**Interacció i Disseny d’Interfícies**

**Definitions:**

HCI: Human Computer Interaction, is a field that deals with the study (to improve) of how humans interact with machines / devices. HCI is a very relevant issue when evaluating the quality of an application. An application must fulfill its requirements and it has to provide an easy access to its features.

When an application is difficult to use, it is perceived as a low-quality application.

UI: User Interfaces, are tools and methods that are used to communicate between the user and the system. It may be determinant on ease-of-use perception of application.

HCI is about understanding and critically evaluating the **interactive technologies** people use and experience and understanding contemporary **human practices and aspirations**.

The software crisis in the 70s lead to focus software engineering with a new view, including non-functional requirements such as **usability and maintainability**.

Initial models of HCI

One of the original focus of HCI was usability.

* Originally stated as “easy to learn, easy to use”.
* More on this later today…
* GUI: comprehensible, accessible, easy to use.

Helped to influence computer science and technology development more broadly and effectively.

It grew to include other areas, not restricted to computer science.

Usability: defined in ISO 9241 standard as the ability in which a product may be used by specific users in order to carry out specific tasks **effectively, efficiently, and with satisfaction** in a specific use environment.

Usability is always referred to a concrete user group and a concrete user application.

* Efficacy is the ability of correctly and completely achieving a certain goal.
* Efficiency is the relation of used resources and the completeness and correctness of achieved goals.
* Satisfaction is the comfort and acceptation of a system by the users and other people that are affected by its use.

UX: User Experience, is not about technology, industrial design, or interfaces. It is about creating a meaningful experience through a device. The perception left in someone’s mind following a series of interactions between people, devices, and events. What you remember and feel from the use of a device.

Interaction Design: is about shaping digital things for people’s use. How we interact with devices (“digital things”). For example: unlock the phone with your face…

Desktop Systems:

* Large screens.
  + Space for everything.
* Mouse pointer.
* Keyboard.
  + Adequate for creating content.

Mobile Systems:

* (Relatively) small size.
  + Must carefully think on what to fit.
  + Notifications often not properly solved.
* Interaction with the finger / stylus.
* (Almost) no keyboard.
* Software limitations.

Tablet Systems:

* (Relatively) large size.
  + May fit what we need.
* Interaction with finger / stylus.
* (Almost) no keyboard.
* Software limitations.

Tools for Mobile Development:

* Native tools:
  + Provided by the OS designers.
  + Focus on the OS features.
* Cross-platform:
  + Provided by third-party institutions.
  + Focus on facilitating the development.
* Other third-party software:
  + Focus on facilitating the development.

Two main ways to develop:

* Web apps.
* Native OS apps.

Web apps:

Pros:

* Develop once and deploy everywhere:
  + Almost any system has a capable browser.
* Easy updating:
  + App is loaded every time the browser connects to the page, only needed to change the server code.
* Well-known tools and techniques (PHP, Java…).

Cons:

* Limited user interfaces.
* Not as rich as native apps in terms of:
  + UI, Communication, Access to local resources (camera, GPS, …).
* Inefficient and insecure communication protocol.
* Mainly designed for large displays with mouse.

Native apps:

Pros:

* Richer UI.
* Many controls.
* Safe and fast access to local resources:
  + GPS, camera, files…
  + Efficient communication.
  + Any protocols allowed.
* Smaller variety in languages and tools (SDK).
* Designed for small screens and touch controls.

Cons:

* No universal access:
  + Each OS has a different app format and development environment.
* Difficult to manage updates:
  + Require individual (user guided) updates per device.
* Less general than desktop programming:
  + Though a lot of new material is on the web.

**Design Principles**

Effective interfaces:

* Instilling in their users a **sense of control**.
* Do not concern the user with the inner working of the system.

Effective applications:

* Perform a maximum of work while requiring a minimum of information from users.

**1. Aesthetics:**

Fashion should never trump usability.

* Aesthetical appearance is appealing.
* But design based on fashion will artificially generate obsolescence.
* A new fashion should not detract from user-performance.
* The current trend of contrast reduction is really painful (see examples).
* In any case, user test: users know standard controls. Non-standard controls require cognitive effort and produce errors.

Fashion (gestures) trumping usability (lack of affordances).

Text that must be read should have high contrast, the text should be read perfectly without much more effort for the user (good size of the letters, good contrast, good size…).

* Favor black text on white or pale-yellow backgrounds. Avoid gray backgrounds.
* Use font sizes that are large enough to be readable on standard displays.
* Favor particularly large characters for the actual data you intend to display.
  + As opposed to labels and instructions.

But sometimes high contrast can be aesthetically pleasant, so it is important to have the adequate contrast.

**2. Anticipation:**

Bring to the user all the information and tools needed for each step of the process.

* Should anticipate the user’s needs.
* Information in place & visible (if the user cannot find it, it will never be used).
* Requires deep understanding of both the task domain and the users.
* The penalty may be the complete lost of the user or client.

In any case, user test.

**3. Autonomy:**

Computer interface, and task environment all “belong” to the user but user-autonomy doesn’t mean we abandon rules.

* Give users some breathing room
  + e.g., provide a certain degree of customization –desktop-.
* Enable the users make their own decisions.
  + Otherwise, they may feel constrained and frustrated.

Keep the user informed:

* Autonomy/Control cannot be exerted in the absence of sufficient information.
  + Provide information on current state, tasks.
* Keep the information timely, and accurate.
  + Progress indicators that are inaccurate are annoying, e.g., updated indicator showing a 5’ task that turns and hour!!!!
  + Lying the user is never a good practice.

Progress indicators:

* Should appear timely and be informative.
  + Tasks that require >1seg => cursor busy.
  + Tasks that require enough time (>10seg) should show progress indicator.
    - Indicate the computer is busy.
    - Should estimate time or % of task.
    - Tell users what’s happening.
    - Let the user do another thing.

Error messages:

* Should provide information on the reason for the error.
* Should provide a clue on how to act.
* Should not be fast enough so that the user cannot read them.

**4. Color:**

When using color to convey information in the interface, also use clear, secondary cues.

* Approximately 10% males and up to 1% females have some form of color blindness.
* With age, people start having vision problems.
* Use websites such as http://enably.com/chrometric/ to test.
* Use color, but use it wisely.

In any case, user test after aesthetic color changes.

Warning: color meaning can change depending on the culture, one color may have a special meaning in a specific culture.

**5. Consistency:**

Must be analyzed in different dimensions:

* Levels of consistency.
* Induced inconsistency.
* Induced continuity.
* Consistency with user expectations.

Levels of consistency:

* Platform consistency.
  + Guidelines (e.g., UX Android, iOS…) and in-house (same company…).
  + Unwritten rules (assumed by the community).
  + Keep a general look & feel across products/services.
    - Communicates brand.
    - Makes adoption easier.
* Across suite of products.
  + Communicates family (e.g., Microsoft Office, Google apps).
* In-app: in a single app/web.
  + Specific look and feel.
* Visible structures: Icons, symbols, etc.
  + The appearance must be strictly controlled.
  + Positioning must also be similar.
    - Ensures what people learn is valid across the app/webpage.
    - Improves learn ability.
* Invisible structures.
  + If implemented, make them strictly consistent everywhere.
  + In any case, using invisible structures (hello Microsoft, hello Apple) just makes their use obscure and difficult.
    - Expecting the user will google for learning your product features is not the solution.
* Interpretation of the user behavior.
  + Never change the meaning of a habitual action.
    - It is one of the worst things you can do to the user.
    - User take a long time to learn things.
    - Such actions become subconscious with time, for example, changing a learnt gesture is extremely frustrating.

Induced inconsistency:

Make objects different if they act different.

* E.g., if the trash can is destroying the document, make it appear different than a trash can.

If your app/webpage has changed substantially, design can be changed to enforce this fact.

* Otherwise, the user might not notice and continue using the app/webpage the same way (and it might not work).

Induced continuity:

Over time, strive for continuity, not for consistency.

* If your renewed app does a lot of different things and the look and feel is exactly the same, users will use it the same old way.

New versions of products may change big areas (e.g., new features…).

* Make them slightly different from previous version.
* Previous knowledge may serve the user to guide their path.
  + E.g., maintain the familiar look: same button icons for the things that have not changed…

Consistency with user expectations:

If users expect something in a certain way, do not force them to learn a new way.

* Even a new button/task may have some user expectations.
* If all the users expect the same, and it is different from what you are offering, go for the path of minimum resistance.
* It doesn’t matter how fine a logical argument you can put together for how something should work.

Unless your new way offers clear advantage (e.g., Xerox Drag rule).

Green and red colors have meaning.

* Using them inconsistently leads to larger reaction times.

**6. Default values:**

Avoid the cursor appear in unpredictable positions.

Should be easy to rewrite.

* E.g., automatic selection of the default text on a field.

Not all fields require a default value.

* If no clear winner/advantage, do not put a default value at all.

**7. Discoverability:**

Any attempt to hide complexity will serve to increase it.

* Generating the illusion of simplicity actually does not simplify things.
  + E.g., invisible Mac scroll bars.

If the user cannot find it, it does not exist.

* Note the difference between the buyer and the actual user.

Use active discovery offer (not tested) features to the user.

* Guide people to more advanced features.
* Mention a feature that exists.
* Recall it at intelligently spaced intervals.
* Stop mentioning it once explored or adopted.

All controls should be visible and not over the content area.

* Some exception only if space is limited (e.g., smartphones or tablets).
* Should provide a standard trigger that will expose all controls.
  + Don’t do this in desktop.
* Communicate your gestural vocabulary with visual diagrams.
  + On-screen hints or training required.

User test for discoverability.

Apple example: Invisible scroll bars.

* Scroll bars serve two different purposes:
  + Informing.
  + Navigating.
* Making them invisible does not inform the user on the relative position.
  + Or where there is more content to be explored.
* Even worse: Apple renders scroll bars over the contents!
  + Making them occlude or prevent their selection.

**8. Efficiency:**

Look at the user’s productivity, not the computer’s.

* Keep the user occupied:
  + Typically, the highest expense by far in a business is labor cost.
  + Maximize every user efficiency.
  + Don’t improve IT’s productivity by pushing work to users.
  + Think organization-wise.
* The great efficiency breakthroughs in software come from fundamental architecture changes.
  + Not in the surface design of the interface.
* Don’t ask for the same information twice.
  + It is very annoying and unproductive.
  + Store it for future use.
    - Sometimes not done due to programmers’ laziness.
* Gather the information as needed
* Provide default values when possible.
  + May accelerate interaction.
  + May help novice users.
  + Provides tips for valid values, formatting.
  + Sometimes the answers will be valid in multiple cases.
    - Default folder for program installation.

Error messages should actually help.

* Explain what’s wrong.
* Tell the user specifically what to do about it.
* Leave open the possibility the message is improperly being generated by a deeper system malfunction.

“Error-1264” does not mean anything to the average user.

**9. Explorable interfaces:**

Users want to feel free when exploring interfaces.

* Two important principles that facilitate exploration are:
  + Make Actions reversible.
    - E.g., “Back” in a webpage, cancelling long actions…
    - Promotes exploration.
    - Otherwise, a perfect user is a slow user.
  + Always allow “Undo”.
    - Otherwise, you need to confirm everything (Again, slow).
  + Users should always have an easy way out.

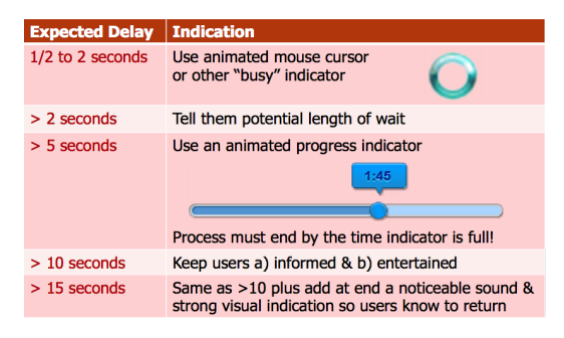
**10. Fitts’s Law:**

The time to acquire a target is a function of the distance to and the size of the target.

* Large objects for important functions.
* Small objects for functions you would prefer users not perform.
* Reduce the number of targets to acquire.
  + Not only their distances.

**11. Informing users:**

Keep users informed when they face delay.



Acknowledge all button clicks by visual clue within 50 ms.

Start making everything faster.

* Eliminate any element of the application that is not helping.
* Be ruthless.

Wearables come with an even higher level of expectation:

* No one waits to see what time it is.
* Or to see who is calling, what the temperature is outside…

**More principles for usability:**

Choose metaphors that will enable users to instantly grasp the finest details of the conceptual model.

Try making your concepts visually apparent in the software itself.

* Buttons are pressed, sliders dragged…

Expand beyond literal interpretation of real-world.

* If a metaphor is holding you back, abandon it.

Ensure that users never lose their work.

* This principle is all but absolute.
* Users should not lose their work as a result of:
  + error on their part,
  + the vagaries of Internet transmission,
  + or any other reason other than the completely unavoidable (e.g., travel sites).