Figure 10: Pitch, yaw and roll.

2.4.2 3D Displays

3D displays are different too with respect to the previous ones, in fact we have:

- Desktop VR: ordinary screen, mouse or keyboard control but perspective motion give 3D effect
- Seeing in 3D: through helmets or special glasses etc.

Note that sometimes these devices can bring to sickness (like head spinning) so technology is trying to improve in this sense.

2.5 Physical Controls, Sensors, Etc.

2.5.1 Dedicated Displays

In dedicated displays, representations can be analog or digital. Head-up displays instead can be found in aircraft cockpits and show most important controls depending on context.

2.5.2 Sound

Beeps, bongs, clonks, whistles and whirrs used for error indicatons or confirmation of actions.

2.5.3 Touch, Feel, Smell

They are used by humans to interact with AR/VR systems, like in games and simulation. Touch and feeling are important, while smell and taste experiment very limited technology currently.

2.5.4 Environmental and Bio-Sensing

This part include all the sensors around us (car courtesy light, ultrasound detectors...) and even our own bodies (body temperature, heart rate...).

2.6 Paper, Printing and Scanning

2.6.1 Printing

Printing is the action of representing something on paper or physical entity, from a digital one. Images made of small dots are the most common ones and their critical features are resolution, speed and cost. Dot-based printers are the most common and are dot-matrix printers (line of pins that can strike the ribbon, dotting the paper), ink-jet and bubble-jet printers (tiny blobs of ink sent from print head to paper), laser printers (like photocopier, where dots of electrostatic charge are deposited on drum which picks up toner rolled onto paper which is then fixed with heat. Thermal printers instead are more common in workplace.
Every letter we see printed or also on the screen has a font which can change the readability of text through its features (size, type,...).

2.6.2 Scanners

Scanners are devices which take paper and convert it into a file (bitmap) so it is the contrary of printing. They are used in desktop publishing for incorporating photographs and other images, document storage and retrieval systems, special scanners for slides and photographic negatives. Note that paper can also be used as input for certain programs, thus we call it paper-based interaction.

2.7 Memory

2.7.1 Short-term Memory (RAM)

Random Access Memory is on silicon chips and is accessed and used at runtime, then erased when turning off the PC, so it is volatile. Some non volatile RAMs are used to store basic set-up information. They can be found on typical desktop computers.

2.7.2 Long-term Memory (Disks)

Magnetic disks of every kind and optical ones. They can also be found in nowadays PCs and computer systems.

2.7.3 Flash Memory

Silicon based but persistent, usually for USB devices.

There are some relevant features related to memory, like methods of access, format, size, compression which can affect the speed and capacity of the memory.

2.8 Processing and networks

Designers tend to assume fast processors and make interfaces more and more complicated. However, that's not the right thing to do as there are a lot of old processors still running in production use nowadays. There are also problems if system is too fast, like excessive text scroll speed. On interactive performance, we always have to assume some limitations like in computation, storage channel, graphics and network. In this way we will be able to design an UI correctly.
Networked computing is instead everything which allows access to large memory and processing, other people (email, chats...) and shared resources. Anyway, it has some issues too, like delays, conflicts and unpredictability. Internet is a great example of such a thing.

3 HCI Foundation 3 - Interaction

What is the so-called interaction? *Interaction* is the communication between user and system, basically. It is quite simple, but it contains several concepts and components that we will analyze in this section.

3.1 Models of Interaction

There are several models that describe interaction modes and situations, but now we will analyze the two most relevant ones.

- *Donald Norman's Model*: it's a model which consists of seven stages

  1. user establishes the goal
  2. formulates intention
  3. specifies actions at interface
  4. executes action
  5. perceives system state
  6. interprets system state
  7. evaluates system state with respect to goal

  This model concentrates on user's view of the interface. A fast-and-easy way to see such a scheme is the execution/evaluation loop:

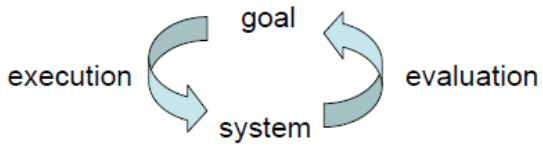


Figure 11: Execution(2-4)/evaluation(5-7) loop.

  Remember that some systems are harder to use than others, i.e. user's formulation of actions is different from actions allowed by the system, and user's expectation of changed system state is different from actual presentation of this state.
  For fixing human errors, we improve interface design to avoid slips, and improve system understandability for avoiding mistakes.

- *Abowd and Beale framework:* an extension of Norman, composed by four parts:
  - user
  - input
  - system
  - output

Each part has its own unique language and the interaction consists of translation between languages.

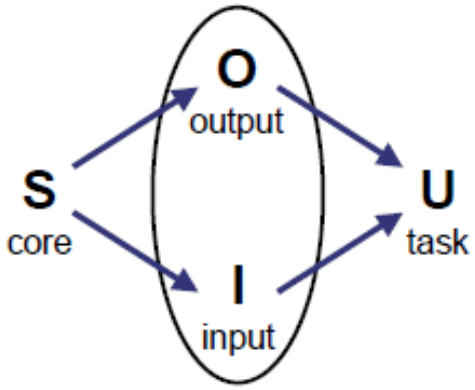


Figure 12: Abowd and Beale framework scheme.

In this model, user intentions are translated into actions at the interface, translated into alterations of system state, reflected in the output display, interpreted by the user.

3.2 Ergonomics

Ergonomics is the study of the physical characteristics of interaction; it is good at defining standards and guidelines for constraining the way we design certain aspects of systems. Some examples of ergonomics are: use of color, surrounding environment, health issues,...
Industrial interfaces are different from office ones, so we have to think about which environment we are interested in. Glass interfaces are used nowadays in industrial contexts: these interfaces are cheaper and more flexible, also because they contain multiple representation of same info.

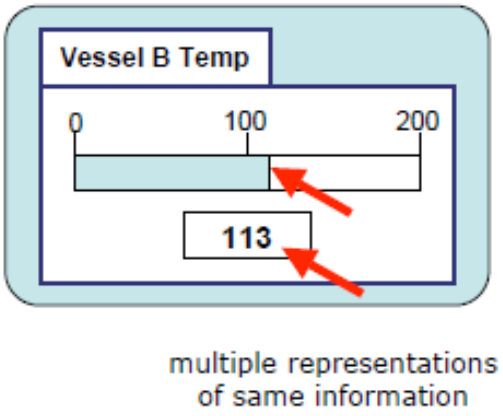


Figure 13: Glass interface example.

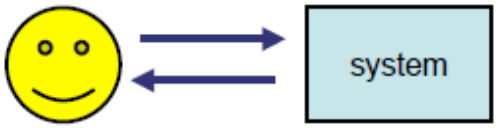


Figure 14: Direct manipulation: user interacts with artificial world.

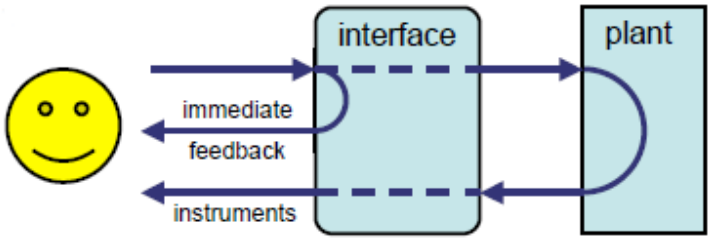


Figure 15: Indirect manipulation: user interacts with real world through interface.

3.3 Interaction Styles

There are several interaction styles such as command line interface, menus, natural language and so on. Now we describe them in detail.

- *Command line* is a way of expressing instructions to the computer directly, it is suitable for repetitive tasks but better for expert users than novices. It offers direct access to funcionality and it is usually more powerful than a standard UI. Commands and abbreviations should be meaningful for its use.
- *Menus* are a set of options displayed on the screen, that can be selected. Visible options are selectable by numbers, letters, arrow keys, mouse and so on. Often, options are hierarchicallly grouped.
- *Natural language* is familiar to user, so we mean speech recognition or typed natural language. Problems are that natural language can be vague, ambiguous and solutions could be trying to understand a subset, picking key words.
- *Query interfaces* are question/answer interfaces suitable for novice users but with limited functionality. Query languages are used, such as SQL when dealing with databases.
- *Form-fills* are primarily for data entry or data retrieval and screen looks like paper form. There are several labels and text entry fields.
- *Spreadsheets* are a sophisticated variation of form-filling, like grid of cells that contain a value or formula, etc.

- *WIMP interfaces* are interfaces composed by Windows, Icons, Menus and Pointers, that's why they are so called. M and P can also be Mice and Pull-down menus. This is the default style for majority of interactive computer systems, especially PCs and desktop machines.
- *Point and Click interfaces* instead, are used in multimedia, web browsers and hypertext. We just have to click something to get it work. Typing is minimal.
- *3D interfaces* are the ones used in VR but also the UIs where light and occlusion give depth, simply.

3.4 Context

Context affects interaction and includes other people, motivation and inadequate systems. These are its components.

3.5 Experience, Engagement and Fun

When we design a UI, we have to think about User eXperience, UX. Let's see a small difference between a real cracker and a virtual one, just to understand a simple experience.

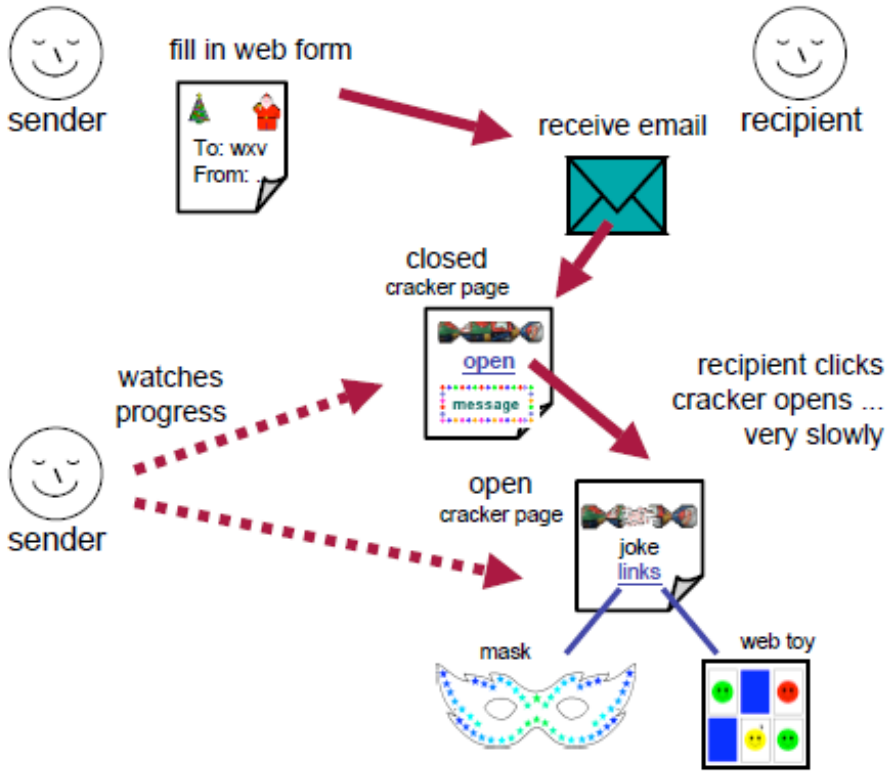


Figure 16: How crackers work.

	real cracker	virtual cracker
Surface elements		
design	cheap and cheerful	simple page/graphics
play	plastic toy and joke	web toy and joke
dressing up	paper hat	mask to cut out
Experienced effects		
shared	offered to another	sent by email message
co-experience	pulled together	sender can't see content until opened by recipient
excitement	cultural connotations	recruited expectation
hiddenness	contents inside	first page - no contents
suspense	pulling cracker	slow ... page change
surprise	bang (when it works)	WAV file (when it works)

Figure 17: Crackers experience.

In physical design, there are many constraints (ergonomic, physical, context related, aesthetic...) and they are often in contrast, so we have to find a tradeoff (let's think about the contrast between ergonomics and physical aspect!).
When designing a UI we have to make sure that physical aspects reflect logical ones (e.g. on/off buttons) and put inverse actions, like open call/close call on old phones keyboard and on washing machines buttons.
General lesson: if you want someone to do something (and to improve your product's UX), make it easy for them and understand their values!

4 Mobile Design Patterns

In order to understand if a mobile app has a good or bad mobile design, it is useful to read bad reviews on app stores because they give a lot of information about how we can improve such an app. In fact, good reviews are usually not useful for this purpose.
One way to approach the design of UIs is to learn from examples that have proven to be successful in the past. *Design patterns* are solutions to a recurrent problem within a specific application domain. We now examine each of them in detail.

4.1 Navigation

It is about how users move through the app views. Apps with good navigation just feel simple and make it easy to accomplish any task. We divide such patterns in primary and secondary.

4.1.1 Primary Navigation Patterns

They consist of navigation from one primary category to another (like in top-level menus of a desktop app); they are divided into

- *Persistent Navigation* which concerns interactive navigation components that are permanently visible
- *Transient Navigation* that must be explicitly revealed with a tap or a gesture.

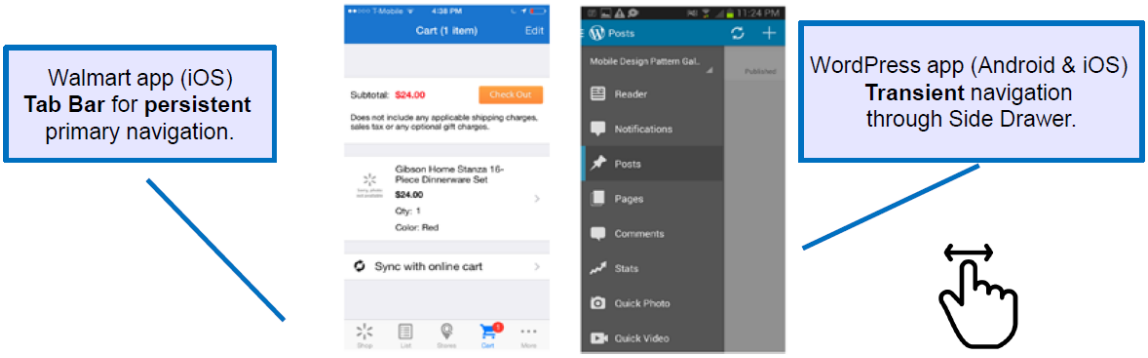


Figure 18: Primary Navigation Patterns.

For persistent patterns, we have a series of elements that we will be now explained in detail.

- *Springboard* is a landing screen with options that act as launch points into the application, using a grid layout for items of equal importance, or irregular layout to emphasize some items more than others.

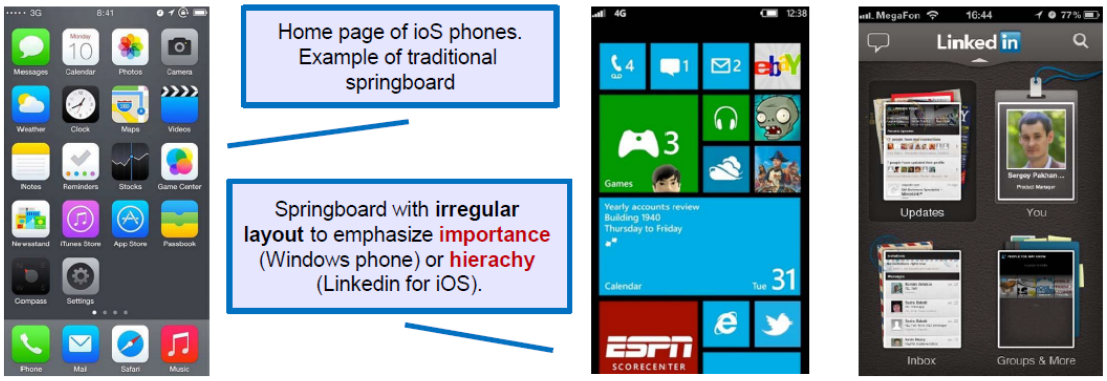


Figure 19: Springboard Layouts.

- *List Menu* is a view where each list item is a launch point into the application, and switching modules requires navigating back to the list. Expanding lists are the ones that expand with respect to the user's action, like in Android's CollapsingToolbarLayout.

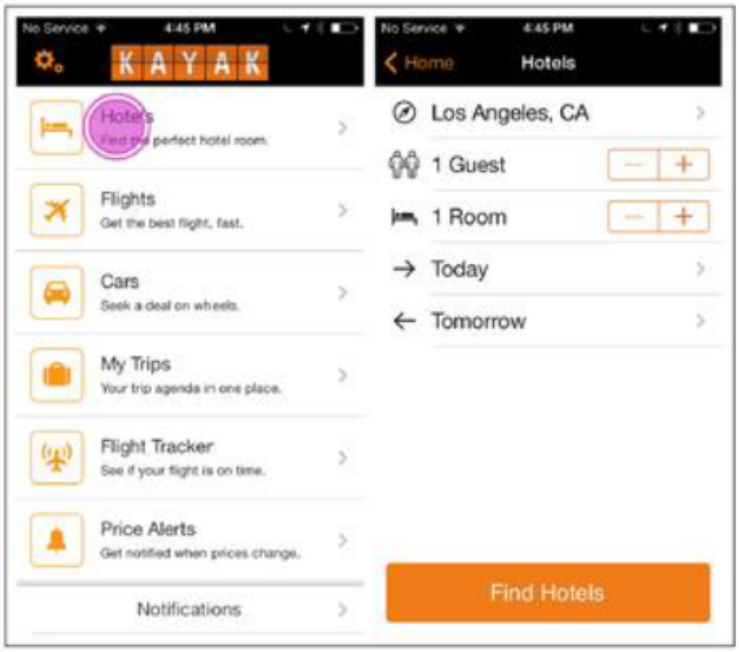


Figure 20: List Menu.

- *Cards* are small containers that logically encapsulate relevant information. Card navigation is based on a card deck metaphor, often including common card deck manipulations such as stacking, shuffling, discarding, and flipping. They are good for presenting similr objects whose size or supported actions can vary considerably.

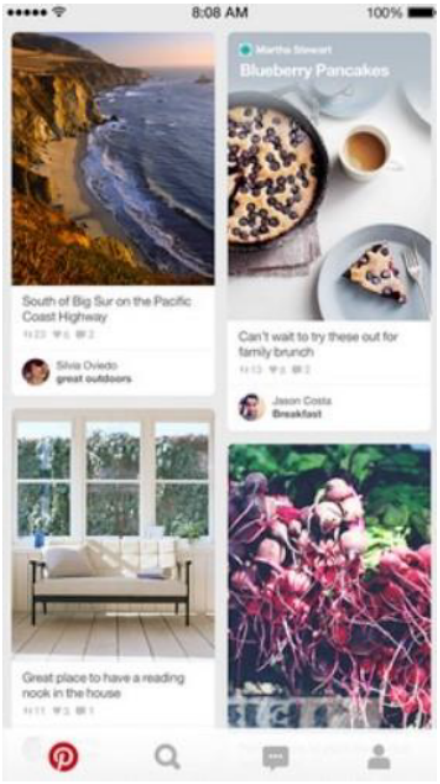


Figure 21: Cards.

- *Gallery* pattern displays live content (news, recipes, movies or photos) arranged in a grid or a carousel.

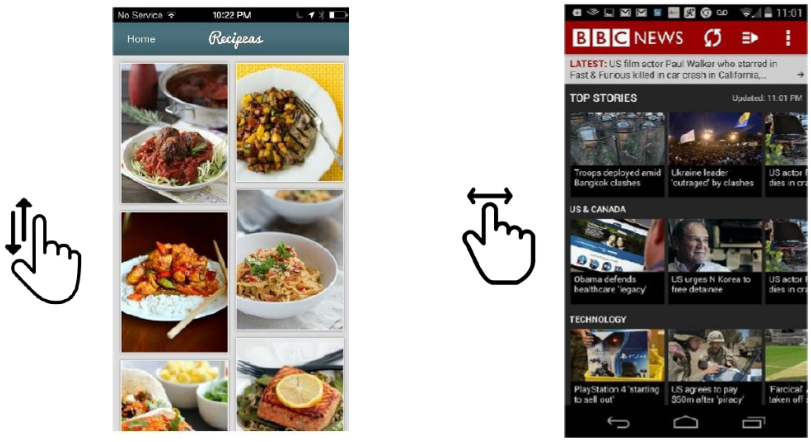


Figure 22: Gallery, grid (left) and carousel (right).

- *Tab* bars are suggested to navigate flat information structures: users can navigate directly from one primary category to another because all primary categories are accessible from the main screen.

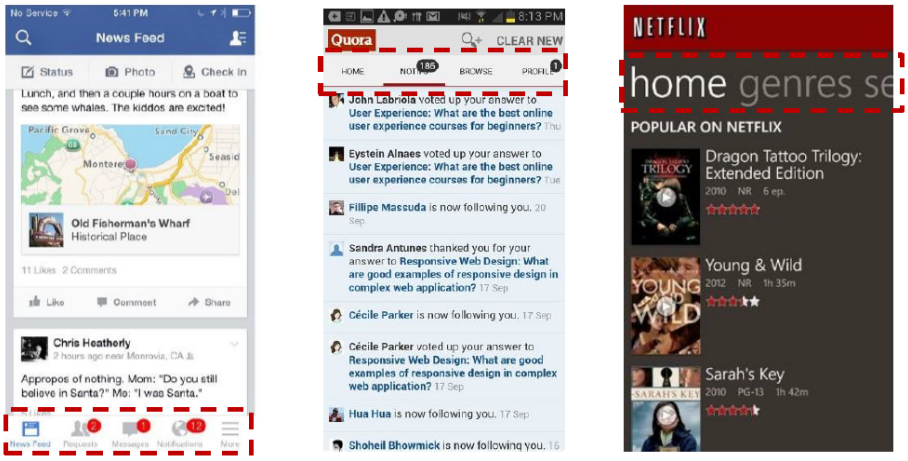


Figure 23: Tabs.

- *Metaphor* is characterized by an interface designed to match its real-world counterpart.

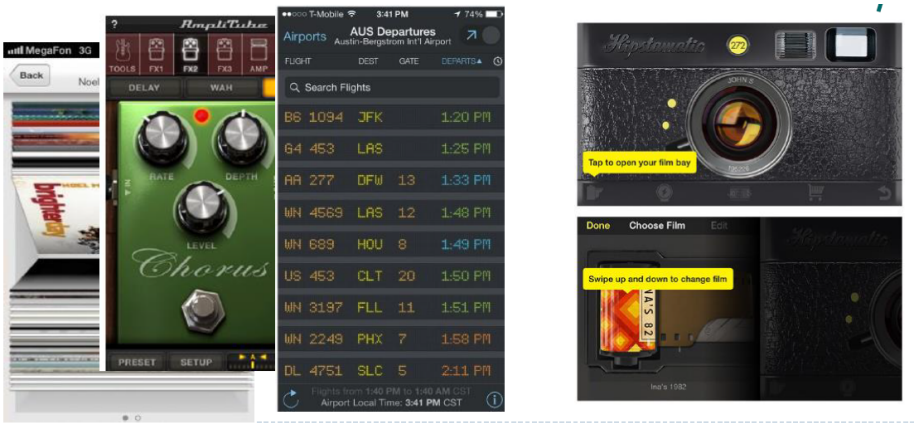


Figure 24: Metaphors.

- *Infinite Area* is a pattern where the entire data set can be considered to be a large, navigable 2D graphic, like it happens in Google Maps.

For transient patterns instead, we have to start from the meaning of the word "transient": it means "staying a short time", which is exactly how such navigation components work. They are hidden until we reveal them; then we make a selection and they disappear again. Let's examine them singularly:

- *Retracting Tab*, when tab bar collapses or appears when the user is scrolling or swiping through content;

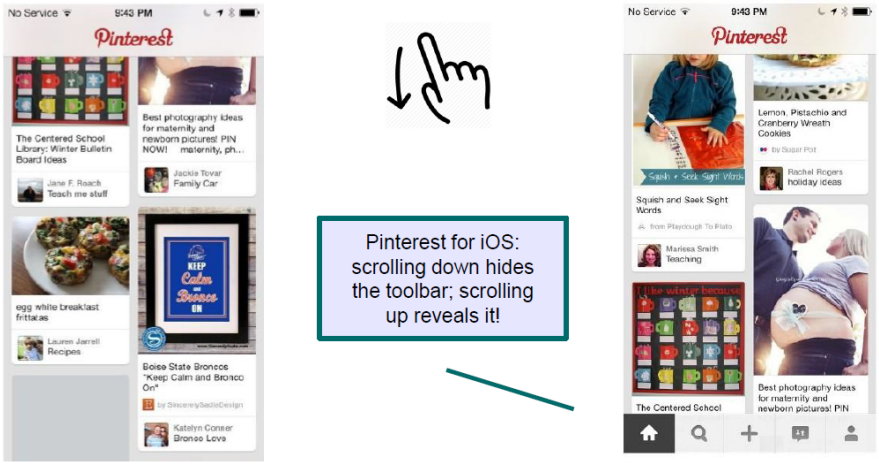


Figure 25: Retracting Tab.

- *Side Drawer*: there are two styles of Side Drawers:
  - *overlay*: a swipe gesture will reveal a drawer that partially covers or overlaps the original screen content
  - *inlay*: a swipe gesture will open a drawer that pushes the original screen content partially off-canvas

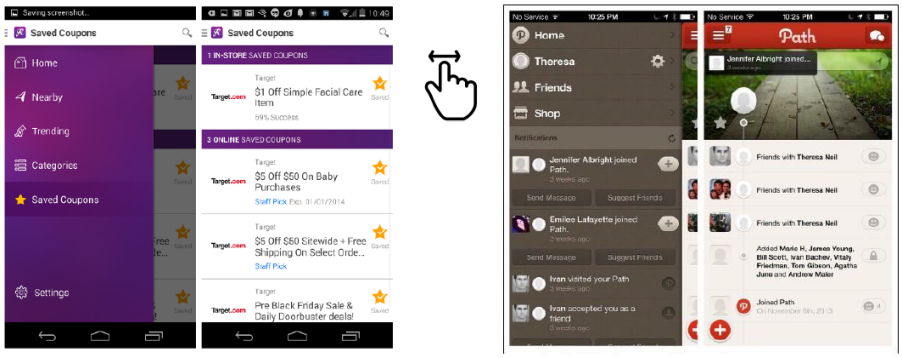


Figure 26: Side Drawer: overlay (left) and inlay (right).

- *Toggle Menu*, which can be an inlay that pushes the content down below the menu, or an overlay that appears as a layer above the content. The overlay design is the more common option in native mobile apps; it should not cover the whole screen but instead let the background peek through. Tapping anywhere in the background should also hide the menu.

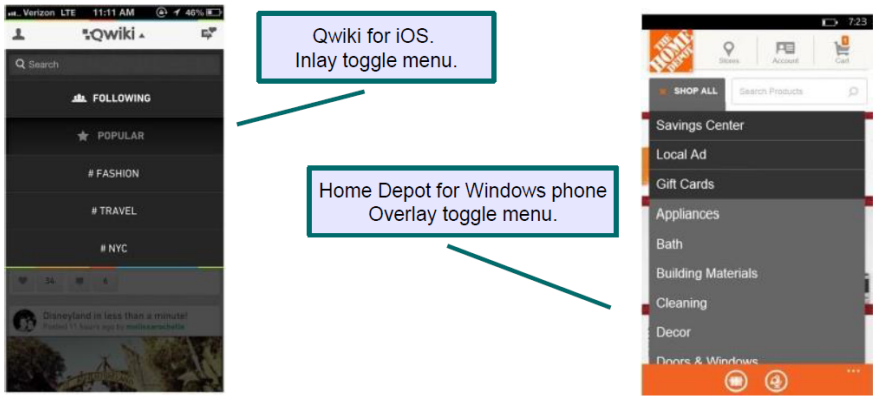


Figure 27: Toggle Menu.

Persisent vs Transient Navigation: when deciding which to take into consideration, we have to answer a few questions:

- Is your app "flat"?
- Are the menu categories equivalent in hierarchy and are there just a few primary categories (i.e. three to five) in the app?
- Do your users need the menu to be always visible for quick access?
- Do the menu categories have status indicators, like the number of unread emails, for instance?

if the answer is "yes" to one or more, it is better to choose persistent navigation.

4.1.2 Secondary Navigation Patterns

It regards the moving and navigating within a selected module.

- *Page Swiping*: this pattern for 2ndary navigation can be used to navigate quickly through content using the swipe gesture. The most common way to communicate this navigation pattern is via page indicators (three little dots) or with cards.

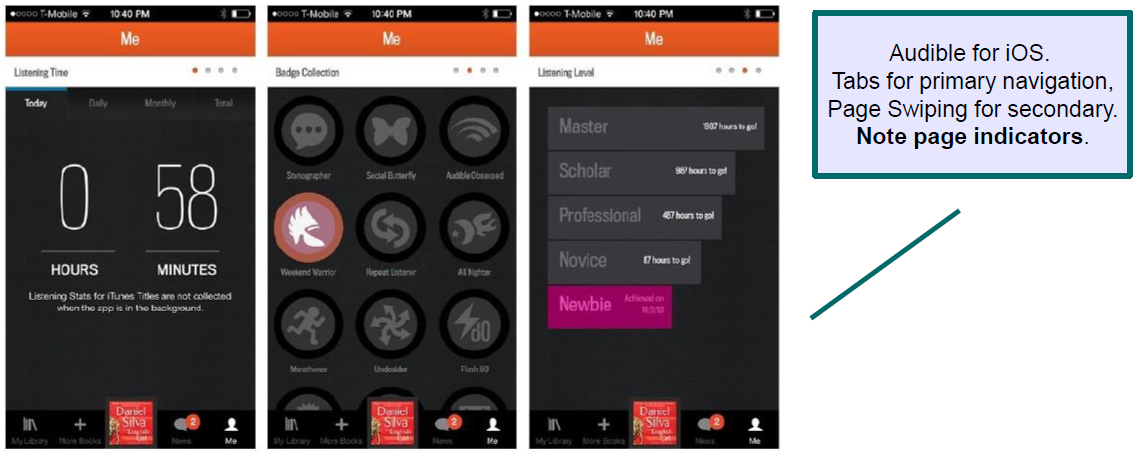


Figure 28: Page Swiping.

4.2 Forms

Forms are used for data entry and config features.

4.2.1 Sign In

Sign in forms require a minimal number of inputs, like username, password, command button, password help and so on. Often, they are visible in the same form of, or alongside with, sign up forms but sometimes they are separated (accessible via buttons). Remember that registration has to be short and done only if necessary.

4.2.2 Check-In

Check-In allows people to use the GPS on their mobile devices to let friends know exactly where they are. We have to keep it ultra-short and design it for speed and efficiency.

Figure 29: Check In.

4.2.3 Comments

App sections used for inviting and allowing users to leave comments. We always have to clarify what is being commented and show other people's ones over time. Timeline should be in minutes ago, hours ago, day of the week etc. depending on the range of viewed past comments.

4.2.4 User's Profile

We put the name and the picture in evidence. We also show users' contribution to the app or to the social network and provide action controls.

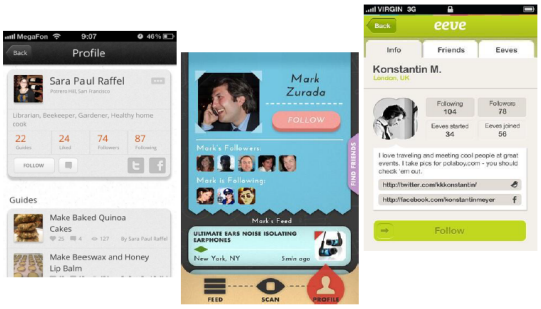


Figure 30: User Profiles.

4.2.5 Share

Always provide an "off social" way to share, e.g. by email. Remark what is being shared and keep track of past logins.

4.2.6 Empty Datasets

Avoid white-screens, explain why the dataset is empty, avoid error messages. Let's say you have a history, display "Empty history" with a fancy drawing!

4.2.7 Multi Steps

Show the user where they are and where they can go. These steps are often organized as sequential workflows. We also have to minimize the number of pages and steps.

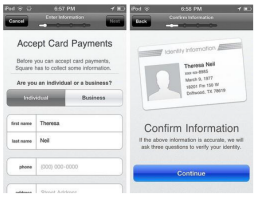


Figure 31: Multi Steps.

4.2.8 Settings

They have to be put inside the app, clear and grouped. Easy to understood!

4.3 Search

We subdivide search in four types.

4.3.1 Explicit Search

We have to offer a clear button in the field, provide an option to cancel the search and use feedback to show that the search is being performed.

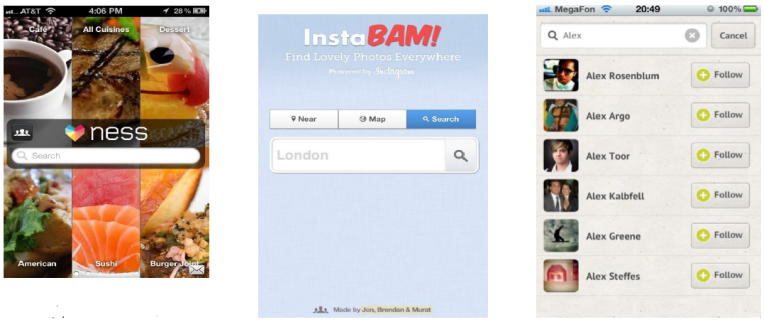


Figure 32: Explicit Search.

4.3.2 Dynamic Search

Automatically filters a given list of items dinamically, during typing. Works well for constrained datasets, like an address book.

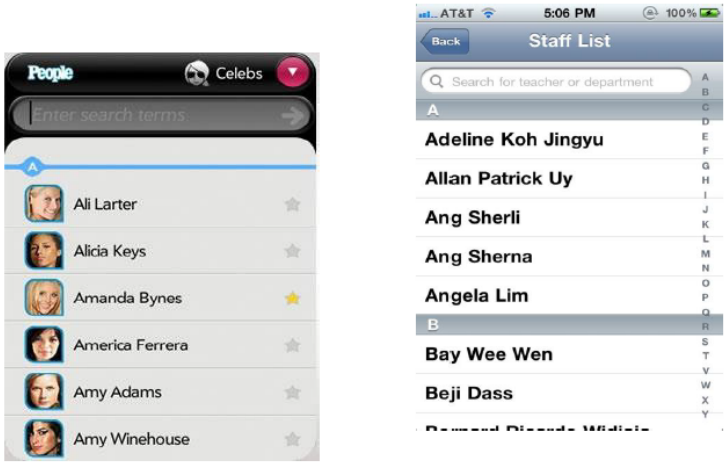


Figure 33: Dynamic Search.

4.3.3 Search Form

We have to minimize the number of input fields, follow form design best practices and use only when strictly needed.

Figure 34: Search Form.

4.3.4 Search Results

We scroll down to analyze the results, apply a reasonable default sort order and call for action.

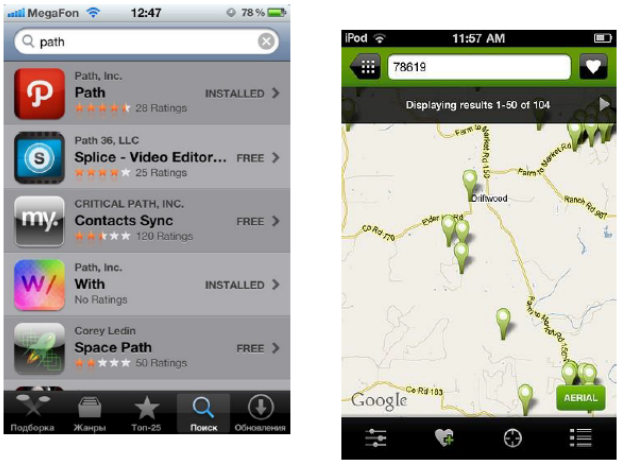


Figure 35: Search Results.

4.4 Tools

4.4.1 Toolbar

A toolbar contains screen level actions, generally displayed at the bottom of the screen. Different from the tab bar! We have to choose icons that are easy to recognize, or use lables plus icons.

4.4.2 Contextual Buttons

If buttons are necessary, they should be displayed in proximity to the actionable object. We have to choose a familiar icon or use a text label for such buttons.

4.4.3 Inline Actions

They should be in proximity to the actionable object. We have to choose, again, a familiar icon or use a text label. Inline actions are the ones present in an Android's RecyclerView, for example, which do something with respect to the selected item. No more than 2 inline actions should be added for item.

4.4.4 Call to Action Buttons

We don't have to hide the main call to action in a menu or disguise it as an unrecognizable icon in a toolbar. We have to choose a good contrast and clear label. Think about the FloatingActionButton of Android.

4.5 MultiState Buttons

They work well for a series of tightly correlated actions that will be performed in succession (like an enable/disable button).

4.6 Actions on Maps

We provide visible markers, using as much screen as possible.

4.7 Helps and Tutorials

They are helpful tips displayed the first time a user launches an app.

4.7.1 Dialog

We keep dialog content short and make sure there is an alternate way to access instructions from within the app. Think about the AlertDialog of Android.

4.7.2 Tips

We place tips in proximity to the feature they refer to, keeping the content short and removing the tip once interaction begins.

4.7.3 Tour

A tour should highlight key features of the application, preferably from a user goal perspective. We keep it short and visually engaging. They are usually presented only on the first run of the app.

4.7.4 Video

Demos and screencasts should showcase key features or show how to use the app.

4.7.5 Transparency

Transparencies are not meant to compensate for poor screen designs. We remove it once interaction begins.

4.8 Feedback and Affordance

4.8.1 Feedback

Feedbacks are composed by *errors* in plain language that propose a solving method, *confirmation* provided when an action is taken and *system status* which provides feedback about the system's status.

4.8.2 Affordance

In affordance we have *Tap*, *Flick* and *Drag*.

5 Designing Mobile Interfaces

A mobile interface is different from the one we find in a desktop application, and for this, we have different challenges when designing a mobile one:

- *Tiny screen sizes* which do not offer much space to present information or choices
- *Variable screen widths* which impose us to use it intelligently when designing the app
- *Dynamic physical environments* in which the mobile is used (bright sun, dark cinema, ...)
- *Touch screens* that need links and buttons large enough to hit easily
- *Difficulty of typing text*, hence we have to minimize this action
- *Limited user attention* of which we should take care during design

In order to design usable mobile interfaces we have to rely on already-defined design patterns, invest on the first time UX which tells us a lot, and use prototyping like mockups.
One of the central problems of a User-Centered Design (UCD) process is how to provide designers with the ability to determine the usability consequences of their design decisions. We can follow **Shneiderman's 8 Golden Rules**:

1. *Strive for consistency* in action sequences, terminology, command use...
2. *Enable frequent users to use shortcuts* to perform familiar actions more quickly
3. *Offer informative feedback* for every user action at an appropriate level
4. *Design dialogs to yield closure* i.e. let the user know when they have completed a task
5. *Offer error prevention and simple error handling* i.e. clear instructions to recovery
6. *Permit easy reversal of actions* to relieve anxiety and encourage exploration
7. *Support internal locus of control* (the user is in control of the system)
8. *Reduce short-term memory load* keeping display simple, providing time for learning action sequences...

or **Norman's 7 principles**

1. Use both knowledge in the world and knowledge in the head
2. Simplify the structure of tasks
3. Make things visible
4. Get the mapping right, i.e. make sure that the user can determine the relationships (like larger size of money -¿ larger value)
5. Exploit the power of constraints to make the user perform the right action only
6. Design for error

7. When all else fails, standardize

One way to approach UI design is to learn from examples that have proven to be successful in the past. Design Patterns are solutions to a recurrent problem within a specific app domain. These are the ingredients for usable mobile design:

- **Information Architecture**: the organization and structure of data within an informational space. In other words, how the users will get to information or perform tasks within an app
- **Interface Design**: The design of the visual paradigms from which the user will assess meaning and direction given the information presented to her/him
- **Interaction Design**: The design of how the user can participate with the information present either in a direct or indirect way, meaning how the user will interact with the application to create a more meaningful experience and accomplish her/his goals
- **Information Design**: The visual layout of information presented to the users
- **Mobile Design Patterns**

5.1 Information Architecture

Information Architecture represents the core of the user experience. From a simple mobile website to an iPhone/Android app, the i.a. defines how the information wil be structured. The truly successful mobile products always have a well though and defined info architecture. The first deliverable to define info architecture is the *site map* that represents the relationship of content to other content and provide a map for how the user will travel through the informational space.

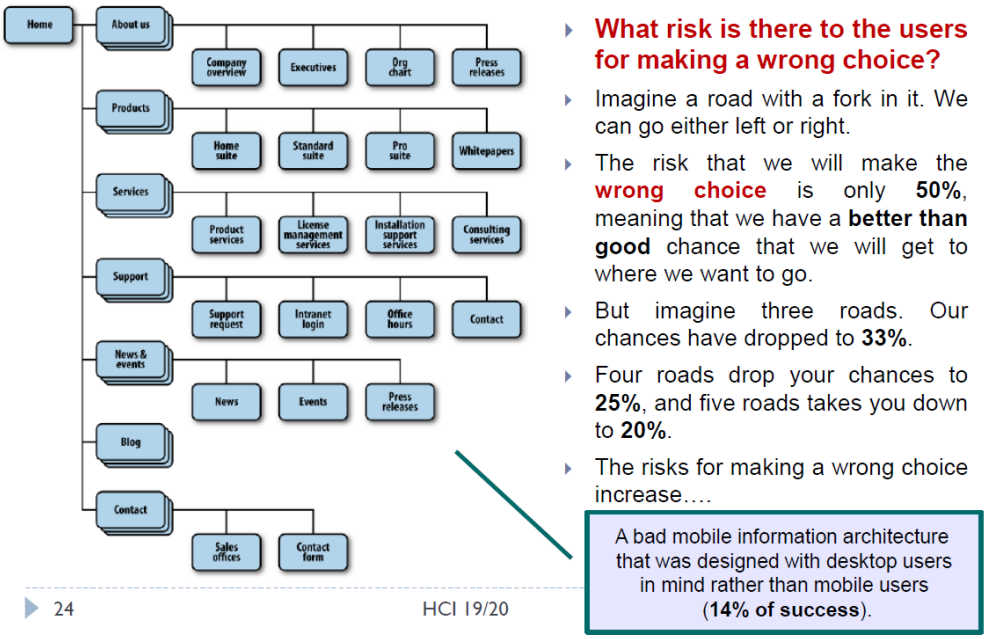


Figure 36: Bad Site Map for Mobile.

- In the mobile context, **tasks are short** and **users have limited time** to perform them.
- **Limit users' options**: A mobile information architecture should provide **5 navigation areas or less**.
  - The risks to make the wrong choice are minor.
- **Suggestion**: Make the path through the information you present **logical** and **easy to predict**.
  - put **markers** to let them where they are.
  - put a **back-button**.
  - *When mobile users select the wrong path, they should immediately click back to where they started and go down another path, eliminating the wrong choices to find the right ones.*

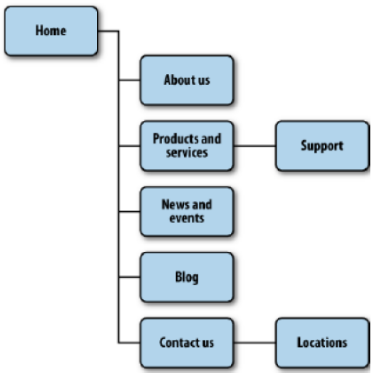


Figure 37: Good Site Map for Mobile.

5.2 Interface Design

Interface design analyzes the visual layout of content presented to a mobile user and how the user assesses meaning and direction from it. The greatest challenge to creating a mobile design that works well on multiple screen sizes is filling the width. a traditional solution is the use of *vertical designs*: the interface design is a cascade of content from top to bottom, similar to a newspaper where the content consumes the majority of the screen.

Figure 38: Vertical Design.

For content-heavy sites and apps, this solution works well. The problem is when it is required to present a large number of tasks or actions. We see some alternatives to accomplish this goal.

- *Axis* is the most basic and common information principle for organizing content and consists of an imaginary line that is used to align a group of elements in an interface. The design feels ordered with them. Axis can be made more apparend if the edges of surrounding elements are well defined (axis reinforcement).

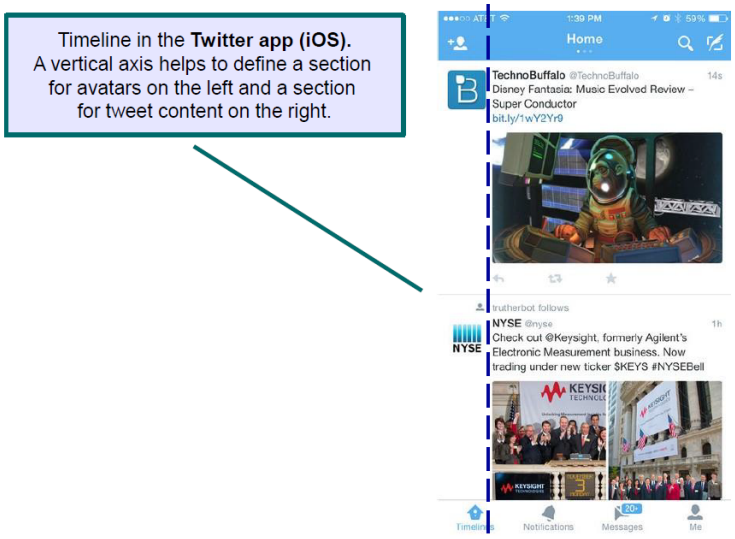


Figure 39: Example of Axis Reinforcement.

When we encounter something linear, such as an axis, we naturally follow the line in a direction; lines in fact encourage *movement* and interactions.

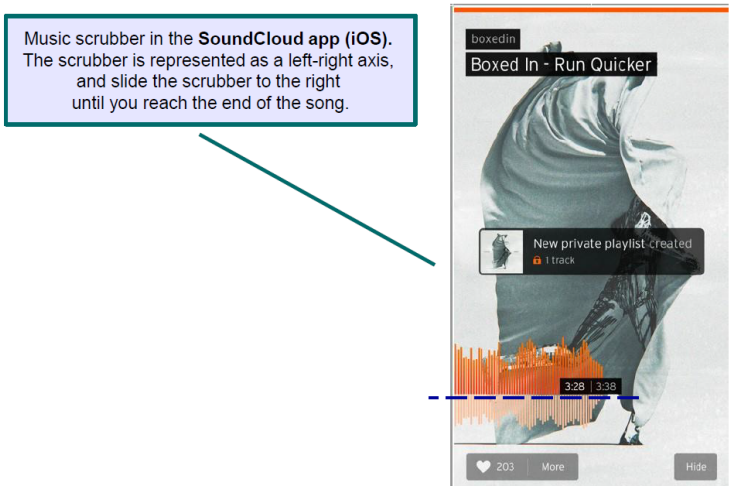


Figure 40: Example of Axis Movement.

We call *Infinite Axis* when an axis endpoint is undefined, i.e. we can slide through the axis indefinitely until we reach something interesting or we get tired of such interaction (social networks).

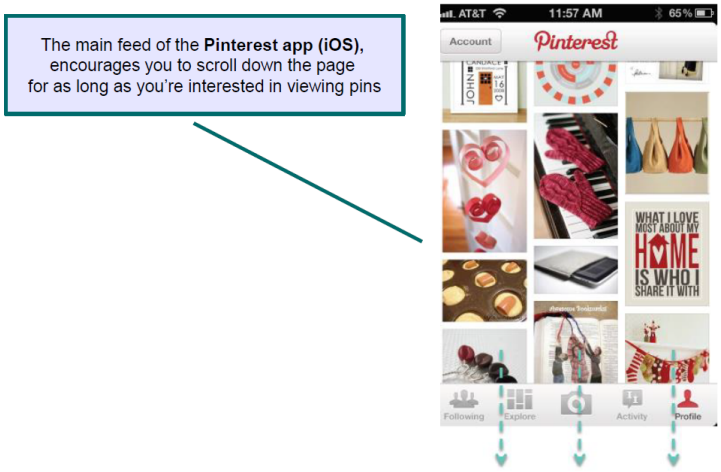


Figure 41: Example of Infinite Axis.

- *Simmetry* helps to make the design feel armonious and easy to read, both top-bottom and left-right. Think about Android's image gallery.

- *Hierarchy* is obtained when an element appears more important in comparison to others in a design. We have different hierarchies:
  - by Size (trivial)
  - by Shape (rounded Instagram profile picture wrt the rest)
  - by Placement (end of an axis is more hierarchical than points through the axis)
- *Rhythm* is the movement created by a repeated pattern of forms, like the one that we can see in Instagram's feed with images and videos. The same pattern repeats over and over, hence the user knows exactly where to look in the rhythm to get a specific info.
- *Break* in a repeated pattern gets more hierarchical and usually divides in sections a rhythm (think about "who to follow" and "who you searched" when we see the list of people in instagram search!).

5.3 Interaction Design

Interaction Design investigates the way people interact with their mobile devices. The interaction is any direct or indirect communication between a user and his/her mobile device. Direct is with user feedback, indirect is with sensors or similar stuff. When we perform direct one, we can do it through single or multi touch, and also through physical buttons like a pad or so.

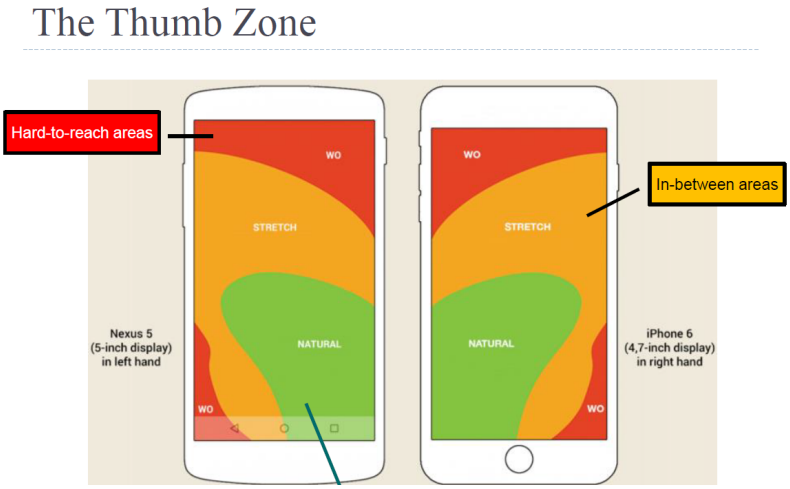


Figure 42: Thumb Zone.

We have to place relevant content within the thumb's reach. Important content needs to be aimed towards the thumb, while secondary content (everything that is not oftenly accessed) can be placed in the stretch. Better not to use hard to reach areas at all.

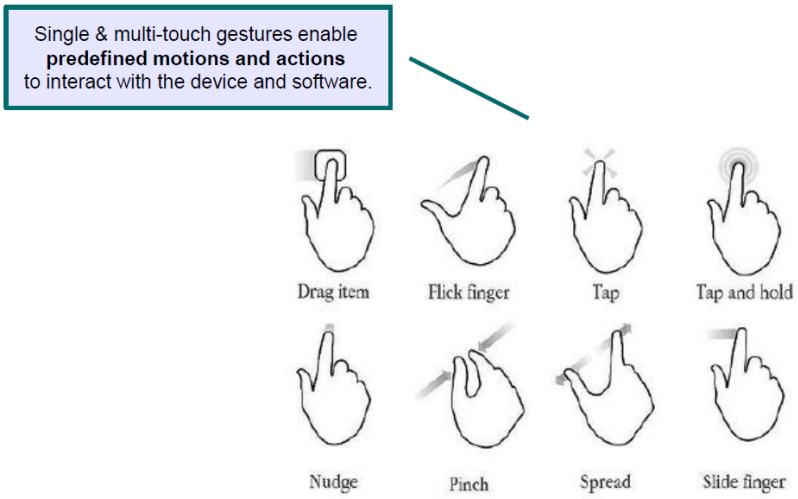


Figure 43: Touch interaction and gestures.

Gestures have to be embraced with attention. Users expect gestures to work the same, regardless of the app they are currently running. We have to make sure that users can reach them!

5.4 Information Design

It concerns the visual layout of info presented to the users. Three main aspects to consider:

- Design for "fat fingers ". Make your links and buttons large enough to hit easily
- Design for distracted users.
- Think about colors and typography. And think about motion.

Users react differently to colors, since they evoke different emotions. A predefined color palette has to be chosen when designing an app.

Figure 44: Colors emotions.

Remember also to follow readability guidelines, i.e. readable font and space between letters and lines, organizing the text in short paragraphs.

6 User-Centered Design (UCD)

The primary aim of the process of design and implementation of an interactive system should be to maximize the usability of the system. It is therefore important for us to understand:

- The characteristics, methods and tools of a process of design and implementation that can maximize usability
- The characteristics of usability
- How to measure and/or evaluate the usability of an interactive system (another class on evaluation)

UCD is also referred to as the user-centered methodology or human-centered design or human-centered methodology, and is an approach to design that grounds the process in information about the people who will use the product. UCD intends to ensure that the user is at the center during the design process in order to realize products that meet usability requirements.
Usability is the property that reflects the ease of use of a computer system. It is defined by 5 principles: learnability, efficiency, memorability, errors and satisfaction. It is important in improving the software quality. The use context also improves the same way. Let's see some advantages of usability:

- Allows you to focus on user needs and organization
- Increases productivity
- Improve the quality of products
- Improve quality of life
- Allows compliance e.g. with ISO standards, European directives on displays

We observed that a common definition of usability is: "the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use."
Here are some characteristics of UCD:

1. Know your users
2. Actively involve users early and continuously
3. Rapid and frequent iteration of designs with usability assessments
4. Multidisciplinary team

According to the ISO 13407 standard, there are four essential user-centered design activities which should be undertaken to incorporate usability requirements into the software development process:

1. understand and specify the context of use: the quality of use of a system depends very much upon the context in which a system will be used. In some cases contextual information may already be known, although, where a new product or system is to be introduced, then it will be necessary to collect the relevant contextual information. At the end, the following aspects are understood:

- the characteristics of the intended users
- the tasks the users will perform , and allocation of activities between users and system
- constraints and characteristics of the socio-organizational and technological environment in which the users will use the system

the result of this initial activity are embodied in a document which desscribes the context of use for the proposed software.

2. specify the user and organizational requirements: building on the context of use description obtained previously, an explicit statement of the user-centered requirements for the new software should be formulated, with identification of relevant users' range, prevision of a clear statement of design goals...
3. produce design and prototypes: the key goal is to simulate the design solutions using paper or computer based mockups, with which we can explore design solutions and prototypes in general and then later presenting them to a representative sample of users. We have to involve users early in order to explore and refine design choices in light of feedback, and iterate design solutions until the design/usability goals or requirements are met.
4. carry out assessments/evaluations: this is a crucial phase in which we develop an evaluation plan. Three steps:
   - identify anomalies and defects most relevant at this stage
   - select the best solution for the system in light of the requirements at this stage
   - report the results and recommendation for refining the design at this stage; the refine design becomes the design in the next stage

   This evaluation process is iterated at each stage until design/usability goals or requirements are met.

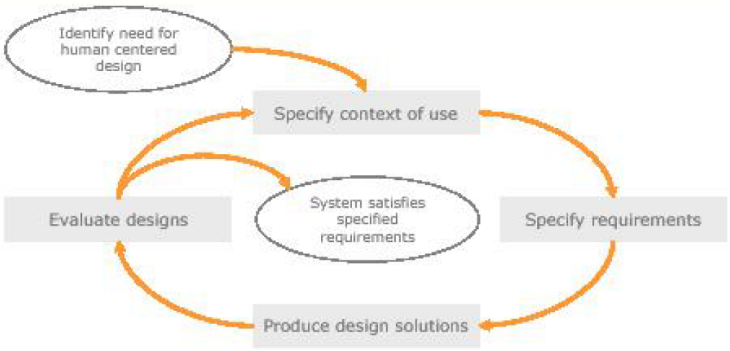


Figure 45: UCD Activities.

There are many methods which can be used to achieve the goals of UCD: requirements analysis, task analysis and evaluation techniques.

7 Interaction Design Basics

Interaction design is not just about the design of interactive systems. Interaction design also includes the design of artifacts (e.g.tools/resources such as office equipment). Interaction design is about the design of interaction itself. However, i.d. is not just abut the artifact that is produced, it's about understanding and choosing how the artifact is going to affect the way people work or live. It is also about documentation, manuals, tutorials etc.

Design is achieving goals within constraints. Goals are the purpose of the design we want to do, while constraints are materials, standards, platforms, cost, time... we have to find the tradeoff. The golden rule of design is: *understand your materials*, that in HCI means understanding computers, humans and their interaction.

The central message or core of interaction design is the user. We have to know their background, computer experience and so on. For doing it, we use Personas, Scenarios, Questionnaires, Interviews.

8 Cognitive Models

Cognitive models are models represented by goal and task hierarchies. They model aspects of user: understanding, knowledge, intentions and processing. Common categorizations are: competence vs performance, computational flavor, no clear divide (we choose one).
Goals - Intentions, Tasks - Actions
For goal hierarchies we can follow different schemes:

- *GOMS*: Goals, Operators, Methods and Selection. Operators are basic actions performed by user, methods are decompositio of a goal into subgoals/operators and selection means of choosing between competing methods.

```
GOAL: CLOSE-WINDOW
.   [select GOAL: USE-MENU-METHOD
            .   MOVE-MOUSE-TO-FILE-MENU
            .   PULL-DOWN-FILE-MENU
            .   CLICK-OVER-CLOSE-OPTION
            GOAL: USE-CTRL-W-METHOD
            .   PRESS-CONTROL-W-KEYS]

For a particular user:

   Rule 1: Select USE-MENU-METHOD unless another
           rule applies
   Rule 2: If the application is GAME,
           select CTRL-W-METHOD
```

Figure 46: GOMS Example.

- *Cognitive Complexity Theory*: CCT comprises two parallel descriptions: user production rules and device generalised transition networks. Production rules are of the form: *if condition then action*. Transition networks are covered under dialogue models.

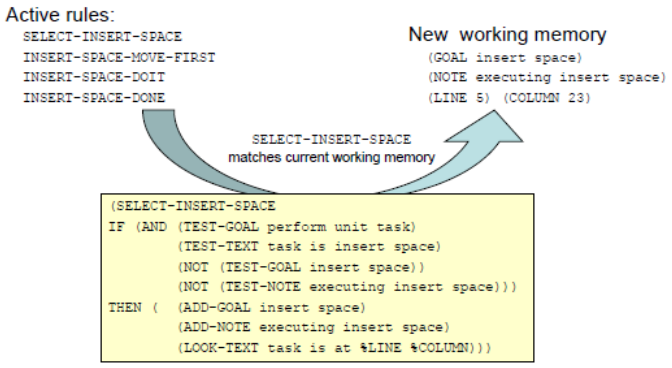


Figure 47: CCT Example.

- *HTA*: they will be explained in detail in next chapters.

Linguistic notations are needed to understand the user's behavior and cognitive difficulty based on analysis of language between user and system. There are two main types:

- *Backus-Naur Form (BNF)*: it is a very common notation from computer science and is a purely syntactic view of the dialogue. There are two components: *terminals* (lowest level of user behavior like mouse actions) and *nonterminals* (ordering of terminals and higher level of abstraction like menu selection). The basic syntax of BNF is nonterminal ::= expression. An expression contains terminals and nonterminals combined in sequence (+) or as alternatives (|).

```
draw line     ::=  select line + choose points + last point
select line   ::=  pos mouse + CLICK MOUSE
choose points ::=  choose one  |  choose one + choose points
choose one    ::=  pos mouse + CLICK MOUSE
last point    ::=  pos mouse + DBL CLICK MOUSE
pos mouse     ::=  NULL  |  MOVE MOUSE+ pos mouse
```

Figure 48: BNF Example.

Measurement in BNF can be done through number of + and | operators. There are some complications, like: same syntax for different semantics, no reflection of user's perception and minimal consistency checking.

- *Task Action Grammar (TAG)*: this method makes consistency more explicit. We have parameterised grammar rules. Nonterminals are modified to include additional semantic features.

In BNF, three UNIX commands would be described as:

```
copy  ::= cp + filename + filename | cp + filenames + directory
move  ::= mv + filename + filename | mv + filenames + directory
link  ::= ln + filename + filename | ln + filenames + directory
```

No BNF measure could distinguish between this and a less consistent grammar in which

```
link ::= ln + filename + filename  |  ln + directory + filenames
```

Figure 49: Another BNF.

consistency of argument order made explicit using a parameter, or semantic feature for file operations

Feature Possible values

Op = copy; move; link

Rules

file-op[Op] ::= command[Op] + filename + filename
| command[Op] + filenames + directory
command[Op = copy] ::= cp
command[Op = move] ::= mv
command[Op = link] ::= ln

Figure 50: TAG.

Among other uses of TAG, we find user's existing knowledge and congruence between features and commands, modelled as derived rules.

Physical and device models are based on empirical knowledge of human motor system. User's task: acquisition then execution. It is complementary with goal hierarchies. Most important type:

- *Keystroke Level Model (KLM)*: it is the lowest level of original GOMS. It has six execution phase operators, and times are empirically determined.

– Physical motor:	K - keystroking P - pointing H - homing D - drawing
– Mental	M - mental preparation
– System	R - response

Texecute = TK + TP + TH + TD + TM + TR

Figure 51: KLM Operators and Time.

```
GOAL: ICONISE-WINDOW
[select
   GOAL: USE-CLOSE-METHOD
   .     MOVE-MOUSE-TO- FILE-MENU
   .     PULL-DOWN-FILE-MENU
   .     CLICK-OVER-CLOSE-OPTION
   GOAL: USE-CTRL-W-METHOD
         PRESS-CONTROL-W-KEY]
```

- compare alternatives:
  - USE-CTRL-W-METHOD vs.
  - USE-CLOSE-METHOD
- assume hand starts on mouse

USE-CTRL-W-METHOD		USE-CLOSE-METHOD	
H[to kbd]	0.40	P[to menu]	1.1
M	1.35	B[LEFT down]	0.1
K[ctrlW key]	0.28	M	1.35
		P[to option]	1.1
		B[LEFT up]	0.1
Total	**2.03 s**	**Total**	**3.75 s**

Figure 52: KLM Example.

Architectural Models: all of these cognitive models make assumptions about the architecture of the human mind, like with LTM/STM, problem spaces, interacting cognitive subsystems...

Display-based interaction: most cognitive models do not deal with user observation and perception. Some techniques have been extended to handle system output (e.g. BNF with sensing terminals, display-TAG,...), but problems persist.

9 Requirements Gathering

A *requirement* is something the product must do or a quality it must have. However, users do not mention some requirements because they assume that they are obvious. Good interviews and observations will help to reveal them. Some of them arise only when models are constructed or prototypes are reviewed. We can divide them into:

- *functional requirements*, what the system must do;
- *non-functional requirements*, qualities the system must have.

Ethnography is an ethnographic technique where the evaluator visits the normal workplace of the users. The evaluator should be as unobtrusive as possible so as to allow the user to work normally. Usually, there are no videos in order to respect this not-intrusion criterion. Attention is paid to how tasks are actually done, as opposed to the way they are thought to be done. The purpose of ethnography is to give designers and evaluators an understanding of the context in which a technology is used.

Interviews should be thematic yet open-ended and discursive to allow the participant to direct the process somewhat. Extracts of interviews can be made to highlight details of particular interest.

- *Structured interviews* have a specific, predetermined agenda;
- *Unstructured interviews* are used during the earlier stages of design and the goal is to gather as much infos as possible concerning the user's experience. It is done with no structure, no agenda.

Focus groups are a technique for collecting data from a range of users. A moderator is required to lead the group, but the session should be as fluid as possible whilst staying on topic. All participants should contribute and care should be taken to cover a broad range of topics and not allow one person to dominate. The collected data may be difficult to organize, but audio recording should help.

Questionnaires need very careful design and piloting and are composed by closed questions and opened ones. Obviously, the first ones are easier to analyze. Sensitive questions have to be asked in a series of separated closed questions so that the sensitive answer is not directly disclosed. Questionnaires require testing before release and statistical verification and analysis, and therefore can be unwieldy.

PACT Analysis is People, Activities, Context, Technologies analysis of a system.

Storyboards are series of scenes/frames from the user experience point of view, usually based on scenarios. They can be used in requirement collection and with mock-ups and they can be time consuming. Storyboards may not accurately reflect actual process to be implemented and should be refined during the design process.

10 Task Models

Task models are methods to analyze people's jobs, i.e. what they do, what things they work with and what they must know. There are several approaches to task analysis that we will study now in detail:

10.1 Task decomposition

In task decomposition, as it is easy to imagine, we split each task into ordered subtasks. The aims of this technique are to describe the actions people do, to structure them within task subtask hierarchy and describe order of subtasks. *Hierarchical Task Analysis (HTA)* is widely used as a technique for task decomposition.

Hierarchy description ...

0. in order to clean the house
    1. get the vacuum cleaner out
    2. get the appropriate attachment
    3. clean the rooms
        3.1. clean the hall
        3.2. clean the living rooms
        3.3. clean the bedrooms
    4. empty the dust bag
    5. put vacuum cleaner and attachments away

... and plans

Plan 0: do 1 - 2 - 3 - 5 in that order. when the dust bag gets full do 4

Plan 3: do any of 3.1, 3.2 or 3.3 in any order depending on which rooms need cleaning

N.B. only the plans denote order

Figure 53: HTA Description Example.

In order to generate the hierarchy, we do three steps:

1. get list of tasks
2. group tasks into higher level tasks
3. decompose lowest level tasks further

We expand only relevant tasks, hence we stop when subtasks are not needed anymore in a branch.

HTA diagrams are composed by explanations on nodes and plans about them:

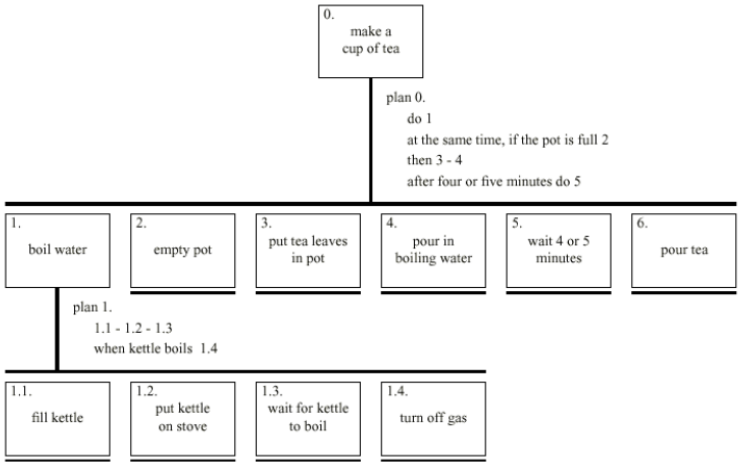


Figure 54: HTA Diagram Example.

We can also refine this diagram through heuristics, like restructure, balance and generalise:

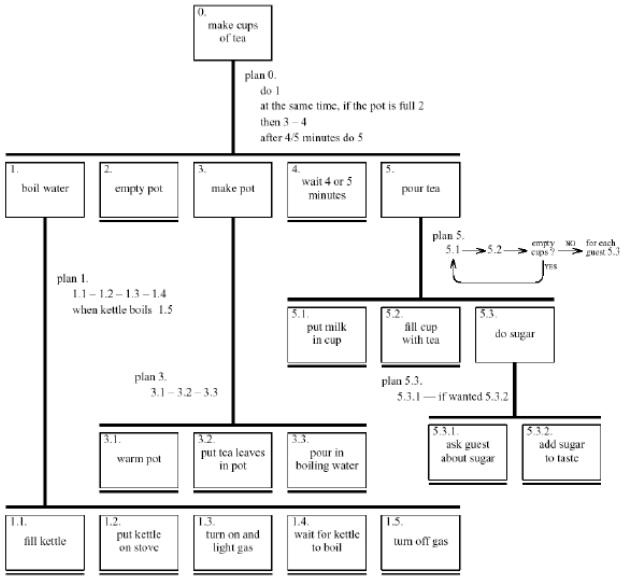


Figure 55: HTA Refined Example.

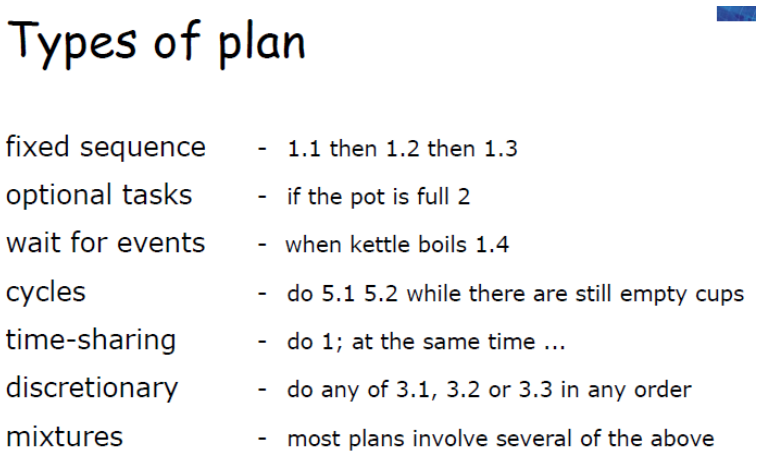


Figure 56: Plans.

Waiting can be misleading because we do not know where to put it, whether in tasks or plans.

10.2 Knowledge based analysis

In knowledge based analysis, the focus is on *objects* used in tasks and *actions* performed. *Taxonomies* represent levels of abstraction.

```
motor controls
   steering  steering wheel, indicators
   engine/speed
        direct   ignition, accelerator, foot brake
        gearing  clutch, gear stick
   lights
        external  headlights, hazard lights
        internal  courtesy light
   wash/wipe
        wipers   front wipers, rear wipers
        washers  front washers, rear washers
   heating  temperature control, air direction,
            fan, rear screen heater
   parking  hand brake, door lock
   radio  numerous!
```

Figure 57: Knowledge Based Example.

There are three types of branch point in taxonomy:

- XOR, normal taxonomy and object in one and only one branch
- AND, object must be in both, multiple classifications
- OR, weakest case can be in one, many or none

```
kitchen item AND
/____shape XOR
/    |____dished  mixing bowl, casserole, saucepan,
/    |            soup bowl, glass
/    |____flat    plate, chopping board, frying pan
/____function OR
     {____preparation    mixing bowl, plate, chopping board
     {____cooking    frying pan, casserole, saucepan
     {____dining XOR
          |____for food   plate, soup bowl, casserole
          |____for drink  glass
```

N.B. '/ | {' used for branch types.

Figure 58: Task Description Hierarchy Example.

10.3 Entity-Relationship Techniques

Here we have focus on objects, actions and their relationships.

Running example

'Vera's Veggies' – a market gardening firm
owner/manager: Vera Bradshaw
employees: Sam Gummage and Tony Peagreen
various tools including a tractor `Fergie`
two fields and a glasshouse
new computer controlled irrigation system

Figure 59: Entity-Relationship Technique Example.

We start with a list of objects and classify them:

- *Concrete objects*: simple things
- *Actors*: human actors
- *Composite objects*: sets and tuples

Attributes can be added to the objects. *Actions* instead are listed and associated with each agent (who performs the action), patient (which is changed by the action), instrument (used to perform action).

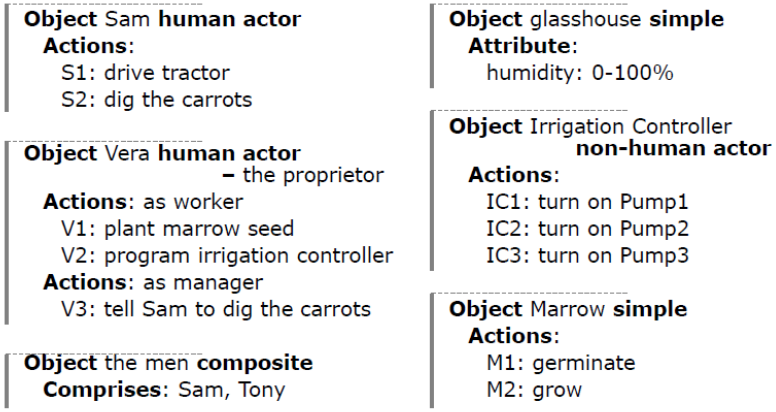


Figure 60: Object and Actions Example.

Events are performance of actions and can be spontaneous or timed. *Relationships* can be between objects, action-object, actions-events or temporal ones.

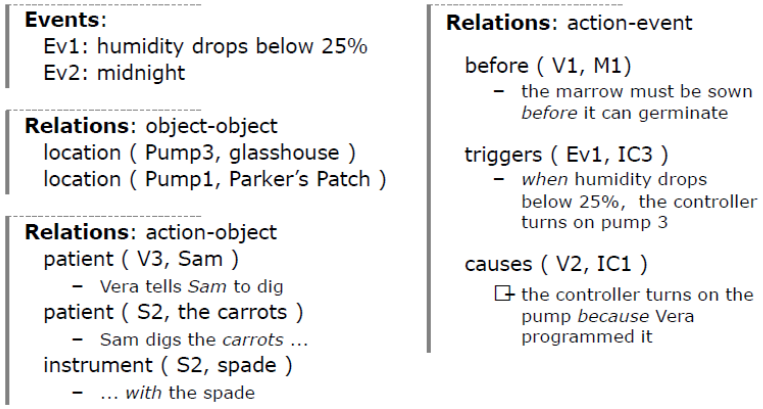


Figure 61: Events and Relationships Example.

Sources of Information can be documentation, observations and interviews.

11 Dialogue Notations and Design

A dialogue is a conversation between two or more parties, usually cooperative. In UIs, it refers to the structure of the interaction, syntactic level of human-computer conversation. There are several levels, such as *lexical* (shape of icons, actual keys pressed), *syntactic* (order of inputs and outputs) and *semantic* (effect of internal application/data). Note that human-human dialogue is usually very unstructured, while in computers we always have a fixed and constrained structure. Dialogue gets buried in the program; in a big system we can analyze the dialogue because dialogue notations help us to analyze systems and separate lexical from semantic. Before the system is built, notations help us understand proposed designs.

11.1 Graphical Notations

11.1.1 STNs

STNs, i.e. State Transition Networks, are diagrams characterized by states (circles) and arcs (actions/events) and describe the behavior of a system in a particular path/function.

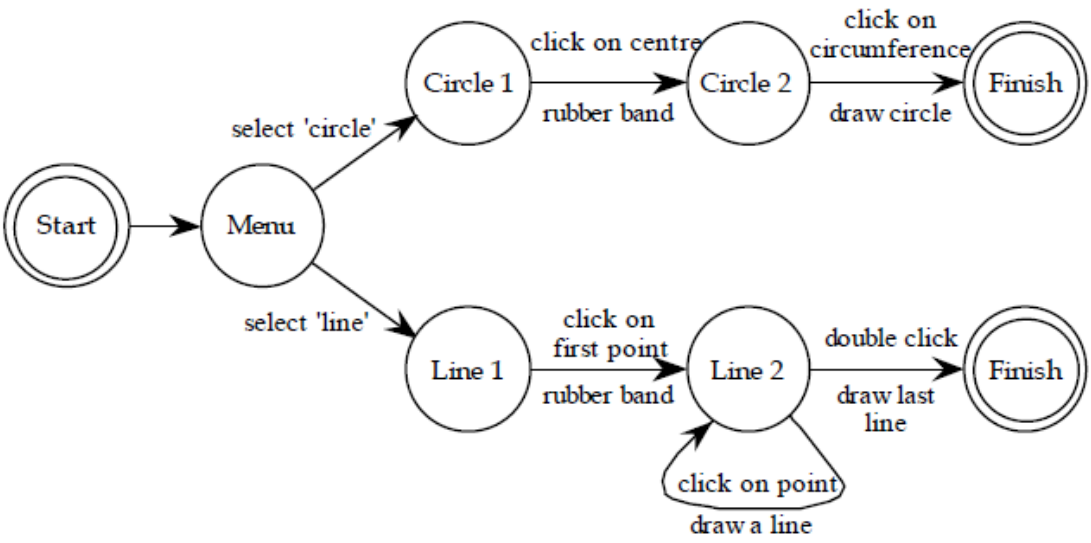


Figure 62: STN Example.

As we can see from the figure, arc labels are a bit cramped because notation is "state heavy" and the events require most detail. Labels in circles are a bit uninformative instead: in fact, states are hard to name but easier to visualize. STNs can also be organized in a hierarchy, such that they can manage complex dialogues. Concurrent dialogues instead, are like what happens in a concurrent dialog checkbox:

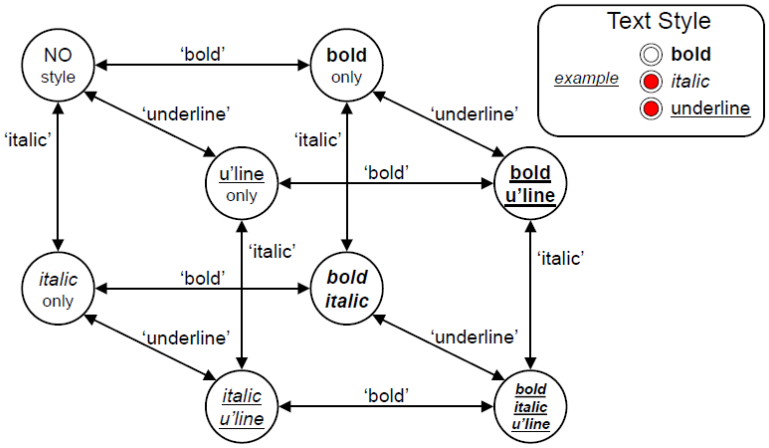


Figure 63: Concurrent Dialogues Example.

Escapes are actions which permit the user to exit from an STN-related app function, like back button or similar. We have to avoid this, generally, and provide an easy-to-use escape to the user instead of a trivial back button.

11.1.2 Petri nets

Petri nets are one of the oldest notations in computing. They are flow graphs, with *places*, *transitions* and *counters*. First two ones are a bit like in STNs, while a counter is the current state. Several counters are allowed, i.e. concurrent dialogue states.

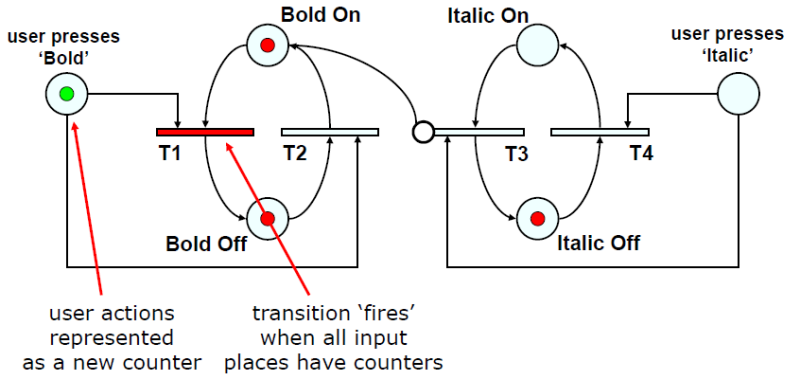


Figure 64: Petri Net Example.

11.1.3 State charts

State charts are used in UML and are an extension to STNs, with hierarchy, concurrent subnets, escapes and history.

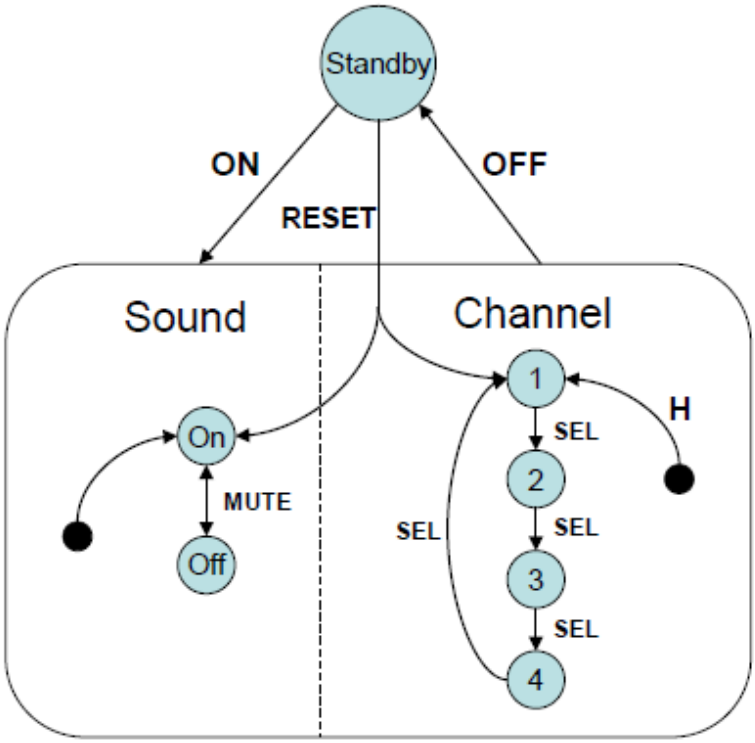


Figure 65: State Chart Example.

11.1.4 Flowcharts

Flowcharts are familiar to programmers and are composed by process/event, not states. They are used for dialogues. There is a COBOL transaction processing and, as we can see frm the following figure, dialogue flow charts are used.

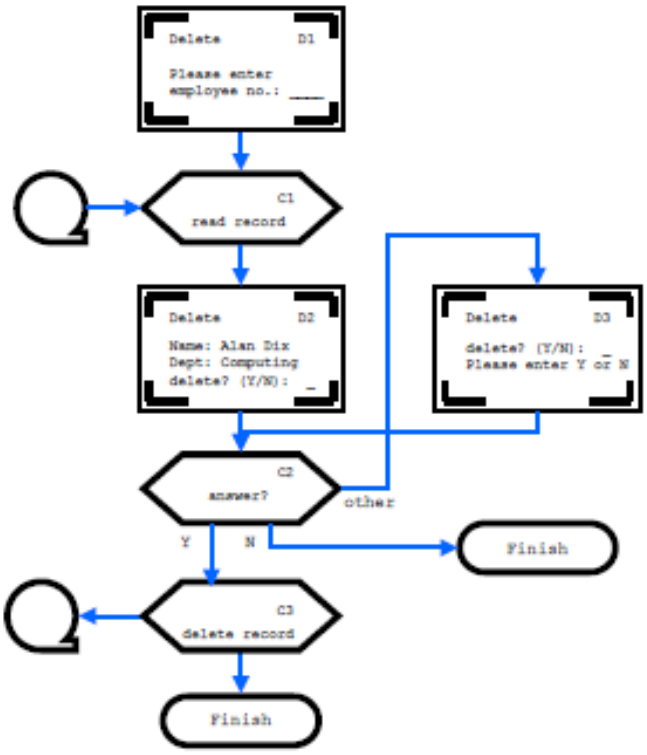


Figure 66: Flow Chart Example.

11.1.5 JSD Diagrams

They are used for tree structured dialogues. They are less expressive but have greater clarity.

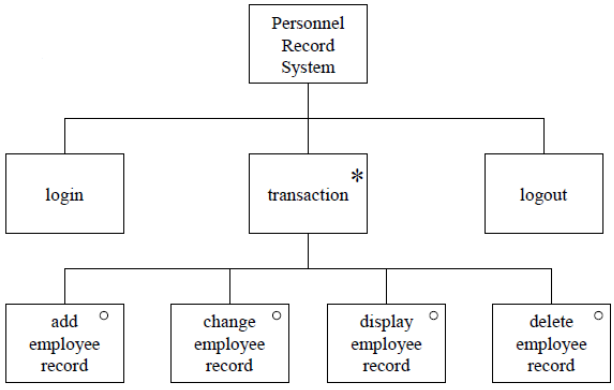


Figure 67: JSD Example.

11.2 Textual Notations

11.2.1 Grammars

- Regular expressions
  ```
  sel-line click click* dble-click
  ```
- compare with JSD
  - same computational model
  - different notation
- BNF
  ```
  expr ::= empty
         | atom expr
         | '(' expr ')' expr
  ```
- more powerful than regular exp. or STNs
- Still NO concurrent dialogue

11.2.2 Production rules

It is an unordered list of rules of the type "if condition then action". They are good for concurrency but bad for sequence!
Event-based production rules are rules added to list of pending events, as they represent events themselves.

```
Sel-line → first
C-point first → rest
C-point rest → rest
D-point rest → < draw line >
```

Figure 68: Event-based prod. rule example.

Prepositional Production System is state based and has attributes and rules.

Attributes:

Mouse: { mouse-off, select-line, click-point, double-click }

Line-state: { menu, first, rest }

Rules (feedback not shown):

select-line → mouse-off first

click-point first → mouse-off rest

click-point rest → mouse-off

double-click rest → mouse-off menu

Figure 69: Attributes and rules example.

11.2.3 CSP and Process Algebras

They are used in Alexander's SPI that we will now analyze; they are good for sequential dialogues and concurrent dialogue, but causality is unclear in them.
In *Alexander's SPI*, we have two-parts specification (*EventCSP* pure dialogue order and *EventISL* target dependent semantics). Dialogue description is centralised and syntactic/semantic tradeoff tollerable.

Semantics Alexander SPI (ii)

- EventCSP

```
Login  = login-mess -> get-name -> Passwd
Passwd = passwd-mess -> (invalid -> Login [] valid -> Session)
```

- EventISL

```
event: login-mess
    prompt: true
    out: "Login:"
event: get-name
    uses: input
    set: user-id = input
event: valid
    uses: input, user-id, passwd-db
    wgen:  passwd-id = passwd-db(user-id)
```

Here we describe *Action Properties*:

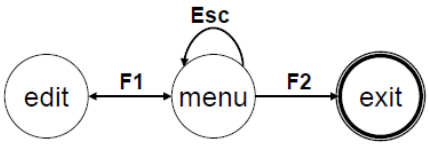
- *completeness*
- *determinism*
- *nested escapes*
- *consistency*

these properties can be checked in an STN: in fact, a good UI designer should take care of them a lot.
State Properties are:

- *reachability*
- *reversibility*
- *dangerous states*

Dangerous States

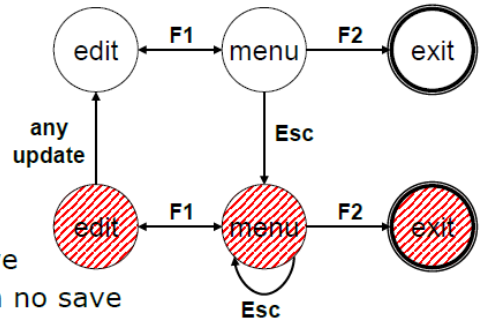
- word processor: two modes and exit
  - F1 - changes mode
  - F2 - exit (and save)
  - Esc - no mode change



but ... Esc resets autosave

Dangerous States (ii)

- exit with/without save ⇒ dangerous states
- duplicate states - semantic distinction



F1-F2 - exit with save
F1-Esc-F2 - exit with no save

Note that such dangerous states can be reached for fault of layout, so we have to design a layout that avoids errors. Think, for example, that we have two nearby buttons for opposite actions. If the user wrongly clicks the other one, it will trigger the opposite action and it could be a disaster in very important, not reversible actions!

12 Design Rules

12.1 Introduction

Design rules (or usability rules) are rules that a designer can follow in order to increase the usability of the system/product e.g., principles, standards, guidelines.
For the principles:

- Abstract and have high generality and low in authority
- Widely applicable and enduring

For the guidelines:

- Narrowly focused
- Can be too specific, incomplete and hard to apply but they are more general and lower in authority than Standards (like ISO and so on)

Design rules should be used early in the lifecycle. We will first look at abstract principles for supporting usability and later on, we will look at the most well used and well known sets of heuristics of golden rules, which tend to provide a succinct summary of the essential characteristics of good design.

12.2 Usability Principles

Usability principles:

- Learnability: the ease with which new users can begin effective interaction and achieve maximal performance.
  - Predictability: support for the user to determine the effect of future action based on past interaction history.
  - Synthesizability: support for the user to assess the effect of past operations on the currentstate.
  - Familiarity: the extent to which a user's knowledge and experience in other real-world or computer-based domains can be applied when interacting with a new system
  - Generalizability: support for the user to extend knowledge of specific interaction within and across applications to other similar situations.
  - Consistency: likeness in input-output behavior arising from similar situations or similar task objectives.
- Flexibility: the multiplicity of ways the user and system exchange information
  - Dialogue initiative: user freedom for artificial constraints on the inpur dialog imposed by the system.
  - Multithreading: the ability of the system to support user interaction for more than one task at a time.

  - Task migratability: the ability to transfer control for execution of tasks between the system and the user.
  - Substitutivity: the extent to which an application allows equivalent input and output values to be substituted for each other.
  - Customizability: the ability of the user or the system to modify the user interface.

- Robustness: the level of support provided to the user in determining successful achievement and assessment of goal-directed behavior.
  - Observability: the extent to which the user can evaluate the internal state of the system from the representation on the user interface.
  - Recoverability: the extent to which the user can reach the intended goal after recognizing an error in the previous interaction.
  - Responsiveness: a measure of the rate of communication between the user and the system.
  - Task conformance: the extent to which the system services support all the tasks the user would wish to perform and in the way the user would wish to perform.

12.3 Heuristics and Golden Rules

Here we list Jakob Nielsen's 10 Usability Heuristics:

1. Visibility of system status: the system should always keep users informed about what is going on, through appropriate feedback within reasonable time.
2. 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.
3. 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.
4. Consistency and standards: users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform conventions.
5. Error prevention: even better than good error messages is a careful design which prevents a problem from occurring in the first place.
6. Recognition rather than recall: make 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.
7. 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.

8. 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.

9. 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.

10. 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.

Here instead, we list Ben Shneiderman's 8 Golden Rules:

1. Strive for consistency: layout, terminology, command usage, etc.

2. Cater for universal usability: recognize the requirements of diverse users and technology. For instance add features for novices eg explanations, support expert users eg shortcuts.

3. Offer informative feedback: for every user action, offer relevant feedback and information, keep the user appropriately informed, human-computer interaction.

4. Design dialogs to yield closure: help the user know when they have completed a task.

5. Offer error prevention and simple error handling: prevention and (clear and informative guidance to) recovery; error management.

6. Permit easy reversal of actions: to relieve anxiety and encourage exploration, because the user knows s/he can always go back to previous states.

7. Support internal locus of control: make the user feel that s/he is in control of the system, which reponds to his/her instructions/commands.

8. Reduce short-term memory load: make menus and UI elements/items visible, easily available/retrievable, ...

Here we list Norman's 7 principles:

1. Use both knowledge in the world and knowledge in the head.

2. Simplify the structure of tasks.

3. Make things visible: bridge the gulfs of Execution and Evaluation.

4. Get the mappings right.

5. Exploit the power of constraints, both natural and artificial.

6. Design for error.

7. When all else fails, standardize.

13 Evaluation Techniques

Evaluation tests usability and functionality of a system. It occurs in laboratory, field and/or in collaboration with users. Evaluation is done on both design and implementation and should be considered at all stages in the design lifecycle. The goals of evaluation are:

- assess extent of system functionality
- assess effect of interface on user
- identify specific problems

13.1 Evaluating Designs

There are three main evaluation techniques that we will examine in this chapter.

13.1.1 Cognitive Walkthrough

It is a technique that evaluates design on how well it supports user in learning task. It is usually performed by expert in cognitive psychology, expert that walks through design to identify potential problems using psychological principles. For each task, walkthrough considers:

- what impact will interaction have on user?
- what cognitive processes are required?
- what learning problems may occur?

13.1.2 Heuristic Evaluation

Usability criteria (heuristics) are identified and design is examined by experts to see if these are violated. Heuristic evaluation debugs design highlighting the problems we should solve to achieve better results. The list of criteria is well defined and standardized.

13.1.3 Review-based evaluation

Results from the literature used to support or refuse parts of design. Care is needed to ensure results are transferable to new design. This kind of evaluation is model-based, in fact cognitive models are used to filter design options.

13.2 Evaluating through user participation

From the title we can easily understand what kind of evaluation is done.

13.2.1 Laboratory Studies

Advantages are:

- specialist equipment available
- uninterrupted environment

Disadvantages:

- lack of content
- difficult to observe several users cooperating

It is an appropriate method if system location is dangerous or impractical for constrained single user systems to allow controlled manipulation of use.

13.2.2 Field Studies

Advantages are:

- natural environment
- context retained (though observation may alter it)
- longitudinal studies possible

Disadvantages:

- distractions
- noise

It is an appropriate method where context is crucial for longitudinal studies.

13.3 Evaluating Implementations

In order to evaluate implementations, we need prototypes of an app, or the whole app itself. *Experimental evaluation* is a controlled evaluation of specific aspects of interactive behavior. Evaluator chooses hypothesis to be tested and a number of experimenta conditions are considered which differ only in the value of some controlled variable. Let's see experimental factors:

- **Subjects**
- **Variables**, things to modify and measure
- **Hypothesis**, what you'd like to show
- **Experimental design**, how you are going to do it

Variables are divided in IV (independent), like interface style, number of menu items and DV, like time taken and number of errors.
Hypothesis is the prediction of an outcome framed in terms of variables, e.g. "error rate will increase as font size decreases". Null hypothesis states no difference between conditions and the aim is to disprove this.
Experimental design can be within groups (each subject performing experiments under each condition, transfer of learning possible) and between groups (each subject performs under only one condition, no transfer of learning).
Analysis of data is something we always have to do. We have to look at data and save the original data before doing any kind of statistics. The choice of statistical technique depends on type of data (discrete or continuous) and information required. Tests can be parametric (assume normal distribution of data, robust, powerful) or non-parametric (do not assume normal distribution, less powerful, more reliable). What information is required? Is there a difference? How big is it? How accurate is the estimate? Parametric and non-parametric tests can answer first of these questions. Experimental studies on groups are more difficult than single-user experiments. In fact, we have problems with subject groups, choice of task, data gathering and analysis.

- Subject groups, larger number of subjects and longer time to settle down, difficult to timetable
- The task must encourage cooperation and perhaps involve multiple channels
- Data gathering is done through several video cameras. Problems are synchronization and sheer volme. A solution could be recording from each perspective.
- Analysis is useful because there is a vast variation between groups. Solutions can be within groups experiments, micro-analysis and anecdotal and qualitative analysis.

13.4 Observational methods

We will now analyze different types of observational methods.

13.4.1 Think Aloud

Think aloud happens when user is observed while performing a task. User is asked to describe what he is doing and why, what the think is happening etc. Advantages are simplicity, providing of useful insights and showing how system is actually used. Disadvantages consist of subjectivity, selectivity and act of describing that may alter performance.

13.4.2 Cooperative evaluation

This is a variation of think aloud. User collaborates in evaluation and both user and evaluator can ask each other questions throughout. Additional advantages are that it is less constrained and easier to use, user is encouraged to criticize system and clarification is possible.

13.4.3 Protocol analysis

- **paper and pencil** – cheap, limited to writing speed
- **audio** – good for think aloud, difficult to match with other protocols
- **video** – accurate and realistic, needs special equipment, obtrusive
- **computer logging** – automatic and unobtrusive, large amounts of data difficult to analyze
- **user notebooks** – coarse and subjective, useful insights, good for longitudinal studies

- Mixed use in practice.
- audio/video transcription difficult and requires skill.
- Some automatic support tools available

13.4.4 Automated analysis

- Workplace project
- Post task walkthrough
  - user reacts on action after the event
  - used to fill in intention
- Advantages
  - analyst has time to focus on relevant incidents
  - avoid excessive interruption of task
- Disadvantages
  - lack of freshness
  - may be post-hoc interpretation of events

13.4.5 Post-task walkthroughs

- transcript played back to participant for comment
  - immediately → fresh in mind
  - delayed → evaluator has time to identify questions
- useful to identify reasons for actions and alternatives considered
- necessary in cases where think aloud is not possible

13.5 Query Techniques

Query techniques are used to gain information from possible users of the app. *Interviews* happen when analyst questions user on one-to-one basis usually based on prepared questions. It is informal, subjective and relatively cheap. Advantages include that it can be varied to suit context, issues can be explored more fully and it can elicit user views and identify unanticipated problems. *Questionnaires* are a set of fixed questions given to user and advantages are that they are quick and reaches large user groups and can be analyzed more rigorously. Disadvantages are less flexibility and less probing.

13.6 Physiological methods

Eye tracking is one of such methods where head or desk mounted equipment tracks the position of the eye. Eye movement reflects the amount of cognitive processing a display requires. Measurement include fixations (eye maintains stable position, number and duration indicate level of difficulty with display), saccades (rapid eye movement from one point of interest to another) and scan paths (moving straight to a target with a short fixation at the target is optimal).
Physiological measurement is an emotional response linked to physical changes. These may help determine a user's reaction to an interface. Measurement include heart activity, swat glands activity, muscle and brain electrical activity... There is some difficulty in interpreting this data, research is required.

Choosing an Evaluation Method

when in process:	design vs. implementation
style of evaluation:	laboratory vs. field
how objective:	subjective vs. objective
type of measures:	qualitative vs. quantitative
level of information:	high level vs. low level
level of interference:	obtrusive vs. unobtrusive
resources available:	time, subjects, equipment, expertise

14 Measuring learnability

Usability assesses how easy UIs are to use. ISO defines usability as "The extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use". It consists of five criteria, but we will examine learnability only.
Learnability is the capability of the software product to enable the user to learn its application. However, there is no consistent agreement on how the concept of learnability should be defined. Some common features that are related to it can be identified:

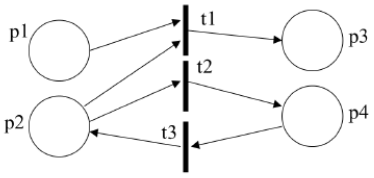
- learnability is a function of user's experience it depends on the type of user for which the learning occurs (experience vs novice users)
- learnability can be evaluated on a single usage period initial learnability or after several usages (extended learnability)
- learnability measures the performance of a user interaction in terms of completion times, error and success rates or percentage of functionality understood

This lack of consensus has naturally led to a lack of well accepted metrics for measuring learnability. Some metrics exist, but they are limited to measure specific aspects of the interaction. Qualitative, mental and quantitative metrics are the ones we just said. All these measurements are performed in controlled lab environments under the guidance of an external evaluator. However, these measurements are good for initial learnability only. Evaluating extended learnability traditionally requires expensive and time-consuming techniques for observing users over an extended period of time.
In order to objectively quantify the extended learnability of interactive systems during their daily use, we define an interaction model that represents the expected way of performing the task (Petri nets), then we record the user's observed behavior during the interaction with the system in a specific user log, and then we introduce the concept of alignment to check if the observed behavior as recorded in the user log matches the expected behavior as represented in the interaction model.

14.1 Petri nets

A *Petri net* takes the form of a directed bipartite graph where the nodes are either places (circles) or transitions (squares or vertical lines). Arcs connect these states.

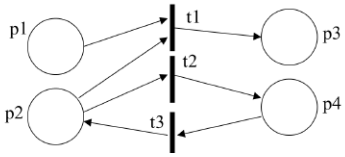
- Formally a Petri net **N** is a triple **(P, T, F)** where:
  - P is a finite set of places
  - T is a finite set of transitions where $P \cap T = \emptyset$
  - $F \subseteq (P \times T \cup T \times P)$ is the set of arcs known as the **flow relation**



$P = \{p_1, p_2, p_3, p_4\}$
$T = \{t_1, t_2, t_3\}$
$F = \{(p_1, t_1), (p_2, t_1), (t_1, p_3), (p_2, t_2), (t_2, p_4), (p_4, t_3), (t_3, p_2)\}$

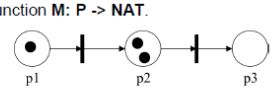
- A directed arc from a place **p** to a transition **t** indicates that **p** is an **input place** of **t**. Formally:
  - $\bullet t = \{p \in P \mid (p, t) \in F\}$
- A directed arc from a transition **t** to a place **p** indicates that **p** is an **output place** of **t**. Formally:
  - $t \bullet = \{p \in P \mid (t, p) \in F\}$
- With an analogous meaning, we can define **input/output transitions** of a place. Formally:
  - $p \bullet = \{t \in T \mid (p, t) \in F\}$ and $\bullet p = \{t \in T \mid (t, p) \in F\}$

$t_1 \bullet = \{p_3\}$; $\bullet t_1 = \{p_1, p_2\}$;
$\bullet p_2 = \{t_3\}$; $\bullet p_1 = \varnothing$;
$p_2 \bullet = \{t_1, t_2\}$; $p_1 \bullet = \{t_1\}$



Marking

- The operational semantics of a Petri Net N = (P,T,F) is described in terms of particular marks called **tokens** (represented as black dots).
- Places in Petri Nets can contain any number of tokens.
- Any distribution of tokens across all of the places is called a **marking**.
  - A marking is a function **M: P -> NAT**.



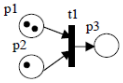
A marking corresponds to a multi-set of tokens.

- The marking of a Petri net determines its **state**.
  - State of the example Petri net: $M = \{(p_1,1),(p_2,2),(p_3,0)\}$ or $M = [p_1, p_2^2]$
- Petri nets must be associated with an **initial marking M_0** and with a set of possible **final markings**.

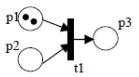
Firing Rules

- The dynamic behavior of Petri nets is characterized by the notion of **firing** (*transition execution*). A transition can **fire** whenever there are one or more tokens in each of its input places.

1. A transition t is said to be **enabled** if and only if each input place p of t contains at least one token. Only enabled transitions may fire.
   - A transition t is enabled in a marking M iff for each p, with $p \in \bullet t$, $M(p) > 0$.



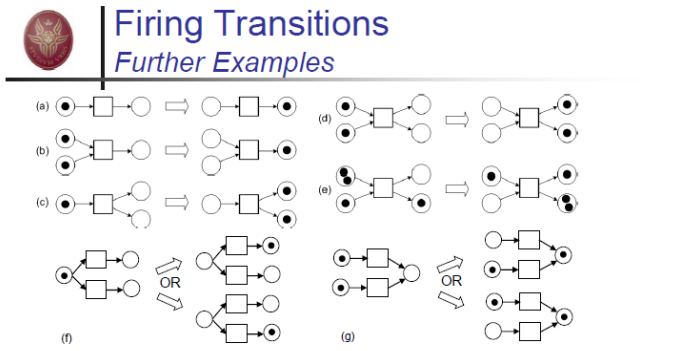
T1 is enabled and may fire!



T1 is not enabled!

2. If transition t fires, then t **consumes one token** from each input place p of t and **produces one token** for each output place p of t.

T1 fires! When a transition fires, the marking and the state of the Petri Net change.



- It is assumed that the firing of a transition is an atomic action that occurs instantaneously and **can not be interrupted**.
- If there are multiple enabled transitions, any one of them may fire; however, for execution purposes, it is assumed that **they can not fire simultaneously**, see example (g).
- An enabled transition is not forced to fire immediately but can do so at a **time of its choosing**.

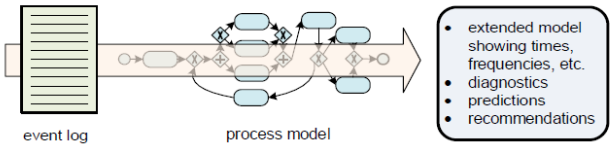
Boundness

- A marking M' is called **reachable** from marking **M** (we write $\mathbf{M} \rightarrow^{*} \mathbf{M'}$) iff there is a firing sequence σ that leads from **M** to **M'**.
- Reachability analysis suffers from the state explosion problem.
- To make reachability analysis possible, a **boundness assumption** is required.
- A Petri net N with initial marking M_0 is **k-bounded** iff for every reachable marking M, $M(p) \leq k$ (k is the minimal number for which this holds).
  - A 1-bounded net is called **safe**.
  - The property of **boundness** ensures that the number of tokens cannot grow arbitrarily.
- In HCI, the focus is on **1-bounded Petri Nets**.

Interaction models like these nets are not enforced by software systems. Traces can be dirty, with redundant or missing actions. Any execution of a relevant task produces a new execution trace recorded in a user log.

Replay

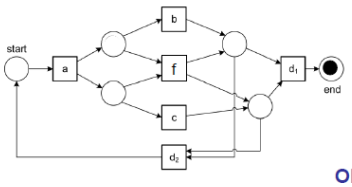
- **Replay** approaches use a ***user log*** and an ***interaction model*** (that may have been constructed by *hand* or *discovered*) as input. The user log is ***replayed*** on top of the interaction model.
- In this way, **discrepancies** between the log (observed behavior) and the model (expected behavior) can be ***detected*** and ***quantified***.



Trace alignment

- Investigate relations between **moves in the log** and **moves in the model** to establish an alignment between the model and a trace.
- **Replay**: If a move in the log cannot be mimicked by the model and vice-versa, such "no moves" are denoted by > (and may have a **cost**).

Alignment of **t = <adbc>**



Move in the model (only)

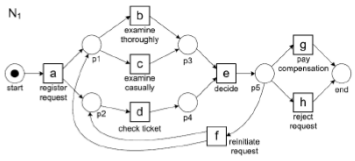
a	d	b	c	>
a	>	b	c	d

OPTIMAL ALIGNMENT

Move in the log (only)

Several possible alignments

<abdeg>



a	b	d	e	g
a	b	d	e	g

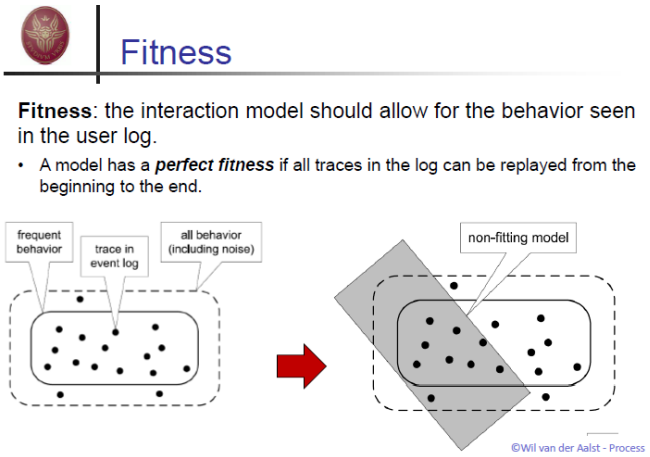
a	b	»	d	e	g
a	»	c	d	e	g

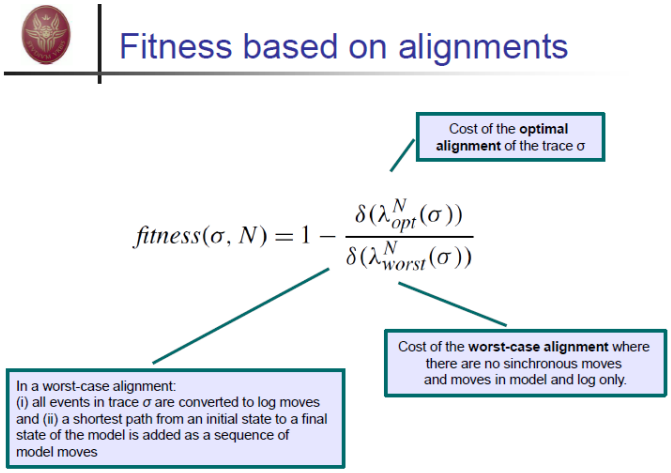
a	b	d	e	g	»	»	»	»	»
»	»	»	»	»	a	c	d	e	g

Trace alignment allows to:

- verify if a trace is compliant with an interaction model
- identify the root and the severity of each deviation

The result of the alignment is the fitness value, i.e. how much the log adheres to a model of interaction (it can vary from 0 to 1).
To measure the learnability of a system we can analyze the rate of the fitness values corresponding to subsequent executions of the system over time. An increasing rate will correspond to a system that is easy to be learnt with respect to its relevant tasks. Given an initial low fitness, a not increasing or stable rate may indicate the presence of some learning issues that need to be fixed.





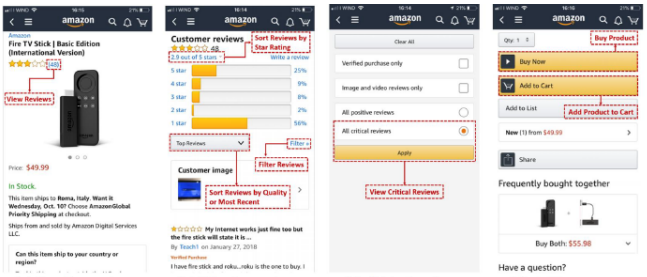
14.2 DECLARE

From Petri Nets to Declare

1. **What is DECLARE?**
   Formal semantics grounded in Linear Temporal Logic (LTL) that has been proven to be adequate for designing interaction model
2. The use of declare models enables an HCI designer to define just the behavior of interest, making it easier to define the models
3. Models expressed as set of constraints, such that everything that does not violate the model is accepted

Constraint	Explanation	Examples
Existence constraints		
EXISTENCE(a)	a occurs at least once a	✓ bcac × bcc
ABSENCE(a)	never occur	✓ bcc × cac
Relation constraints		
RESPONSE(a, b)	If a occurs, then b occurs after a	✓ caacb × caac
PRECEDENCE(a, b)	b occurs only if preceded by a	✓ cacbb × cabb
Mutual relation constraints		
COEXISTENCE(a, b)	If b occurs, then a occurs, and vice-versa	✓ cacbb × cac
SUCCESSION(a, b)	a occurs if and only if it is followed by b	✓ cacbb × bac
CHAINSUCCESSION(a, b)	a and b occur if the latter immediately follows the former	✓ cabab × cacb
Negative relation constraints		
NOTCOEXISTENCE(a, b)	a and b never occur together	✓ cccbbb × acbbb
NOTSUCCESSION(a, b)	a can never occur before b	✓ bbcaa × aacbb
NOTCHAINSUCCESSION(a, b)	a and b occur if a does not immediately follows b	✓ acbacb × abcab

Capturing user actions with Declare



(a) Reading reviews. (b) Filtering reviews. (c) Critical reviews. (d) Adding to cart.

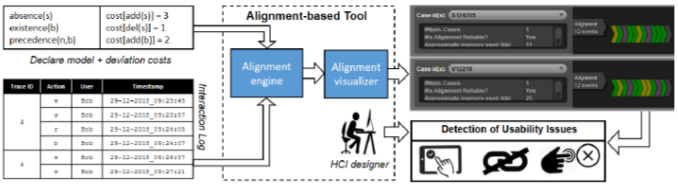
Some screenshots of the UI of Amazon shopping mobile app. In this example, we focus only on a subset of possible user actions such as the user reading only the critical reviews associated to a Fire TV stick, and then purchasing it.

Interaction models via Declare

If we consider our running example, we can specify the interaction model that describes the expected behavior underlying the relevant task, such as the user only reading the critical reviews associated to a Fire TV stick, and then proceeding to buy it, as the set consisting of the following declare constraints:

- **absence(s)** means that action s = Sort Reviews by Quality or Most Recent cannot ever be performed.
- **existence(b)** means that action b = Buy Product must be executed at a certain point of the interaction.
- **precedence(n; b)** forces n = View Critical Reviews to precede b = Buy Product.

The declarative approach [16]



Conclusion and Future Works

- Our approach couples interaction models represented as Petri nets with the notion of alignment wrt. user logs to obtain a more precise measurement of the learnability of interactive systems.
- **STRENGTHS**
  - The approach can be enacted while the system is used in **real user contexts**.
  - The ability of associating **different weights** to the deviations identified during an alignment.
  - For a specific system's task, **different interaction models can be developed** to represent the different interaction strategies of novice and experienced users
- **LIMITATIONS**
  - Only few interactive systems provide a structured recording of user logs.
  - There should be one or more identifiable entry/exit points in the user log for extracting the specific traces associated to the task.
- **FUTURE WORKS**
  - Further robust longitudinal studies to be performed in longer time frames.
  - Identification of threshold values for the fitness.