UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Master's Degree in Computer Science

Academic year 2023/2024

WEB INFORMATION MANAGEMENT

Prof. Massimo Marchiori

Written by Michael Amista'

Table of contents

1. Introduction to a website

1.1 The problem of "time"

There's a parallelism between a website and a shop. In both there's a window (homepage) that shows what the website/shop contains/sells.
What do the people expect from a homepage? Many things, but the most important is the **information**. The problem of information synthesis has not been invented with web pages, it's a common problem in *information communication* (the problem was born with journal articles).

The most important information components of a website homepage: **INFORMATIVE AXES**:

- **WHERE**: what kind of site I arrived at, the content.
- **WHO**: who represents the site.
- **WHY**: why should I stay here? Which are the site benefits?
- **WHAT**: what the site offers?
- **WHEN**: latest news, the news of the site.
- **HOW**: how to get to the main sections of the site.

That's the **core of information**, what the people want to know. If these parts are missing the information is uncomplete. Users want to know this information and so we need to convince the user to surf on our website.

But that's not so easy to fill the homepage with this information. **There's a "small" problem called "TIME"!** Users have expectations and have **limited time**. It's easy to put everything on a webpage but people don't have the time to read/see everything.

A user arrives to our homepage, and he spends on average, **31 seconds**. So, we have on average 31 seconds of time to convince the user to stay in our site and show him the information components. And the way to do it is to **compress the information**.

For example, *how much text can we put in our home page?* It depends on how much faster the people read but on average from **200 to 300 words per minute which go down to 180 in case of computer screen reader** because it reads slowly than a human. So, if we put more than 93 words we have finished our time. They should be quite less, because the user doesn't spend all the time just to read, but there is also time lost to analyse the visual layout, images, links and more.

We want not only he coming to our site, but we also want him to come back! So, what about the returning user? Same expectations? He knows you already, so the limited time for the home page will not be used for the axes WHERE, WHO and WHY. The time will be spent only on the remaining axes, WHAT, WHEN and HOW.

The disadvantage of returning user is more demanding, and he has less time to spend for us.

Time on homepage: first visit (31 seconds) / second visit (25 seconds) / third visit (22 seconds)

So, if we want to make our returning users happy and satisfied, we have a little treasure of **19 seconds** for our homepage, to split among the WHAT, WHEN and HOW parts. That is a maximum of **57 words for all these components**. The user will skip the parts that he already knows.

When I open a website, the main information should be easily detectable, try with this:

Too hard! It's not clear! We can do better reorganizing the homepage into sections and rewriting things in a better way:

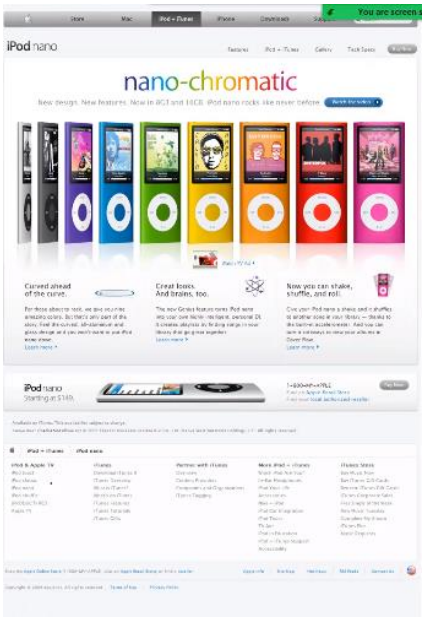
And after the home page? If we have success the user goes on to the other pages of the site so we cannot put everything into the homepage.

What must we care about in the other pages? Same thing of the homepage? NO, the user already knows where he is, so no more need for all the WHERE, WHO, WHY, WHAT, WHEN, HOW axes.

Once we attracted someone inside our site, we have the advantage that typically they are less likely to go away. On average, **the returning user stays more time**. From the 31 seconds of the homepage, we pass to **53 seconds**! That's because the user entered in the shop/site, he looked at the window/homepage and now he's inside.

This additional time allows us to add information, so having pages that are more specific than the homepage. But always remember to pay attention and don't exaggerate: there are always the max limits on text ➔ 53 seconds corresponding to **159 words**.

For instance, that's the old version of the Apple website, the red box on the left figure contains the 159 words (53 seconds) that consume all the user's time! Users do not have the opportunity to know the advantages of the new iPod because that information is much later in the text.



The site has been redesigned (right figure). A lot of information to share? ***Learn more...*** after every "long" section. It's a user choice to know more about a feature, you'll be redirected to another webpage with another 53 seconds to spend. Every page must be designed on 53 seconds of reading. **The user chooses what to do!**

There is another time concept that is very important for users. **The global time is the time it takes the user to be satisfied: he found what he wanted on our site**. For instance, in the iPod example the global goal is to buy a new iPod and the global time is the time I spend to buy the iPod. The global time goes over the single page, it's the experience.

The first global time is the time by which the user has got an idea on us and decides whether to stay with us (to the final goal) or to go. It's the so-called **choice time**. The choice time is **1 minute and 49 seconds**.

When a user goes away because he's not satisfied, we probably (88%) lose him forever. Have the best price or product is not enough, the user will go away anyway if he's not satisfied.

The second global time is when the user expects to have found what he wanted, and so to have successfully navigated our website. This limit is **success time** and it's about **3 minutes and 49 seconds**.

So, what matters most is the good balancing among *homepage, internal pages* and *trail (path)* that the user follows to arrive where he wants. Moreover, given the success time (3m 49s), he expects to reach his goal after having seen the homepage and about three pages and a half of our site.

1.2 The importance of structure

For every goal we can build the best path for the user to achieve it. The **tree structure** of our site then becomes critical, as well as the distance from the home page. From the homepage, after **one click** (max 2) we have to convince the user, and after other **two clicks** (max 3) we must give them what they wanted! We have to offer shortcuts to the goal.

The structure of the web has changed: whereas time ago navigation always usually started from the homepage, nowadays this is not true anymore → to search x, navigation can start from any place (think about Google research), this is called **deep linking**. The **user can see an internal page before the homepage and he cannot see the homepage never**. For us this is a challenge!

The situation is therefore more complex: **each page can be the first page that a user sees**. So, let's see what happens to the axes: some axes become mandatory, others optional, and others can be omitted.

Mandatory informative axes:

- **WHO** (typical shortcut: logo in the upper-left part)
- **WHAT** (typical shortcut: direct link to the home page)

Optional informative axes:

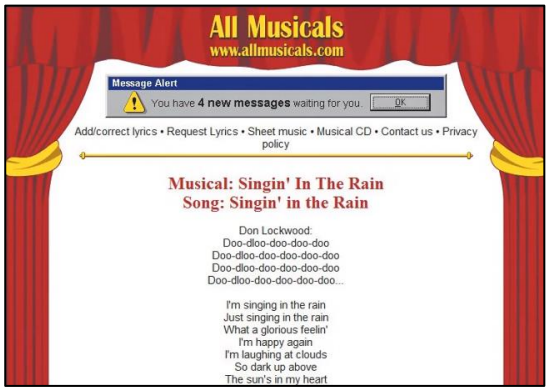
- WHEN (completely optional)
- WHY (suggested!) == short description
- HOW (suggested!) == typical shortcut: the search functionality (preferred position up-right part). Suggest inserting correlated pages.

The most important informative axe is **WHERE** because the user is "thrown" in the middle of our information forest → typically **we should make clear the context**.

Why WHAT is not enough? To avoid for the user to always go to the homepage, wasting a click.

If the user directly lands on a certain page, we have more information on what he wants! Let's use that information, instead of pushing him back to the home page.

Example: if a user enters in an ecommerce where he can buy a tv, putting a shortcut like "Do you want see other electronic products?" is a good idea to avoid the user goes to the homepage searching manually for other products section (if he's going to do that! With a direct link the probability he will visit that page increases by a lot).



***Typical error*: put the homepage menu in a specific page**.

The user doesn't care about the global functionalities in a specific page. He prefers to know what he can do inside that page and to get all the other website pages we can put a shortcut to the homepage where it is possible to use the main menu.

The techniques that explain to the user where he is, are called **breadcrumb**. Three main kinds of breadcrumbs (we can use just one or all three):

- **Location**: where we are (Home >> News >> News x). It helps users to navigate in the website;
- **Attribute**: shows the attributes of the given page, so it does not necessarily correspond to the location, but contains the path from the main type and its subtypes (for example a webpage that sells electronic components may have HW / graphics cards / nvidia / geforce). Each page has tags that help to understand the categorization;
- **Path**: show the path taken by the user to arrive to the page. Typically, they are dynamical.

The classic separators for breadcrumbs are two: ">" and "/".

2. Usability problems

Can be divided into two main classes: **persistent** and **non-persistent**. The persistent problems are those that didn't change much along time (typically, the worst problems). The non-persistent ones, instead, changed (typically, for the better).

2.1 Persistent problems

When navigating inside a site, an important problem that needs special care is the **lost in navigation**. Users must be always conscious of where they are within the site, so WHERE axis is fundamental.

We also talked about breadcrumbs, but we can add something. Navigation is not only where I am but also where I'll go. **To know where to move next, we also need to know where we have been**. Sure, we can just suppose a user to remember where he has already been but doing so, we pass the burden to the user: remembering this information **makes navigation heavier** → we have to satisfy the user!

For instance, we can use a functionality that is present in browsers since a long time: **change color to links that have been already visited**. Giving more powers to designers, many just forgot why this functionality was introduced → designers don't have idea of the rules of usability, they just use the more elegant thing. So, a big usability problem arose: **don't change colors to already visited links!**

Paradox: user that like our site will navigate more but the more they navigate, the more we ask for them to remember the pages they visited. So, the more they navigate, the more they must memorize, and to get tired. *[It's like a customer must do the work of a casher where he goes to the supermarket → he must remember everything he put in the basket and after he has to pass the items to the counter → it's a stressful experience!]*

Users must be able to move quickly, ***what are the navigation movements that are most used?***

1. **clicking on a link** to go to another page;
2. pressing **back!** Users prefer to navigate back even many times, **even when there is a direct link**. They use it even up to 7 times (7 clicks), instead of doing one direct click → a functionality often forgot.

How so? Instead, users are wasting time? Yes, it's paradoxical, but there is a reason. Users are still **minimizing the computational effort**. The advantages of the back button are related to no need to remember the followed path and the interface is consistent for every site.

So, the corresponding usability problem is then **not to allow the proper use of the back button**. Usually happens with dynamic pages that are badly handled, not saving the navigation state in a way that is compatible with the back button.

Another possible interference, and related severe usability problem, is to **open a new browser window**. Many designers use new windows to try clearly separate new content. Opening a new browser window gives a few problems to the user.

The first bad problem is to **disallow the use of the back button**, with all the consequences seen above the navigation. New windows can typically be of two kinds: a new browser window, or a new tab. They both irritate and confuse users, although the worst one is the first (new window). The worst problem, in this case, is that the page stays above the existing navigation → a lot of windows opened that can hide the other ones opened.

A related problem is the **pop-up**: they are opened **without the user consent**. We will come back later but this type of window is so hated by users to deserve a special mention as a standard alone usability problem.

Another big usability problem is to **violate web conventions**. A web convention is not a web standard but just a technique used by most web sites that users learnt surfing on different websites. Respecting conventions is related to one of the most famous usability laws. **New methods, never used by nobody, can be better but it takes time and effort to be understood from the users**.

Users spend most of their time on other websites so we should not bend users to our will if they are used to other things on the web. If we do otherwise, they need to adapt to our different design → that takes time, energy and leads to frustration and timers' expiration.

Another problem: using too much **empty language or languages with little content and lot of slogans** → that's no propaganda (what works on TV not always works on websites).

[*home page of montblanc pens*]

Another related problem is the *form*. One of the more severe usability problems is to **use difficult and monolithic text**. This often happens also in sites that can ignore user timers and their feelings, for instance public/governmental sites, or private but monopolist.

General web text is different than normal text: besides timers, reading on a screen is more difficult → text should be simplified, so to counterbalance the additional effort.

The rules of web text:

- Base rule: 100% normal text → 50% web text.
- If generalist audience: 100% normal text → 25% web text.
- It's helpful to start with the conclusion and then expand (synthesis of information: I know what I'm going to read and so I can decide to go on or not).

2.2 Non persistent problems

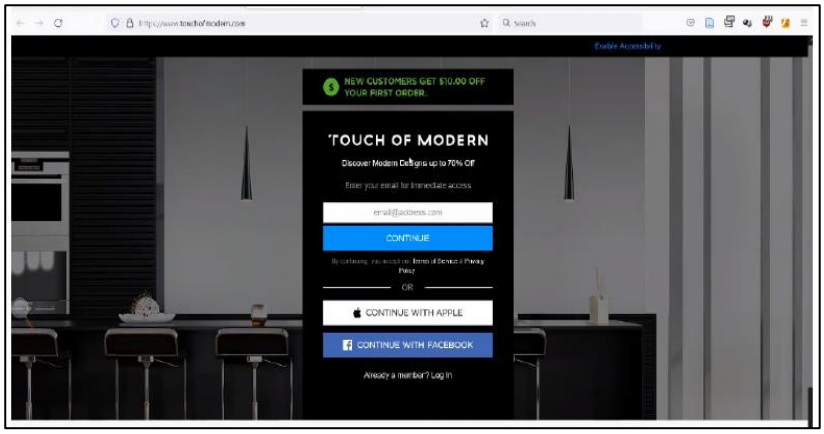
The *non persistent* problems are those that changed in time.

One of this kind of problems are the **splash pages**: a kind of intermedium page between us and the website's home page.

Avoid at all cost: besides not being liked by users, **they make them lose precious time**.

Even worst if animated → it's a non persistent problem because the trend is changing: with faster internet people can think to put everything on this page and this wastes user's time.

Another problem is **asking personal information**.



Instance of the problem is the *premature registration*.

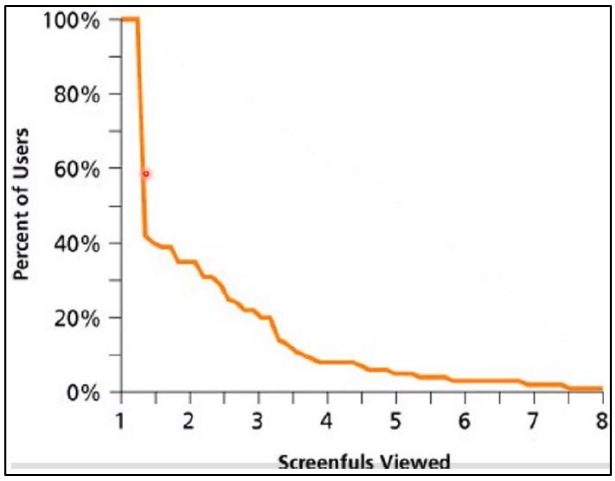
We haven't already seen the website and they ask our information!?

Other examples are the ***pop-up*** that are displayed, for instance, when we are not already registered in the site.

These are to avoid: **how much time is the user losing to give information? The time doesn't stop when a pop-up appears, it goes on!**

In the case of premature registration, there is also the additional disadvantage of the registration, which implies a further **computational effort** to insert and then remember a data pair made by login and password for that service. Also, premature registration brings a decrease of potential users (on average less than 1 out of 10).

Last but not least, the **trust problem**: giving away personal information requires a *trustworthy* site, and in case of premature registration the trust bond has still to be established!



Another problem is **scrolling**. On average, how much users are willing to scroll? 1.3 screens (so seeing a total of 2.3 screens) → they hardly see what is beyond, and when they do their effort increases.

Mostly of the people just see the content of the first scroll.

NOTE: the mobile rules are different because the screen is "longer".

Are there differences between home vs internal pages? Yes:

- First visit to the home page: 23% scrolls
- Internal pages: 42% scrolls
- Repeated visits to the home page: 14% scrolls

The instance of iPod site is perfect: the standard size of a web window is **1024x768** and what happens is the image takes a lot of space **that could be taken by more important information**.

Bad attitude: put oversize image that takes a lot of space.

Users are forced to scroll to see what there is beyond these oversize images.

From laptop to net-books' reference size: 1024x600. But what is the max size? The average *safety size* to consider is 800x600.

The problem of scrolling: **too much information expanded on the vertical screen axis**. What about the horizontal scrolling? When information clashes with the horizontal axis, we have the **frozen layout problem**.

We scroll vertically, so often the horizontal axis is just "frozen", giving a set width or a (too large) minimum width → we have to move the horizontal axis to see beyond the frizzed part. This way, then the window size goes beyond these parameters we have **serious visualization problems**.

For instance, in the old iPod site we have the frozen layout problem, and the hidden components are important for the user, like the hidden search bar.

When the window is smaller than the set/min size (not full screen) then things go wrong for lack of space. In this case to access horizontal information the users need a horizontal scrolling which is one of the most hated things by users → **it is uncommon** on the web, and it is not within our classic information modalities.

In our life, we have always been used to handle textual information with one leading dimension: books, screen pages etc → in these systems, the horizontal axis is always set. This problem also augments the **dimensionality** of the information space: **2 dimensions to manage! →** more computational effort!

2.3 Bloated design

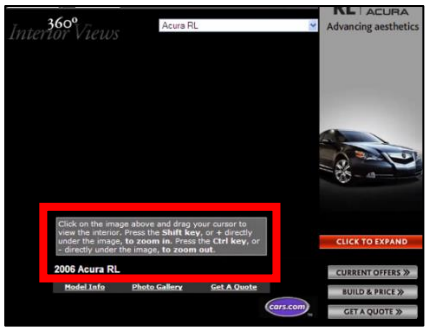
Essentially, building an over-the-edge design, trying to impress people with effects but what one likes can be just terrible for others.

It can be good for spot on TV, but it doesn't work on the web, it's extremely annoying users! Both for aesthetical reasons, and for practical reasons → **computational effort increases!**

The *browser wars* are the perfect example of the bloated design scenario. Browsers wanted to be special and unique from the others, so they created some new effect commands (highlight/move the text in all directions, blink tag, etc) to impress users, things that now are hated.

In the big family of "bloated design", there are also the ***multimedia abuse***. For instance, the one of **3D interfaces**. The 3D interface is a big temptation: what best way to surprise and impress users? **No success** → same principle of the 3D in cinema: great idea but the computational effort to watch a film in 3D is too high for people (both in *perception* and *usability*).

An example of "success": Google Streetview, but how much we use it? Not much because after a certain time we are tired, the problem is that **there isn't a standard interface to interact with 3D technology on the web** and our brain cannot process this type of movement for so long.



Another example is the showroom or the 360° view. There isn't a standard and many websites have to explain how to interact with these webpages. **So, we have to learn something new available <u>only in a specific site</u> because in others the behaviour is different!**

But, over all these problems, there is a good way to present these characteristics on the web. We can shape 3D object idea into 2D snapshots, and these are massively better!

For instance, for clothes market it's common to want to see all the angles of the dress → images gallery is the solution of these types of requests.

Another way to impress users with special effects is using **plug-ins**. But plug-ins suffer from a fundamental problem: they are "plug-in" and as such non-standard → they must be installed first.

Users don't like to install plug-ins for a couple of reasons:

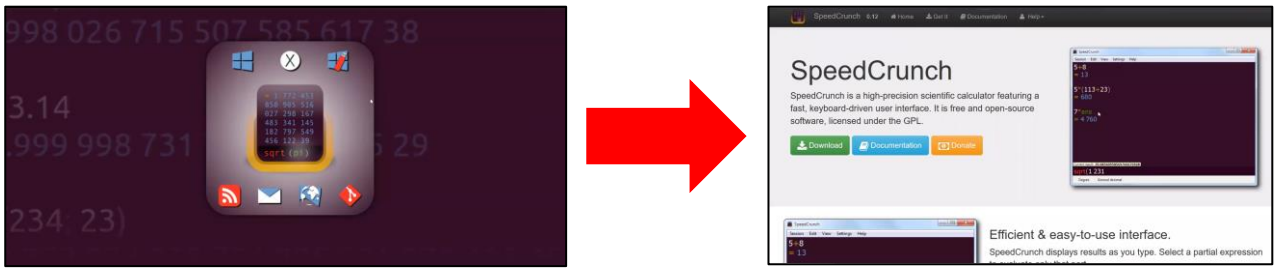
- ⇨ The **TRUST FACTOR**, they don't really know what's going on, so their trust level is very low → **better to do nothing and avoid problems**. They start to gather some trust only one year after their distribution.
- ⇨ The **TIME**, users don't want to waste time. On average, every request of plug-in's install makes for a loss of 90% of the users.

<u>Example</u>: there were two kinds of problems with *Flash plug-in*:

- ⇨ **INTERNAL**: it is always a plug-in so suffers from the typical problems of that category (although with less user loss, due to the wide trust factor → it's not a trust problem). There are also time problems, due to potential high load time and other problems we'll see later.
- ⇨ **EXTERNAL**: flash gives many powerful tools and creative freedom, the risk for the user is to get **bloated design** and so a **higher computational cost**.

On 31 December 2020 Adobe stop the support for Flash. The main hypothesis concerned about these usability problems.

Now we have different standard technologies that let us to use correctly the “flash effects", technologies like HTML5, JavaScript but **use that is not a guarantee that things are done well → we have to put attention**.



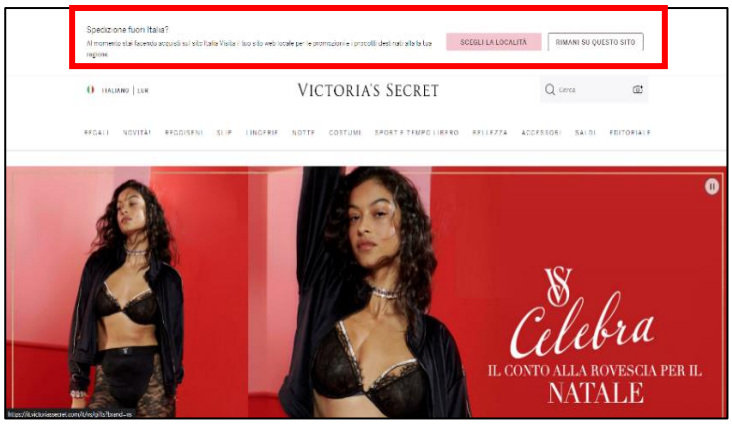
In the above example the first image represents the *SpeedCrunch* website made only by using HTML5 and this is the perfect example of wrong usage of these technologies. In that website all the informative axes are missing, and the general usability is terrible. Then we have the upgraded version where a better work has been done.

So, **do we have multimedia that has low computation cost? And we can use them?** The answer is YES. This is the **VIDEO**: its success depends on its low (almost zero!) computational cost (tv, web streaming) → the “heavier” thing that we have to do is just choose a program.

We can use VIDEO to power up a website. But there are some problems:

- **BANDWITH** needed to handle it. A big error is to handle video on our own, without resorting on other sites like YouTube. If we have to manage a lot of access to our websites and to our videos, we need to have the resources to do that, BUT there are platforms that can do that for us so why don’t use them?
- **TIPICALLY CAN EXCEED USER TIMERS!** Videos can be potentially attractive, but we need to pay attention to user’s time. The preferred average time is 1 minute (at the maximum 2 minutes but we are risking). There are exceptions, in particular if the user has to watch all the video to arrive to the goal.

(A small parenthesis on Victoria’s Secret website):

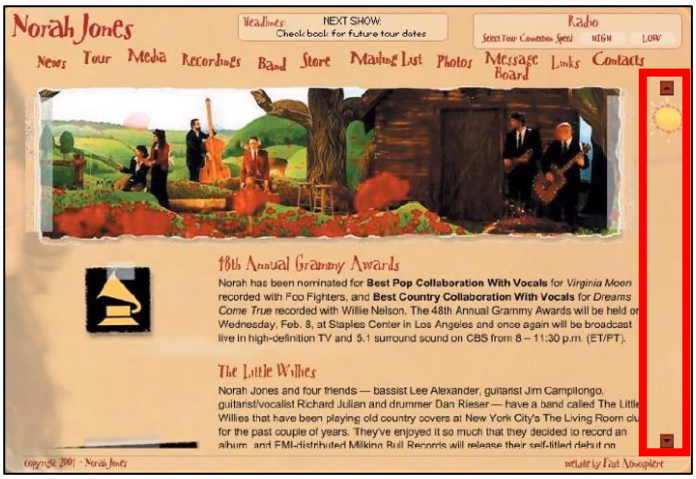


When there’s a difference between the device language and the location of our IP, some websites can ask us if we want to remain to the Italian version or go to another → BUT THIS IS WRITTEN IN ITALIAN and that’s a problem.
There’s also a bloated design here → too large image that change frequently and if I reduce the window there’s also the horizontal scrolling problem.

2.4 Broken visual metaphor

This problem happens when a certain visual metaphor is in place, and users has corresponding expectations. We find this type of problem **when we have a scenario when a webpage broke the user expectations wrongly using visual elements**. These problems make the user wasting time → when he tries an action that he thinks is the right one then he has to recalculate the action because it wasn't what he expected → MORE COMPUTATIONAL EFFORT.

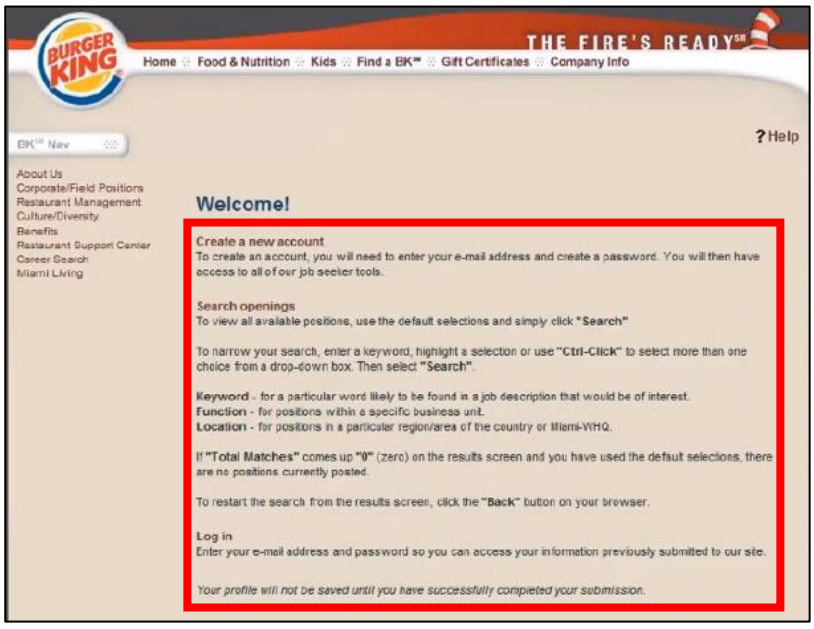
Some examples:



This is a scrollbar but the user doesn't see this → he only sees a sun, just a visual effect, something nice on the website! That's a **broken visual metaphor!**

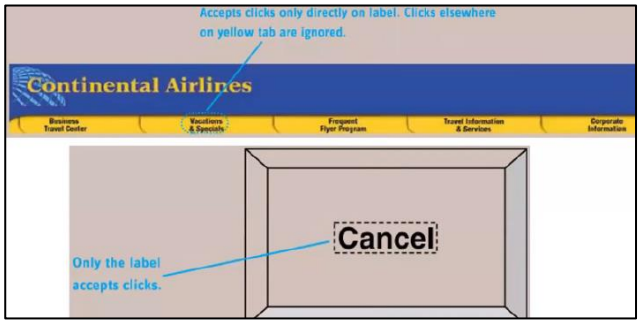
Also happens when a site tells us to click on a product for instance and the click doesn't work because we need to click on the text of the product → broken visual metaphor.

Another one is the erroneous use of the blue bold colour on the text → user think that is a link but it's not clickable → broken visual metaphor. Maybe user find a "link" to reach a goal and it's not clickable → the worst thing.



This information is not clear for the user, why they don't add a link to the mentioned operations??

For these operations you have to click over the images and not over the button → a complete failure!



In the same category we find the **visual metaphor of buttons**. Elements that look like buttons that are not clickable → only the text is clickable but not its container.

Possible scenario: if we have a maximized window the button becomes bigger while the text remains the same and then we have a very large button but it's not clickable if we don't click right over the tiny text.

Other scenario: when we position the mouse over a menu link and it's highlighted → user thinks he needs to click over the highlighted area and not just over the text.

2.5 Excessive convergence between desktop and web

Another problem is excessive convergence between desktop and web. In other words, **the mistake of directly bringing desktop tools to the web**.

For instance, the menu component. It's a powerful desktop tool, and users are well used to it, so the idea is to recreate them inside the web environment.

- ADVANTEGES: no new training, so low computational cost + save a lot of space + fast navigation!
- DISADVANTAGES of this idea are related the nature of menus: in the desktop they show *commands*, in the web instead they show *information*. The structure is always tree-like, but potentially there is much more information.

The situation gets even worst when we think of all the considerations about the ***reference sizes***. Tool1 (the mouse) + Tool2 (the menu) → disasters. 83% of users doesn't center the right item in the web menu. **54% goes out of the menu**. Which means that the frustration level grows along the number of levels.

So, another serious problem is the **movement**. How do users move on the screen (that is, how they move the mouse)? When (average of) users wants to go from A to B in a webpage he ***follows a straight line*** → the faster way (red path in the menu figure) not the longer ones (blue path). **In the case of multilevel (cascade) menus, the max level should be two**. Moreover, the shortest path rule should be considered, and so design ***fault-tolerant*** menus for such paths.

Good way to design a menu: horizontal one with a vertical box (if we have a multilevel menu) which it's opened when we click over one of the horizontal voices.

2.6 Text of a webpages

We already talked about text quantitatively, talking about timers. We now see other important aspects of text in webpages. There are some rules to follow:

1. **Text has to be readable**. We need to pay attention on the font-size, the text must not be too small → BAD PRACTICE: diminish the font size to put more information/fit the layout. Text should have a minimum size: at least 10 points on every screen.
2. **Text has to be resizable**. There will always be someone not comfortable with our text size → like a pair of shoes we need to offer a way to feel comfortable the user → BAD PRACTICE: use the browser settings to change the size of the text (computational effort).
3. **Text should be recognizable as a normal text**. Be aware of different fonts and anyway to diverse attributes to create graphical effects → **use only one font, maximum two**.
4. **Beware contrast**! A bad cost leads to more computational effort for the user.
5. **Be careful because what it looks nice may not be so good**.
6. **The upper-case text reads slower**: users take at minimum 10% more time. There are other ways to emphasize the text.
7. **Don't use images instead text**:
    a. Bad scalability.
    b. When the page size gets bigger → longer loading time.
    c. No paste&copy allowed.
    d. Bad interactions with search engines.

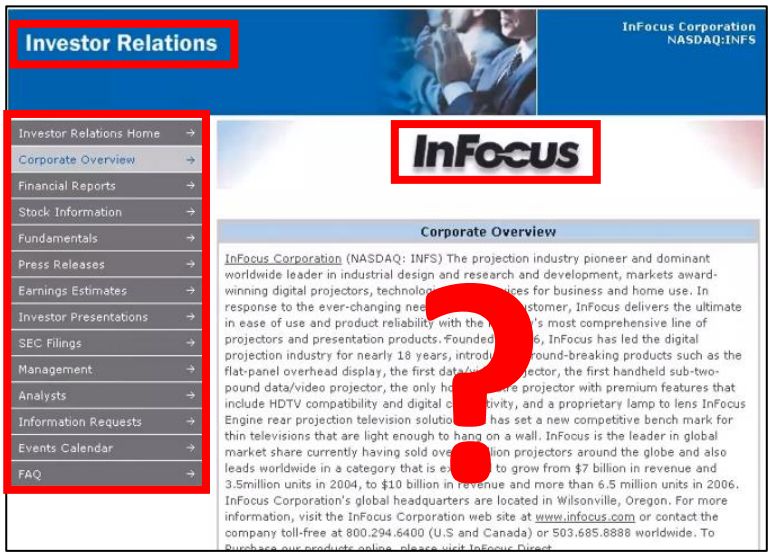
LOREM IPSUM PROBLEM: **usually web designers use this text to design a mock-up of webpages**. When we have an empty box of text, and we want to see how the layout changes we fill that box with lorem ipsum text and observe how the layout changes and eventually when the website is finished, we replace that text with the real one.

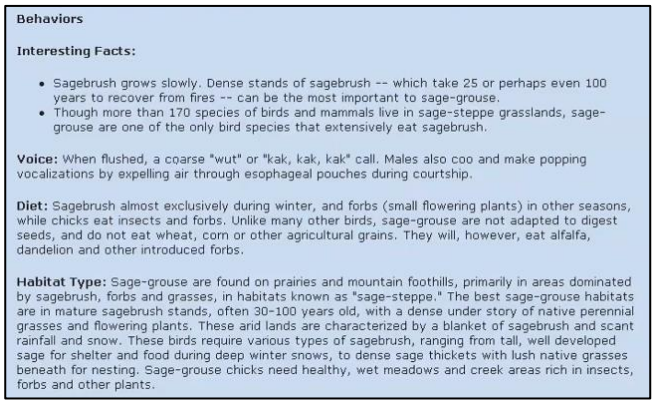
Text is seen as something dispatched from design, and typically his content is ignored. In reality however, users read, and not only…

When a user approach to a webpage he creates a fast scan of all the page, a mental map of information in that page. The scan happens necessarily in a continuous way because we use our eyes continuosly. So, a good design minimizes this scanning effort, keeping into account it

is done following this linear progression. When our eyes meet a big text box, they do not perform a good and fast scan, and so the fatigue grows. We need to give a good structure, in a wat that fast scanning can be successful. **The same quantity of text can be seen different, depends on the layout used!**

Try to create a mental map of this site. We can highlight different box that represents different main parts of the webpage, like title, menu, etc:





Better layout: our brain puts a flag on "highlighted" parts.

For instance, here, we have flags for each bold text → we can understand what a certain section contains before reading it.

With lorem ipsum all of this is lost! Lorem ipsum is related to blocks of text that must be placed in a page, completely ignoring the mental map of the user.

So, **the structure is important** and there are various ways to structure a webpage properly. We saw already the good practice to divide text into blocks (for the reading timers). Breaking into smaller blocks can also be done with lorem ipsum, but we can do it in a much better way.

Can be done, for instance, using descripting title (as we see in the previous example) → **in general, every relevant block of text should have a descriptive title**. As we see, this helps the scanning phase to create a categorization/subclassification.

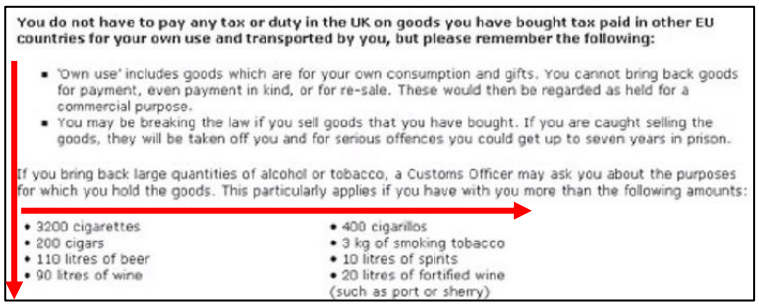
Similarly, we can use ***keywords*** inside blocks of text. Short and relevant, they make scanning fast and easy. Every block text is skipped, and only the keyword is kept. **Do not abuse in quantity!**

A particular case for keyword is ***links***. Links are noted even more than keywords → more important. But it is important to follow the same principles of keywords → they need to be short as the keyword otherwise we don't help the user with the scanning phase.

Additional problem: **the big mistake/shortcut of using the title of the page they link at → this is not the goal of a keyword**.

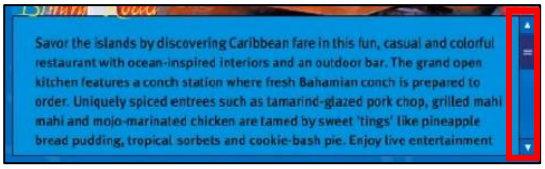
Other problems: using similar link all together; using the web address of the page; "click here to do something" links.

Another good way to structure text and to help the scanning phase is, for instance, to use ***itemized lists*** (ordered or not ordered) using simple html and CSS → we offer a structure to user → it splits complexity allowing to get information in an easier way. Typically, **we can use itemized lists when we have at least 4 elements** → when we have less than 4 elements and we use the itemized list we are increasing the complexity of a simple information. Itemized lists increases user satisfaction up to 47%.



When we put more than one list, their efficiency decreases linearly with the number of lists arranged vertically and exponentially with the number of lists arranged horizontally.

Note: having a horizontal list increases the computational cost and that is caused by the introduction of another dimension: the horizonal one → same principle of horizontal scroll bar.



An important rule with the text is the following: **never design the layout, producing mock-ups, separately from the text because this leads to most of the webpage text errors**.

3. E-commerce websites

This type of website has some peculiarities: being more specific sites we can be more precise on their guidelines. The most important things here are the **price** and the **product**. **Note that these two elements are put at the same level by the users.**

3.1 The price

Users want to know the price of a product in easy way. We need to pay attention to the timers and the rules on the mental map. So, where is the best place for the price? **The price has to stay near the product**. It could be a stupid answer, but we'll see that even if obvious for us it's not so easy for the web designers.

Many designers fall into the trap of **hyper-association**: they force the user to do other operations to know the price of a product. In the left-example the user needs to click on the product to know the price.

***Hyper-association* causes a loss of primary information,** and a user who wants to know more is forced to click "hoping" that things go well.

These clicks are called **gambling clicks**, because users do a kind of risky gamble → if user doesn't reach the reward, he wasted time, and this decreases drastically his satisfaction. The perfect instance is the "I'm feel lucky" button in google search → after we write some text, and we press this button the engine goes directly to one of the sites that it would be appears with a normal search.

One of the problems with prices is that **sometimes there is not a definite price at all**. For instance, in case of products that are not sold online but by other shops (online or physical). There are no justifications → **always give a price!**

When the exact price is not available, we can give an *approximate* price (best), or a *range* of prices (second best).

We must **pay attention on users' trust**. Remember all we have seen with the informative axes, especially the WHO one → all that is quickly ruined when some advertisements techniques.

CLASSIC ADVERTISEMENT

In this media, the user is contact with the advertisement message just for little time → he has to be impressed and hint. It can be done in several ways, one of them is attract the user with the price. There are at least a couple of classic tricks used in advertisement:

1. Using a **bait price**, which is different than the real price.
2. Using a **net price that doesn't correspond** to the "all-included", the final overall price.

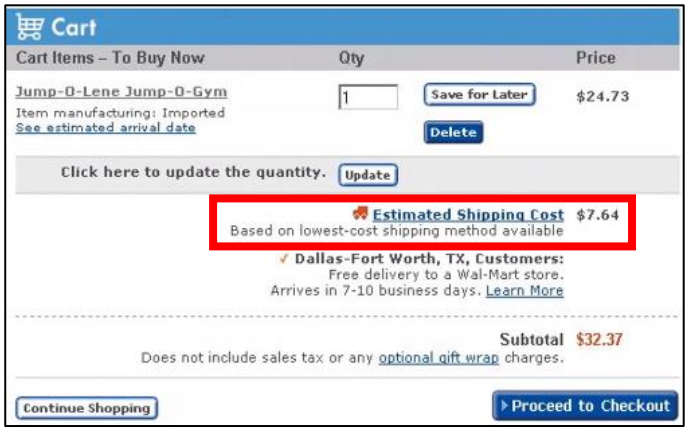
In both cases the price that is shown is different (lower) than the final price that the user will have to pay. The error is thinking that these rules can be applied even to the web, but **internet is a different place than TV**.

Our brain stores information in two areas: the ***short-term*** and the ***long-term (primary)*** memory → same principle with RAM/HD. When we have to memorize something, we don't put all in long-term memory, there will be too much information to remember (not always important). Most of the things that we listen, we see, goes to the short-term memory → we forget them and only a small part of them goes to the long-term memory.

In the physical word advertisement is needed to attract someone into the shop with pretty and capturing images. The primary recall (product and good price) goes to primary memory whereas the detail (exact price) tends to fade staying the short-term memory. For our brain is easy to process images than number → we can remember a beautiful image but for number is harder.

Inside a physical shop, we cannot use such tricks, because the short-term memory is effective → we see immediately the product price and take actions accordingly to what we collect. These advertisements techniques don't work on a website. The famous "starting from x euros", "average per night", etc are just tricks that destroy the users' trust. **90% of the users leave the website when they see these advertisements tricks, and the last 10% has a trust decrease for the site**.

The net price tricks are built by showing a price to the user that is not the final one. For instance, omitting the taxes or the transport cost. Solutions like "50€ + transportation" don't work in the web because the user doesn't see all of information even if "+ transportation" is present in the price string → user has to go to check-out to see the total (again hyper-association).



The final price is usually obtained by completing the transaction. This looks ok but is not an optimal solution! **We are forcing the users to do gambling clicks**.

BETTER SOLUTION: if the final price is not perfectly defined put a range with minimum and maximum of total.

3.2 The product

One of the most important things a ecommerce has to provide is, intuitively, **a good description of the product**. Not so obvious as it seems. Note: pay attention to the same problem of the hyper-association seen with the price → users don't have to do gambling clicks.

The big error is to assume the user already knows the product and comes to the site only to know the price. In fact, the user always expects a complete description of the production the site. These errors lead ecommerce websites to lose users → **incomplete descriptions bring users to go to other competitors' sites** (on average 99% of the users) and, moreover, it gives the impression the site itself is not professional.

It's interesting compare **price vs description**: if my price is lower will the user come back only due to that? REMEMBER: there is a cost to remember a price → it's just a number and it goes to the short-term memory!
On average lower price up to -20% than the competitors could bring back only 5% of the users. If the difference is enough (more than just 20%) user may remember the price but otherwise it's hard for them.

Beyond the description of the product features, it is also important the **visual description**. It is so crucial because if the user is interested, he wants to see the product at maximum level (even full screen). **When the user is much interested on the product, the timers turn off**: it's a user's choice to get more detail and he can comfortably wait for a more detailed image.

Note: don't force the detailed view → it's a user choice.

Best way: **offer various perspective views with high detail**, mimicking the guidelines seen on the 3D as 2D (example: images gallery). But **remember that visual description and textual description are complementary** → none of them can replace the other one.

4. Behavioural analysis of users

We already saw many characteristics of users' behaviour. Let's delve more in the analysis.

4.1 Newspapers analysis

In the 90s Poynter research institute started to apply new technologies to study the behaviour of the people on that time media: the newspapers. The goal was to analyse on what the user focus on the newspapers to **discover users' rules**.

From that analysis precise behavioural classes have emerged, so let's start to see users behave with the newspaper's media.

What are high attractions spots? **Photos** and **color**. The more the page is colored, the more it is perceived as rich in information → people think the information here is better.

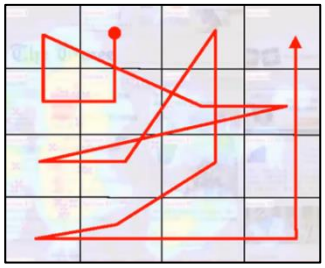
Who wins, in term of attention time by users, between text and images? **Images**, because are seen more than text → 80% to 20%. (Think about the success of comics → same idea). When we combine images and text, we do the hit: pieces of text mixed with an image are seen much more than without a companion image → **images amplify the power of the text**.

What is the entry point (first thing seen) in a newspaper page? The biggest image.

Another interesting thing is that when we open a newspaper the **two pages are perceived like one big screen** → a wider view (single entity of information). We like that type of view because we have a complete view of all the information and there aren't bad surprises on the left or on the right side because we are able to see everything.

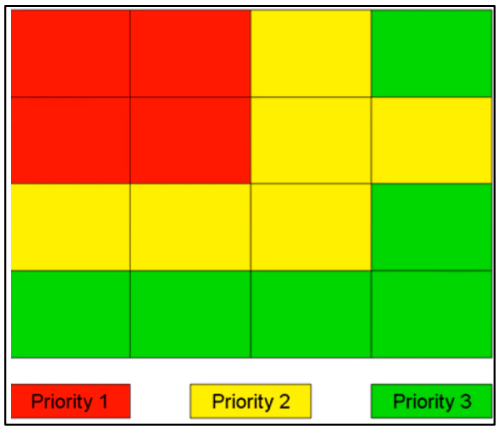
4.2 Web analysis

For much time, people just thought that the web would just follow the same rules. There are some common rules but as said, the **web is a different media where users behave differently**.



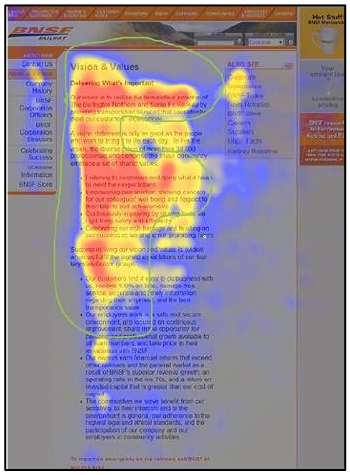
Example of a user webpage inspection → not comparable with the newspaper.

Note that with this "tracker" we can highlight the most visited webpage zones, the so called "**hot zones**" that create a sort of thermographic map of the webpage.



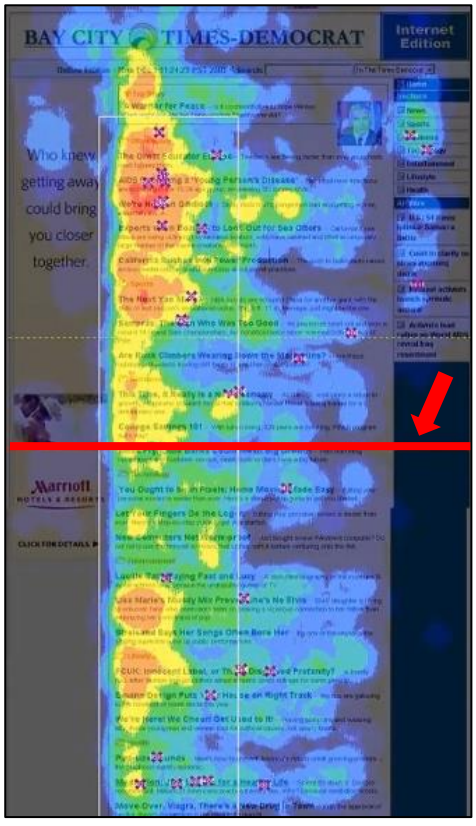
This is an abstract map that can show the users' level of attention in a general webpage using different priority colors. We can pick up this map and put in a webpage and see what happens → if there is a mismatch with this map the webpage is wrongly design.

As we can see there is a first difference with newspapers: **the entry point of a webpage is the top left corner** (for now it's just an approximation).



Let's see better the attention zones, also said **F-shape**, or **ice-cream cone**.

There is another curse, derived from the attention zones: how the user gets information from the webpage? Scrolling. We have already talked about it, but we have never seen how it is performed by users. The great majority **scrolls page by page** (and not line by line) → it is **faster and gives quicker access to new information**.



Whenever users scroll page by page, we have "a different webpage" to examine, a different F-shape zone, where we can get new information.

In the point where the user scrolls, a blind spot is created (in the figure the red line that divides the two "pages" after one user scroll) and here the text is not looked by the user. This scenario is called "**the blind spot effect**".

Unfortunately, this phenomenon changes along with screen size.

4.2.1 Text

Other aspect in webpages: **text is more important than images**. **The real attraction point, starting from the top-left corner, is text**. So, for instance, a logo without text inside or nearby the top-left corner can confuse users.

The best text information flow is one column which works better than more columns (remember the discussion on multiple "horizontal" itemized lists).

If we have more keywords within the paragraph there are effective ways to emphasize those words, helping the scanning phase:

- put on a single line (like a title);
- be bigger than other text;
- be classical hyperlinks (or underlined);
- at the beginning of the row.

+ more effective

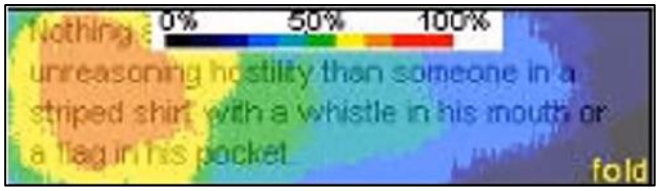
- less effective

Note: **bold** font is useful to emphasize keywords during phase 2 of scanning (normal reading), here we are talking about the phase 1 (webpage first visit).

Note: splitting the entire text in paragraphs a user may prefer one paragraph than other ones, so the solution of this problem is: use *shorter* paragraphs because they attract much more than *longer* ones. The same problem happens with titles: shorter titles attract much more than longer ones. **Having a text broken with more paragraphs provokes a relaxation of timers, encouraging reading**.

A title is an important attractor, but it can be further improved adding below it a short explanation/summary → the so called "blurb". **The blurb increases the hot areas in the F-shape zones → more points of information and attraction for the user**.

Blurbs do not change significantly users' will to go on with the navigation, but they cause a bigger return rate to the page (about +20%).



Blurb can be perceived in different ways, as it is not just seen as a single text block: like a webpage, it can be divided into various zones.

It is perceived asymmetrically: the left part is much more important → for instance, considering again an ecommerce, the price should be placed on the left because it is an information to emphasize.

The layout must be compact or not? A user who has just accessed a page prefers the wide, but then the compact wins in the scroll. So, a compromise must be found between the two, in fact:

- ➔ **Separation = faster scanning**, recommended in the pages for navigation, full of links. If there's too much separation this led to diluted design which would only bring annoyance.
- ➔ **Compact = better learning of the content**, generally the best one (also for the scroll).

4.2.2 Images

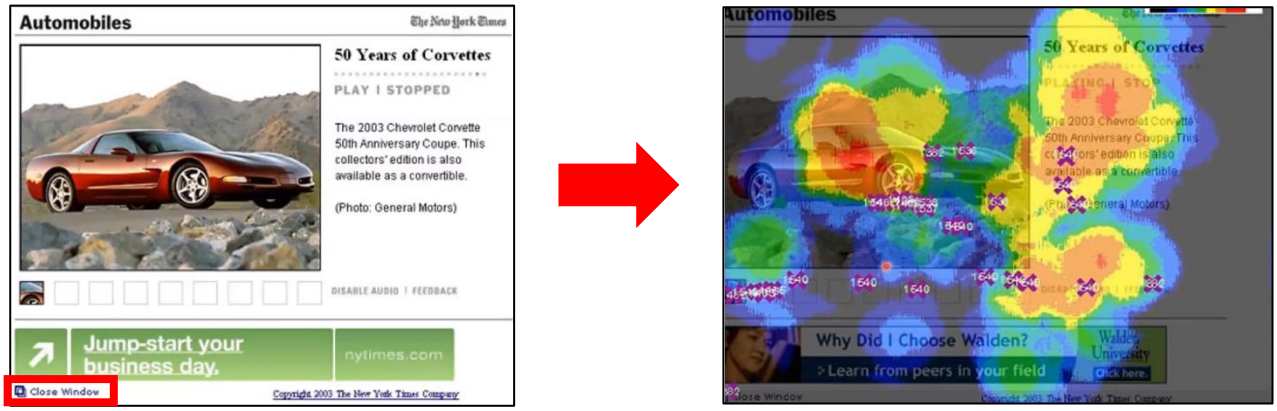
As said, images are not fundamental like text, but they are anyway important. For instance, remember the discussion on ecommerce websites (product images are much important in that scenario). **The minimum size for the images is 210x230 pixels** → below this size, images are too small, and they became icons, not images.

Even if images fall behind text for users' priority, they create a magnet effect. **Images attracts clicks** → we can see that in the thermographic map (look at x elements). If an image is placed near a link, they also should be clickable but **generally, all images should be clickable**, also if it isn't a link, maybe simply providing a zoom. In this way, users are happy because their clicks led to a reward: something happens instead of nothing!

<u>Example of a slideshow with only an image</u>

When we give this page to the user, his attention is to discover the other images but there are no other ones, so user is wasting many and many clicks. And, after wasting much time and clicks he just wants to get out of this site. He tries to find a way out where he's more attracted (hot zones) and he still wastes precious time since the "close window" button is out of the attention zones and behind an advertisement!

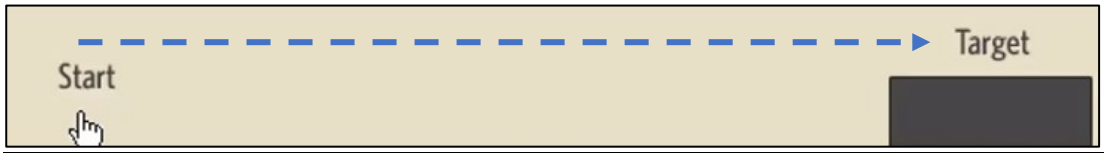
The moral of this example is that when we put the user in a certain situation, he has to have the control of it, avoid getting lost.



4.2.3 Time

One thing that is still missing in the thermographic map, is time. We talked about absolute times, but we didn't face the problem of its dynamic nature: **accessing parts of the page**. This is very important because it interacts with absolute timers.

The **general problem**: users have to move the mouse to reach something (e.g., a button).



FITTS LAW: how much time (T) the user takes to do something (e.g., reach and click a button).

$$T = a + b \log_2 \left(1 + \frac{D}{W}\right)$$

a = start/stop
b = co-speed ("slowness") } user factors

D = distance (from the target)
W = width (of the target, a button) } other factors

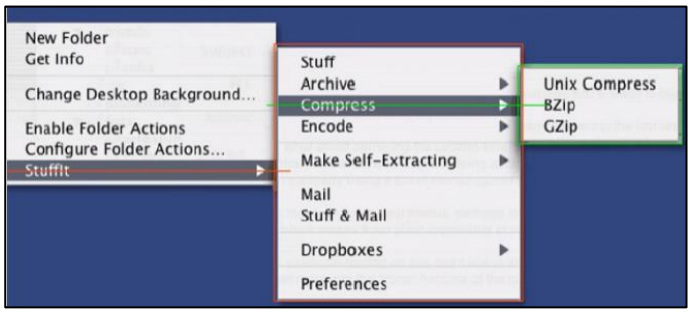
Note that **distance is not so important because the structure of a webpage is not a linear relation but a logarithmic one**. So, if the distance is double, user doesn't spend the double of the time but less more because when the target is far from us, we accelerate the speed of the pointer to reach it faster.

Example: ***Point-and-click*** or ***drag-and-drop***? Fitts law answers:

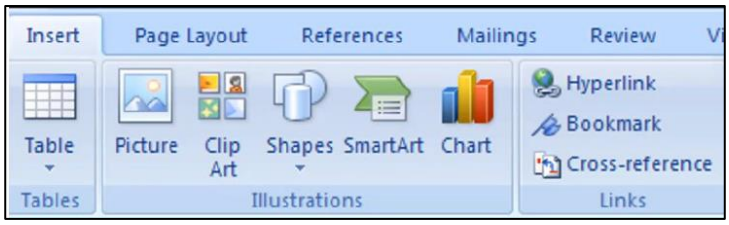
Drag-and-drop has a bigger co-speed b (→ slower speed), due to muscle fatigue: we have to keep clicked the object until we drop it. So, drag-and-drop should actually be avoided in web interfaces → we can offer it as an alternative, but it cannot replace point-and-click.

Implications of Fitts: bigger objects are much easier to click, and this minimize object distances. Break the implicit rules that objects should have the same size: **closer objects can be smaller, and objects far away can be made bigger**.

Now we can better understand, beyond what we said earlier, why people in general dislike menus. It is true a menu keeps commands close by minimizing the distance from each menu voices, but as seen with Fitts law, **elements proximity is not the only factor**.



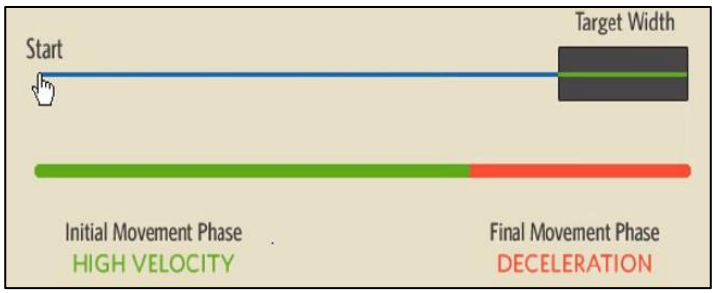
Example of Fitts-menu: balanced menus, much better than just a one long list of options. Also note that here the menu cannot be closed accidentally since all destination directions, even the faster ones, allow the user to stay inside the menu area.



Target Size Rule: size of buttons should be proportional to its usage frequency.

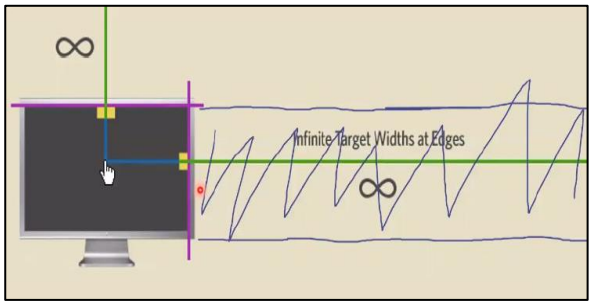
Example: the program *Word (.docx)* has different buttons sizes depending on the used frequency of the button.

Fitts law also depends on the media. For instance, on the touch screen the behaviour is different from the desktop alternative → users don't increase the speed of the pointer but just land over the target.



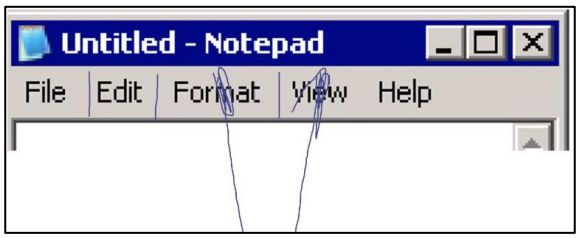
Note: pay attention on the deceleration when we reach the target, there has to be enough space to stop the pointer without overtaking the target. Here this doesn't work with the vertical direction → the target height is not big enough and so the space to stop is small.

A special case of Fitts law: **the border**

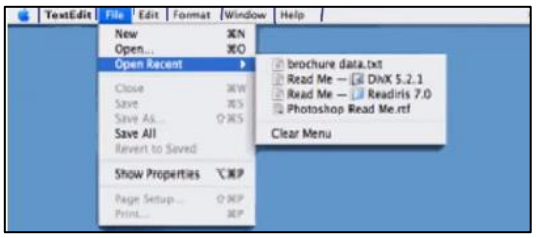


In the border we have enough space to land over the target. Borders are special types of buttons → **infinite size buttons** because we can go on with the pointer and this remains on the border (button). Same principle of the  icons on a desktop interface to manage opened windows.

So, **it is much easier to go to the opposite side of the screen, rather than clicking an object just a few pixels away**.

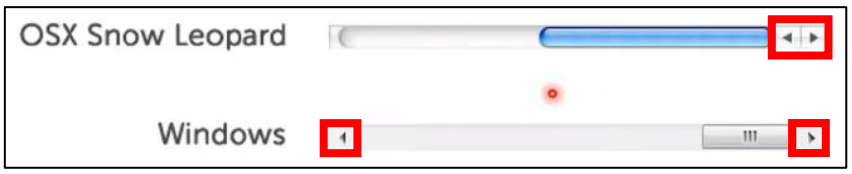


VS

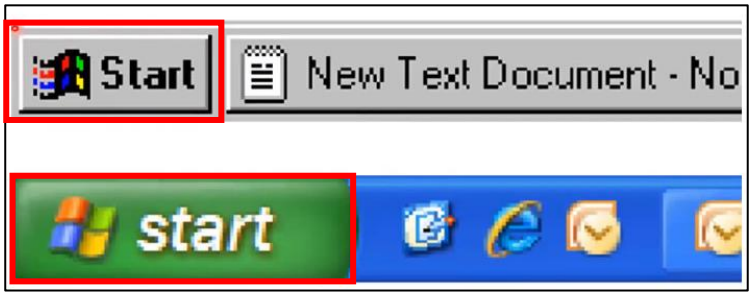


Note that in the first case (windows) we have much less space to stop the pointer than the second case (MacOS) that exploits the power of the infinity buttons mentioned before. Mac style menu are 5 times faster than windows ones → consequence: TASK BAR (bottom program bar of the PCs), which has been developed afterwards, following the Fitts law.

Another example of *Windows* failure and related to Fitts law:



The **corners** (**magic spots**) can be seen as the borders → same principle of infinity buttons but with more power. **In the corners there are the most useful options** (think about the windows environment → if we go to the left-down corner we match the windows button that is one of the most important).



A Windows epic fail just for a few pixels (first one): note that the first solution violets what we discussed before about the power of the corners because it has a specific side where user must click and not all the area compared to the second one.

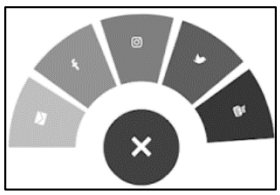
Anyway, staying to the web, **usage of windows can diminish the importance of magic spots** because if we have not full screen-sized windows, we lose the power of the corners.

Note that **not all corners are equal**, some corners are easier to reach than others. The ranking of corners for a right-handed, from best to worst, is the following:

1. Bottom-right
2. Top-left
3. Top-right
4. Bottom-left

There is yet another magical place: **the magic pixel** which creates pop-up that are useful and easier to reach from the user. When we use programs like Word, and we select a portion of text a small menu automatically appears with some options → that's an example of a magic pixel with zero distance!

Another derivation from Fitts law is the **pie menu**. These menus are opened clicking on the magic spot. The menu appears around where the user clicked on giving a faster way to choose the next action.



The **fan menu** is a hybrid between the pie menu and the magic spots, activated on/near borders, or on the corners where we don't have the space for an entire pie.

Pie menus vs linear menus: the **linear menus work better when there are many elements, or when their descriptions require much text**.

Note that we can combine these two types of menus obtaining nice and functional solutions. The menus were created to increase the usability (think about a videogame scenario where we have the time is the main constraint → we need to do things quickly and pie menu and its derivations become useful in this kind of scenario).

4.2.4 Advertisements

Advertising is important because it is part of the classic business model of web applications. It allows you to make a service for free, which leads to a higher the number of users and revenues. Advertising on our site usually comes from external sites.

For the moment we focus on advertisement within the site. Later we will also talk of the dual aspect: how to externally publicize our site/product/brand.

Fundamental fact: user hate advertisements! They click on them just for 0,4%.

We can power up advertisements with **good positioning** and more attractive ads. The best places to put them are:
1) the left column;
2) the top of the page;
3) the right column;
X) we can put advertisements nearby interesting content; they are seen more there.

Advertisements are **beautiful** and **attractive**; we need to keep in mind these characteristics when we use advertisements on the web. **There are things that users like and things that they hate**.

We start from the things that user don't like (*hate rate% ↓*):

10) Automatically plays and moves within a box (79%) [music or video]

9) Blinks (87%)

8) Occupies most of the page (90%)

7) Moves along the screen (92%)

6) Doesn't say what it is about (92%)

5) Covers something you were trying to read (93%)

4) Doesn't have a clear way to be taken away (93%)

3) Try to make your click over (94%)

2) Loads slowly (94%)

1) Pop-up (95%)

So, you have to **use other tricks to attract users**. In real life beautiful people or bright colors attract, but on the web? The images, as already mentioned, are of series B, due to the **zapping effect**: one thinks already seeing an image that it is an advertisement and discards it. You can avoid the zapping effect by proposing ordinary people who behave strangely, or who despise the product itself! This power up images which increase the curiosity of users and then it's done. **The key is confounding users!**

Another related strategy is to **try to not activate the user zapping by confounding him → insert an advertisement in the middle of text of interest** BUT if the image has a border, we are creating separation between the content and the ads, thus activating the zapping effect (bad example of usage).

A positive example is the ***Blending effect***, normal content is mixed with advertising the training set is confused as we can see from the above example (no borders → no user flag for an advertisement to skip → user doesn't discard it).

Note: don't forget about the power of the text: **images without text are nothing!**

For instance, **Google uses *Blending effect* by inserting advertising messages in each search result instead of the first three results**, separated from the real results by an almost invisible light-yellow border (mandatory by law).

Note: web games on our site are a good way to attract users and have a high effectiveness even if trivial. We are talking about games that can be played on our site, maybe to win something (e.g., a coupon) → for instance "*spin the wheel*" game.

Other effects to attract attention:

- If there is someone in the image looking at something, users focus on that thing.

- In images with people displayed in full, the focus is more on the private parts.

Care must be taken **not to insert advertising that is too disconnected to the website content** → they create the ***distraction effect***: the timers go down by -40%, it creates frustration and the desire to return to the site goes down by -80%!

For this reason, there is an increasing attempt to use not generalist advertising but **Behavioral Advertising**, that is **advertising adapts to the content and to the user who views it**, tracing a profile, then recording the interests and hobbies. This is possible because users can be diversified on the web. This type of advertising has an effectiveness up to 100 times higher than normal and an advantage factor of at least 10 times! Furthermore, this technique has no negative effects on the timers and increases the desire to return by 100 times!

4.2.5 Research

Abondance of information can create a big problem: **find the right information**.

The website size is an important factor. Beyond the 100 pages, it is necessary to offer some search tool and beyond 1000, it is essential to offer a good search tool. If search is available, users tend to use it around 99-100%.

Remember the **problem of deep linking**: users may arrive to a website without passing through the homepage and when this happens, on average 60% of users uses the search functionality (if present of course).

SOLUTION 1 (trivial and not so effective): **simulating a search engine**. For example, simulating the Google tools, locally for the site, it's a no-cost solution. We can make this research on google engine "*shoes site:zappos.com*" and we can "paste" the google behaviour on our website (zappos) to show to the user our shoes (we can do that with just one line of code). This sounds great!

However, **this kind of search does not work well**, because it is not designed to work at a low scale, if the user already arrived on the site using a search engine and he didn't find what he needed, it means the search engine itself was not very precise with site information and so it won't work well inside the site.

Another negative point is that **search engines cut off 50% of the indexing** and therefore it will also be done locally with loss of half of the information. The site map is useless for indexing.

SOLUTION 2: **create a local search**, but paying attention to respect the habits of an average user, being used to search engines, it must have a very similar interface, with **a text box and a search button**. The button must have the precise name "search" to don't confuse the user and must be placed on the right (the button on the left creates a delay of 2 seconds).

Note: **less is more**. When we offer a search functionality, we must put just a textbox with a search button on the right and nothing different/else. If we put other/different things from the

standard user timers increase since there's the need to understand how to use this new "interface", and so also the computational effort increases.

CONSTRAINED SEARCH

Like a classical one (textual) but we can add some constraints (we search for shoes, but we want a certain type of shoes of a certain brand). It is efficient and users like it, but care must be taken whether to make it ***static*** (enter the data and then click on search) or ***dynamic*** (each selection of a filter displays the results → even if we didn't apply all the constraints we wanted).

We need to pay a lot of attention when we offer dynamic search because there isn't a standard from which we can build it.

In **static search** the user applies constraints and works only on explicit user action (the pressure of the searching button).

There is **dynamic search** (without pressing the search button) instead it takes a few seconds each time to display the results. **It is worth using it when there are few constraints so as not to waste time for the user**.

A good way to offer search: offer the classical one and also the constrained one.

Many users think that static search is dynamic, they select the search and wait for the results, then they realize that they had to click on the search button, creating frustration.

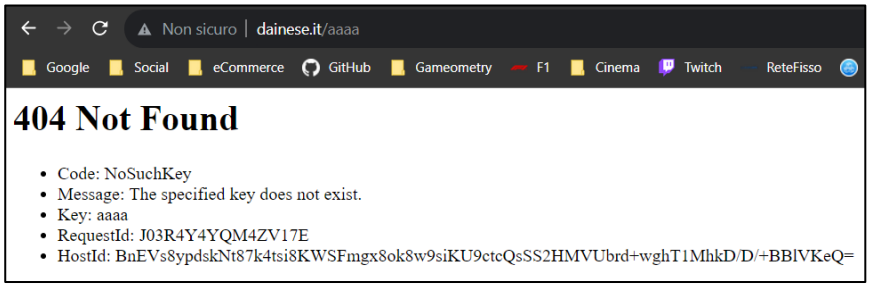
Static with many constraints is used (otherwise user has to do a lot of automatic searches to find the final result), while it is better to use dynamics with few constraints. A union of the two is the **hybrid search**, a static search with automatic launch when all parameters have been filled → not recommended, user doesn't expect this behavior.

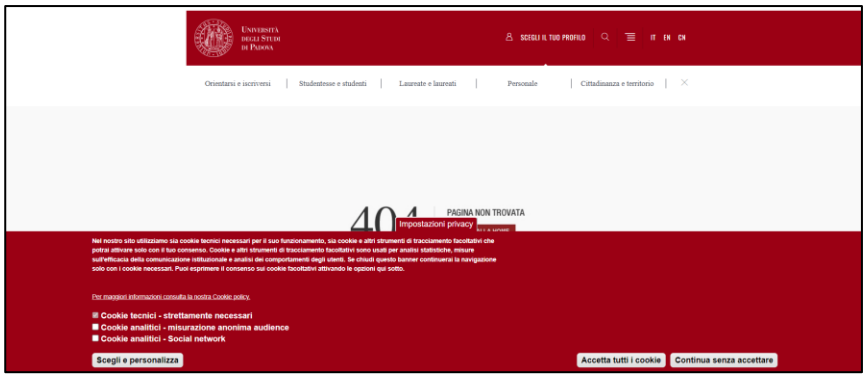
OUTPUT OF SEARCH

The best way is to follow the users and their habits and so, to behave like the big search engines. In the search results it is very useful to allow the ordering on all results and in both directions (price from + high to + low and vice versa). If the result contains 0 data, the best solution is to display a no result message instead of displaying nothing.

A special case of zero results is the **404 error**. The 404-error page **should be easy and understandable** from a common user so, don't use hard terminology: be clear on the fact that the searched page doesn't exist and put a link to working pages, like the homepage to link the user to something that exists and related to what he was searching for.

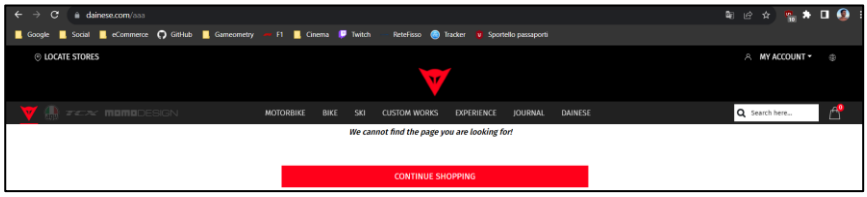
Bad examples (Italian version of Dainese and unipd):



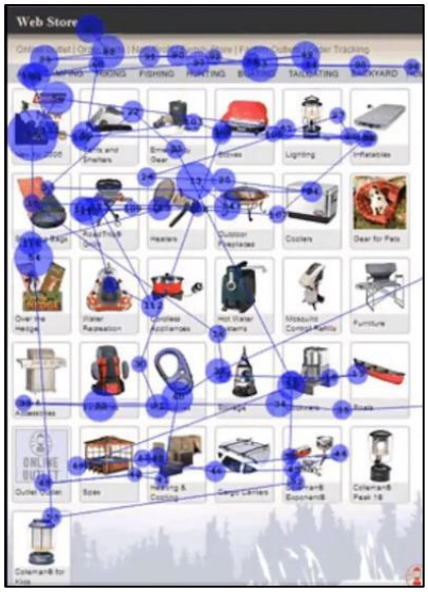


Here the main information is covered by cookies!?

A good example (global version of Dainese):



Note that here we are also offering the website menu helping the user to find what he was searching for.



Other search results layout: **grid view of products** is more compact and displays more products on the same page than a list view.

The problem is the search for a product: the time to find it is very variable → we can reach the product in different ways.

A ***dynamic explosion*** occurs in the brain, due to the non-linear path that we can follow. In fact, the bi-dimensional space tends to confuse users.

For this reason, search engines use a list view of results while the grid layout creates a great waste of time in micro-navigation.

INPUT OF SEARCH

At the beginning of the web, when users used a search box, in most cases they entered single keywords but with the passage of time users use more and more words. **The recommended average length for a search box is 30 characters**, this measure satisfies 90% of users.

- Google: 44 characters (almost 100% of user satisfaction);
- Bing: 57 characters → they increase the size recently, due to a collaboration with ChatGPT: here people shape bigger requests (almost 100% of user satisfaction).

If an input box is too small, scroll occurs. This creates stress (part of the text disappears and user must remember what he has written in the hidden part) → +1% for each scroll character. Furthermore, a small box induces the user to create shorter queries and therefore less specific, leading to less precision and poorer results. You can also use a ***hybrid view***, when the box is not used you can view it in a small fashion and once you clicked on it in a larger format (Volunia).

4.2.6 Research 2.0

Consider ChatGPT: how can we use it on a website? A new release allows to customize GPT creating a **custom agent** for a specific business and it's possible teaching it what it should say, defining its behaviour for any possible user search. A good, and similar but different, alternative is the following.

VOICE XML

We saw users prefer the box "search engine like" but there are other interaction ways which offer something different: web doesn't mean only visual but also **audio**. The web seen as a visual medium is very limiting, the voice is becoming a medium.

VoiceXML was born to go from the screen to the phone, the voice is translated into text and user always interacts with only text directly with the computer.

VoiceXML uses various technologies:

- *Speech Synthesis Markup Language (SSML)*, passes from enriched text to voice, then gives the possibility to give accents and all tonal enrichments to the computer voice, using special characters.
- *Pronunciation Lexicon Specification (PLS)*, specifies word pronunciation.
- *Call Control XML (CCXML)*, handling a voice interaction.
- *State Chart XML (SCXML)* is an evolution of CCXML, it is more general.
- *Speech Recognition Grammar (SRGS)* defines the action grammar, the context in which it is speaking.
- *Semantic Interpretation Speech Recognition (SISR)*, gives a meaning to what the user said. After applying SRGS, I interpret the resulting text.

VoiceXML example menu

```
<vxml version="2.0">
<menu> <prompt> Say one of: <enumerate/>
</prompt>
<choice
next="http://www.sports.example/start.vxml"
> Sports </choice>
<choice
next="http://www.weather.example/intro.vxml
"> Weather </choice>
<choice
next="http://www.news.example/news.vxml">
News </choice> <noinput>Please say one of
<enumerate/>
</noinput> </menu> </vxml>
```

We can provide a list of possible responses from a user question. The possible responses are limited to the specified option, so to the relative websites that can provide the searched information.

Interaction with Javascript to decide how to interpret and do actions: Javascript is the "action" language of VoiceXML.

INTERFACES

Be aware of the possible interfaces that can represent a voice interaction, which can lead to a user unsatisfaction by -42%. Using human assistant interfaces is dangerous because users expect a real common conversation, and they may think they can use a natural language. When this doesn't work, users' frustration increases because their expectations are broken.

Consequently, a good alternative is the currently used in digital assistants like Siri, Google and so on → these technologies give the user the opportunity to see what he wants to see, maybe a robot?

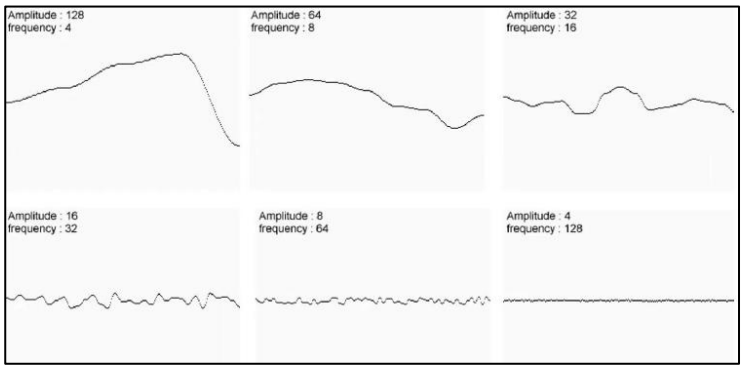
In this way **we are not creating expectations and this works!**

For instance, IKEA stayed with a "cartoon" picture, without going to photorealism and called "ANNA". Only textual inputs are accepted, and it was an **optional** assistant, so it is a user choice to use it. And for these reasons users actually appreciated.

A good interface must give to the user some hints, a direction to follow. In this way the user satisfaction increases, he knows what he has to do. With ChatGPT, users can write everything what they want without direction but this approach doesn't work in a normal website.

INTERFACES 2.0

It has also been noted that making the characters move makes unwanted movements, called fractal noises, more pleasant by an order of magnitude**. The noise is in fact the key**, a curve is created that starts with a large amplitude and little vitality and all pieces are created continuing to increase the vitality and decreasing the amplitude. Eventually they overlap and create **an effect of anything very real**, for example waves or ground or the movement of grass. These techniques make everything more real!



The noise component creates the feeling of verisimilitude. It has a fractal behaviour and if the noise is out of phase, it gives us a strange feeling. An interface made with hand lines, full of noise, is warmer to the user than a classic one with straight lines.

The wearables for now they don't work, they interfere with our reality. The Google glass is too invasive and constitutes a stressful input. The output as a popup is negative and annoying, and the physical media is heavy on both physical and psychological level. The smartwatches have greater potential than Google glass.

5. Search Engines

The success of a site is to be visible outside. Typically, via search engines. It is critical to be ranked highly from a search engine in the **SERP** (Search Engine Results Page).

How much is it important to appear in the top ten? And, within the top ten, how much the ranking is important? **The top ten absorbs more than 95% of all clicks**, in detail:

1. position takes 51% of clicks, therefore more than all the other 9 on the page;
2. position takes 16% of clicks;
3. position takes 6% of clicks;
4. position takes 6% of clicks;
5. position takes 5% of clicks;
6. position takes 4% of clicks;
7. position takes 2% of clicks;
8. position takes 1% of clicks;
9. position takes 1% of clicks;
10. position "**Black Jersey effect**" (in Italian "Malabrocca effect"); in the last place of the top ten the click rate doubles! In fact the clicks are at 2%, the visibility is higher than in the two previous spots.

An essential feature for a site is to be seen outside. Typically, users come to a site via search engines. It is therefore necessary to be as high as possible in the SERP to promote visibility.

In the past some search engines complicated the result interface adding images to attract people.

So, a question could be: "does mixing images and text in the results create differences in reading percentages?" No, when the user enters the section dedicated to images, the percentages stall, once he returns to the results page he picks up where he left off (images became a separate world from the textual result in search engines).

5.1 Spam Index (SPAMDEX)

Also called SEO (Search Engine Optimization) or SEP (Search Engine Persuasion). **It calculates the score of a page, that is the score the search engine assigns to position it in the SERP**. It's interesting to see how it is calculated because, getting the score, we can understand where we can power up our website to get a higher position.

Spam index consists of a **textual component** (written + code) and a **hypertextual component** (PageRank). Let's focus on textual component.

TFIDF (Term Frequency Inverse Document Frequency)

Gives a measure of how important a word is for the page → **TFIDF = TF * IDF**.

TF

Example: given a web page made up of 100 words, "football" is contained 5 times in it, so TF = 5%.

Giving a more formal definition, it **is the percentage of occurrences of a word on a page**. The main problem using only TF is that with common words like, for example an article ("the"), TF is very high.

IDF

It is the **inverse of the frequency of the word within the set of documents on a logarithmic scale**.

Example: given a web site of 1000 words, the article "the" appears 980 times, so 98% frequency, IDF = log(1 / 0.98) = **0.008**. This means that "the" is not an important word so we can scale down its importance. More the simple frequency (TF) is higher less the word is important.

Example: given a web site of 1000 words, the word "bike" appears in 100 pages → 10% frequency (0.1) → IDF = log(1 / .1) = **1**

Example: case of strange words that don't appear frequently so they are more important: given a web site of 1000 words, the word "Schopenauer" appears in 10 pages → 1% frequency (0.01) → IDF = log(1 / 0.01) = **10**

By giving a simplified definition, **a search engine takes all the pages containing the searched word and calculates the TFIDF of each** (usually already pre-calculated). If the query has more than one word the search engine just computes the sum of each word's TFIDF.

Note: if we want to raise the textual score of a page for a word, we have to be careful because raising too much its TFIDF automatically causes lowering the TFIDF of the other words.

Best solution: we cannot power up a specific page increasing all of its words' TFIDF, so we need to focus on a specific set of words (the "champions") and raise their TFIFD, lowering the others.

5.2 Increase positioning

To increase the ranking, one must try to increase the TFIDF of the site's keywords. There are various techniques related to the *spamming set*:

- **Body spam**: insert words in the body of the HTML page. It is effective but disadvantages the reading for the user: we cannot repeat many times a word because this led to reading problems.
- **Title spam**: keywords are inserted into the page titles (title of opened windows in the browser, outside the site), the website content is changed much less. Browsers rewards this technique.
- **Meta tag spam** (<*meta name = ”keywords” content = ”bike, football, etc.”*>): the main advantage is that the content is not modified, but in today's search engines it creates a disadvantage because this method has been abused over the time, so it doesn't affect pretty much the score.
- **Anchor text spam**: insert keywords in the text of the links, the score assigned compared to the common words is higher. Moreover, even the landing page receives benefits in terms of score.
- **URL spam**: insert keywords in the address of the page itself, they receive more bonuses over common words in the text, like the previous technique.

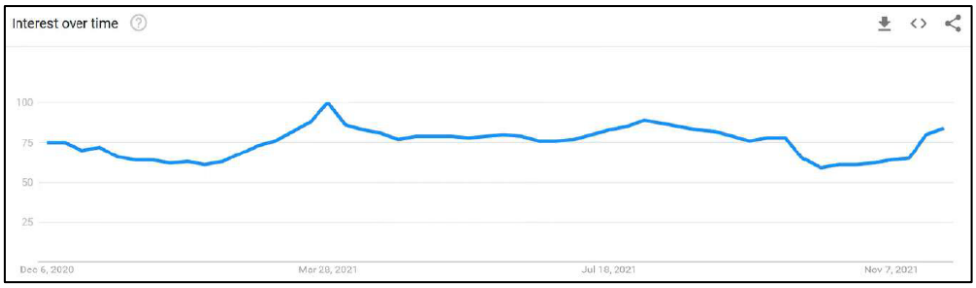
It is also possible to insert keywords also in the content of the page, following some insertion techniques:

- **Repetition**: the word is repeated several times, paying attention to the TFIDF, which if too repetitive creates damage. This technique is easy to spot by search engines which penalize this type of behaviour. It was a good technique at the beginning of the web but, in our days, we cannot repeat a word more than two times, otherwise search engines penalize the website.
- **Dumping**: inclusion of uncommon terms (also unrelated to the page content) that are little used → they will be present in very few places and therefore the score will be high!
- **Weaving**: copying the text of other webpages, modifying it by inserting keywords in random positions, creating more interesting content.
- **Stitching**: paste & copy pieces from different web sources and assemble them together with the site content → easy way to create interesting content to populate the site. The difference between this and *weaving* is that *weaving* consists in paste & copy entire pages while *stitching* keeps just the relevant information to build new website pages.
- **Broadening**: insert not only keywords but also synonyms, similar words and related phrases. Not only to cover user requests, but **search engines use the keyword similarity technique, so words of the same context give additional bonus**. To determine if two words are in the same context, they use the matching percentage of the words in the other sites.

NOTE: for *weaving* and *stitching* one can think that Google can note that the content of other website has been copied but it is not so easy because in both cases we are modifying the original content: in one case we are inserting keywords (so the content is different from the original one), while in the other case we are mixing different pieces of different pages.

Beyond the techniques of textual SPAMDEX, there is also another problem: **what keywords to choose!** We need to know what users want and to know that there is a couple of techniques:

1. **Google Ads Keyword Planner**: automatic keyword selection given a site [to pay].
2. **keywordtool.io**: google self-completion [free].
3. **trends.google.com**: given a keyword it computes its trend to discover how much that keyword is searched.



The problem of *term spamming* is that generally changes the content of the pages, so users are affected. We need to get a method to use these keywords without ruining our pages → we need some way to hide the keywords letting them visible only by the search engines:

- **Content hiding**: just make the background and the text color the same (white over white). Or making the text box so tiny that is invisible.
- **Redirection (302 technique)**: *<meta http-equiv = "refresh" content = "0;url=pippo.html">*, just reload the original page (only for the search engines) with another one (0 is the timer after that the page is reloaded). Actually, not so effective because search engines can avoid this page if they see that command. More effective: use javascript to refresh the page → search engines have a hard time to understand Javascript and in many cases they ignore it. *<script language="javascript"> location.replace("pippo.html) </script>*.
- **Cloaking**: search engines (for instance, Google bot) must declare when they inspect a website and we can exploit this fact offering to search engines pages that are different from ones visible by users. These different pages are built to increase the TFIDF of certain words and, since they can really affect the users experience (reading problems), we'll offer to users a different page under the same address. In this way we have webpages with higher ranking, avoiding causing reading problems for users.

We have seen the SPAMDEX textual part and now we pass to the **hypertextual** one. The textual part represents the local measure while the hypertextual part represents the global measure: which is the context of our page, where it is.

5.3 PageRank

The main idea is that **more links a page have more visibility the page gets**. But not only this:

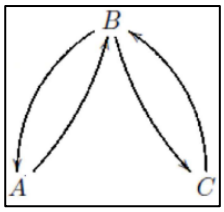
$$\pi_v = \sum_{(w,v)\in E} \frac{\pi_w}{d_w}$$

$$\sum_v \pi_v = 1$$

π_w = number of inlinks / d_w = number of outgoing links

The score (π_v) computed is proportional to the number of links presents in a page, otherwise websites can exaggerate with the number of links. We don't focus on the specific calculus because the search engines do that for us, what matters is the idea behind it.

This PageRank calculation was **easy to manipulate**, so they thought of a reformulation, it was decided to apply the Markov chains and random walks. With this reformulation, the PageRank measures the number of the times that a random surfer (random user) arrives to the relative site.



Using the random surfer technique, we can see that the amount of flow that goes to B is the double compared to the flow that goes to A or C.

There are some problems that can affect the PageRank measure:

- **Spider traps**, that is the infinite navigation in a spider site, a sort of calendar that continues to access the site day after day. This problem happens often.
- **Islands**: when there are different parts of the web that they aren't connected each other.

The solution to this problem is: **teleportation**. It is possible to fix the PageRank patching the original formula:

$$\pi_v = (1-\epsilon)\left(\sum_{(w,v)\in E} \frac{\pi_w}{d_w}\right) + \frac{\epsilon}{N}$$

The original formula + **teleportation factor** which consists in considering the normal navigation and the random navigation to get a more realistic calculus.

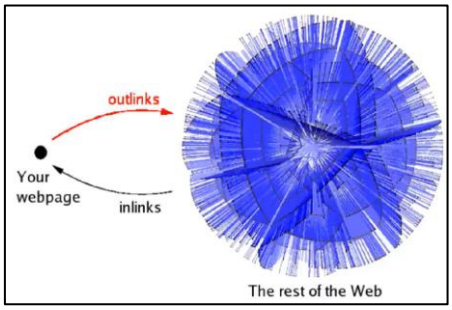
$$\mathbf{T} = \int_0^1 \mathbf{r}(\alpha) d\alpha$$

Another elegant solution: **Total PageRank**. It calculates an average sum of all possible choices. Teleportation = 0 to Teleportation = 1.

There always remains the problem of **dead ends**: a page without outgoing links, the solution adopted is the penalty! It was decided to heavily penalize the pages without outgoing links.

We have seen how PageRank works and now we can see how use it for SPAMDEX.

Two fundamental factors, **inlinks** and **outlinks**:

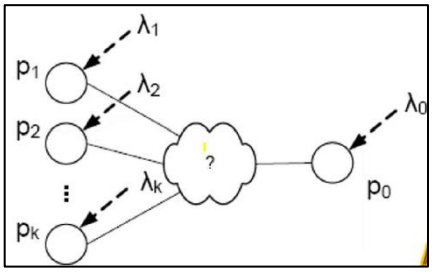


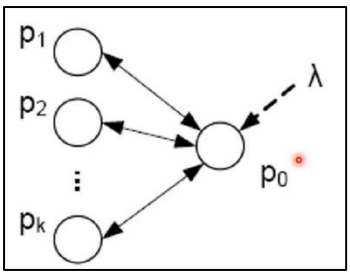
1) **Using inlinks** we try to get from the rest of the web links to our page. The PageRank is always higher due to the high flow that we have in the site. We can consider different techniques to increase the number of inlinks:

- **Infiltration**: infiltrate in other website and try to insert links to our site. For instance, inside blogs, comment section, etc.
- **Honey pot**: create "yummy" content and then naturally receive inlinks. There are legal and illegal ways to do it: we can do it by smart paste & copy from content of other sites.
- **Link exchange**: exchange of links by agreeing with another site.
- **Resurrection**: when a website "dies" we can buy its domain and redirect its flow on our websites putting into it links that redirect users to our site.

2) **Using outlinks** we are giving some entry points to other sites. For symmetry reason, the PageRank decreases because some flow gets out from us. But here there are some interesting properties used by the search engines:

- **Solidity**: adding outlinks to a page apparently doesn't cause an increase of the hypertextual score but if we know what to do, it can increase the PageRank. The hypertextual contribution given by PageRank eventually was not so intuitive as it seemed.
- **Spam farm**: a special link structure (more pages) devoted to increase the hypertextual score → more pages with different links that try to power up our page adding outlinks.





⇨ Ideal structure

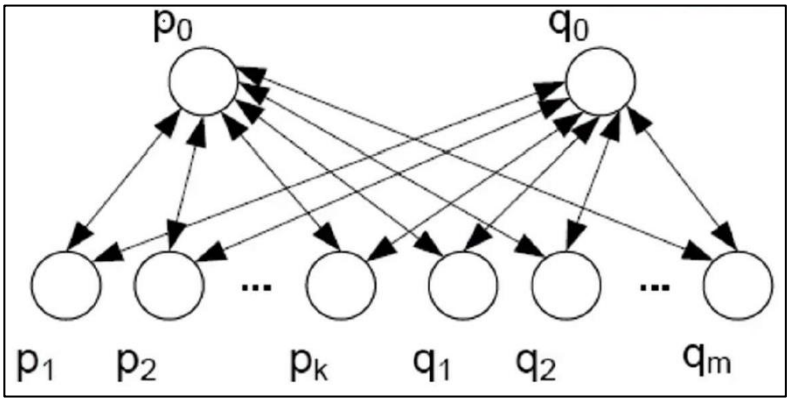
We are collecting the flow from other pages (under our control) via a special structure. This technique has good properties because **it uses the least possible number of links, while still keeps reachability**. Every page in the ideal structure is reachable from the search engine: if a search engine inspects the p0 page, it can also reach the other

connected pages thanks to the bidirectional connection and these other pages gives their contribution to the final score.

5.4 Web alliances

There is also the possibility to join forces with other websites that are not our competitors with an alliance. The idea: join our spam farm with someone else's. There are different ways to do it.

DEEP ALLIANCE

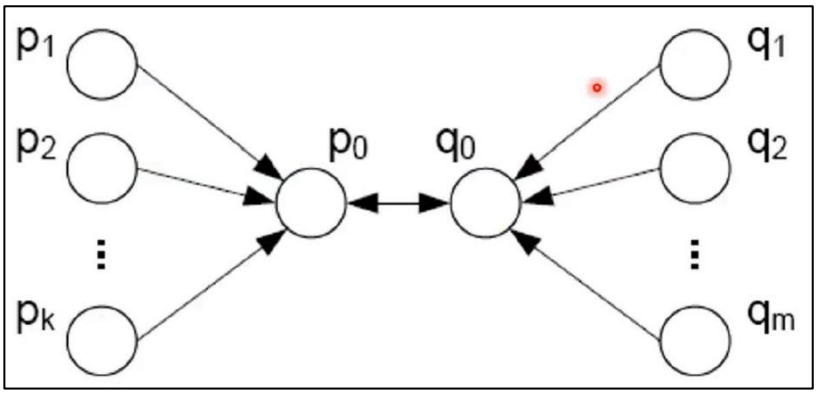


p0 is our website and q0 is the allied site. We can recognize the part under our control (p_i) and the part under the allied control (q_i).

In this scenario we provide links not just to our page but also to the ally's page and vice versa.

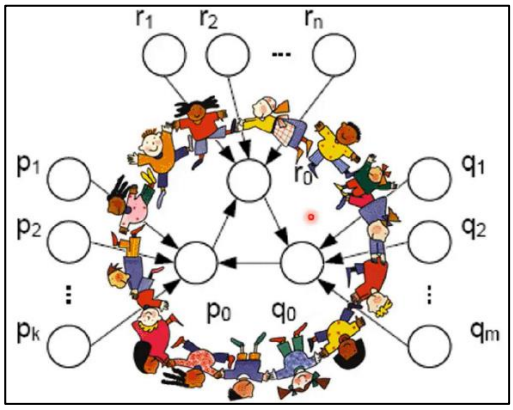
With this technique we can reach the average of the two PageRank → we have a more stable structure: we do our best and then we share goods and bads (= share of responsibilities).

SUPERFICIAL ALLIANCE



This scenario is the opposite of the previous one: we use the minimum number of links to join forces with another website. The PageRank becomes bigger than the maximum between the two websites.

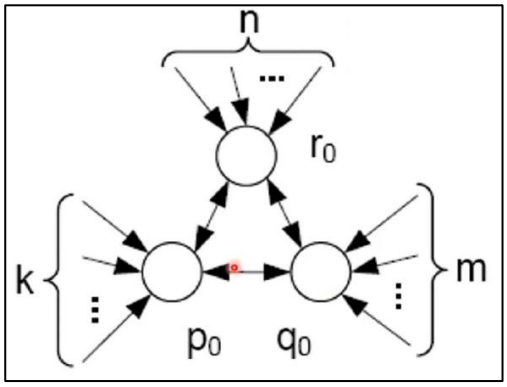
RING STRUCTURE



We are creating a sort of ring around the top pages that links other top pages.

Note: this structure increases the PageRank but if one of the internal links falls, many connection problems are created and therefore PageRank balancing problems.

COMPLETE CORE



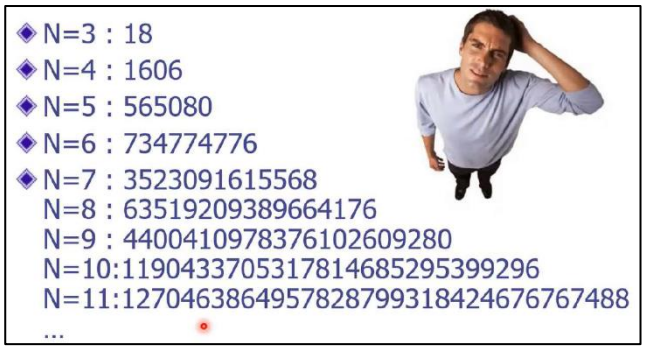
We can shape the ring in another way: every top page is connected with other top pages.

5.4.1 Search engines countermeasures

An important thing to know is that **search engines try to counterbalance our effort**: identify attempts to create spam factories and when these attempts are found out, search engines punish the relative sites.

In reality, *ring* and *complete core* are not the only two optimal structures. There are other structures that give the same result. We just need to have a **strongly connected graph among the target pages** → from every page I can arrive to any other.

So, we just have to create these connections and hope that Google doesn't see them? How many strongly connected graphs there can be? If they are only a few ones, then the countermeasures will work well but if the number N increases search engines can have some problems to identify the structure.

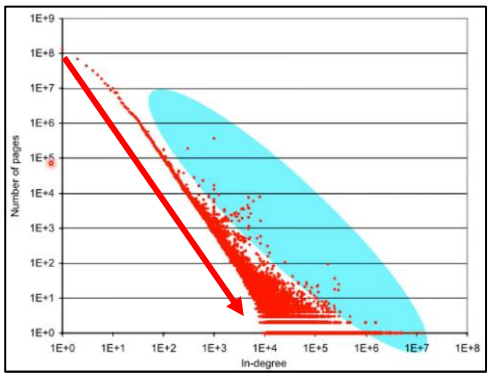


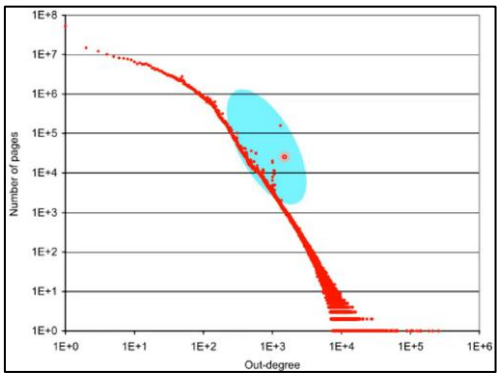
We also have to highlight the fact that this type of inspection has to be done among all the web and not just on our site → the computational cost really increases for the search engines with a high N.

It was therefore necessary to carry out checks in a different way.

FIRST COUNTERMEASURE: the use of the two types of PageRank → the original one and the one with teleportation, both are calculated, and their difference is called **relative spam mass**. If the spam mass is very high, the site is suspicious and therefore penalized. **The success rate with this technique is 95% -100% in finding spam**.

SECOND COUNTERMEASURE: we started from the statement that the **web has a shape**, we then take the website and analyse its shape: if it differs a lot from the "standard" form then it means that the website uses shortcuts, and it's penalized! In addition, incoming and outgoing links are analysed and all links that deviate from the average are penalized.

1) 

2) 

1) We can plot how many inlinks point to a website, compared to its size. The number of links of a website is directly proportional to the size of the website. In fact, we can note a line that represents the general behaviour and all the sites outside this line are suspicious.

2) We can also plot how many outlinks a website has, compared to its size. Even here we can note a common behaviour, a curve, and all the sites far from this curve are suspicious.

6. Names

Let's talk about names on the web in a *social* and *technical* side.

SOCIAL SIDE

When a website has to be created, the first thing to think about, before the structure, it's the name. So how can we choose a good web address?

On average the name of a site increases, on average, **user satisfaction by +10-20%** with a variance up to 50%. So, we need to care about it and to do that, there are some rules that can maximize the potential success of the site:

1. **Shorter names work much better** than longer names. A short name is easier to memorize, users remember about that!
2. **The name should be unique**, sufficiently separatable from other names. In fact, it should not be confused with other names!

   NOTE: a very bad choose is to choose a plural name where the singular one has already taken from others.
3. **Take the .COM**, which is the top-level domain that a site can use. Only this rule power up the user satisfaction by +4.5%.
4. **It should be easy to memorize and write**.
5. **Better to choose existing words rather than creating new ones** (overlapping with rule n.4). In any case, if new words, or acronyms, are used, the important factor is the ratio between the existing words and the new ones: if it is 1:1 it's ok (ex. *Facebook*). This rule leads to 1.5% or -5% of user satisfaction.
6. **The name must sound well**. We have other specific rules for the sound:
   - Name that starts with a vowel work well (+3.7%).
   - Semi-wovel (r, j, y, w) work well (+2.9%).
   - Consonants like f, v, s, z work even better (+3.3%).
   - Consonants p, k, t work better than the remaining ones (+1.9%).
   - Other letters works badly than the previous ones.
   - Sounds associated with bad words in English (like "uh", "yuck") damage the site up to a -44%. So, beware of slang expression that can be cool in real context but not in the web.
7. **No dashes (-)**. The impact usage is -3% of user satisfaction.
8. **Use numbers**. Typically, on the web we can find different sources that advice not to use numbers because people don't like numbers. But in reality, using numbers increase the user satisfaction +8.2%, think about a combination of letters and numbers.
9. **Be careful on how you check whether a domain is free or not!** There are several ways to do that but always use friend privacy policy alternatives because free checker can steal your name. When you have a name, just buy it until it's still available!

TECHNICAL SIDE

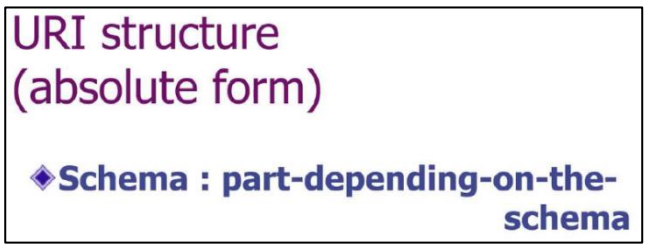
One can think that we already know everything about the addresses (URLs) (www.mysite.com). But there is more. Beyond the URL there is the **URI (Uniform Resource Identifier)**. URIs are superset of the web addresses (https://...) which contains both URLs and URNs.

The classic shape "*www.mysite.com*", that all we know, doesn't belong to any standard! It is just the browser translation for convenience. Example of URIs:

- http://corsi.math.unipd.it/wim
- news:it.culture
- telnet://example.net/453
- mailto:massimo@gmail.com

URL stands for **Uniform Resource LOCATOR**. URLs are those URIs that provide the location to find the resource.

URN stands for **Uniform Resource NAME**. They are just words, ways to identify a resource. URNs are those URIs that stay unique and persistent even when that resource doesn't exist anymore, or it is not available.



The *schema* defines the semantics (the meaning) of the URI. For instance, in http://corsi.math.unipd.it/wim the schema is **http**.

URIs can be:

- **Hierarchical**, with the form: **Schema :// authority path ? query**.

  The authority tells us the name that controls the resource. In http://corsi.math.unipd.it/wim, *corsi.math.unipd.it* is the authority. The path is composed by segments, each one separated by **/segment**.

  "#" is a reserved characters in URIs, needed to separate the URI of an object with an identifier to a fragment of the object: so URIs can refer not only to a resource, but to a subpart too.

  The query represents the input parameters, the searched things.

- **Opaque**, with the form: **Schema : opaque_part**.

  Like, mailto:director@cnn.com. *mailto* is the schema and *director@cnn.com* the opaque_part. Used to send automatically mails or to phone to someone: **tel: +358-555-1234567**, fax: **+358-555-1234567** links.

URI can be also absolute (so, already complete) or **relative** (so, incomplete). To complete a relative URI, it has to be turned into an absolute URI via information deriving from the context.

Beware of using relative URIs because they can be completed in something that we don't want. The most common error is: in my site http://www.mysite.com insert links like www.othersite.com → this error is related to what people see in address bar of a browser thinking that is the link. In this case we must use the completed version http:www.mysite.com/www.othersite.com.

The typical form of URN is this: **urn:NID:...** Where NID is the so-called *Namespace Identifier*: in a sense the "schema" of the URN.
For instance, think about the book ISBN, we can see ISBN as a URN → *URN:ISBN:0-395-36341-1*, this doesn't tell us the specific place where to find the resource but all places where we can find that object.

Back to URI: what is the data format of http://www.sito.it/a/b.html? (Italian? English? French?). That information is not part of the URI/URL: the web address is a black-box string, and every property depends on the schema, not on the string. HTTP provide methods (*content negotiation*) to transmit the correct data format.

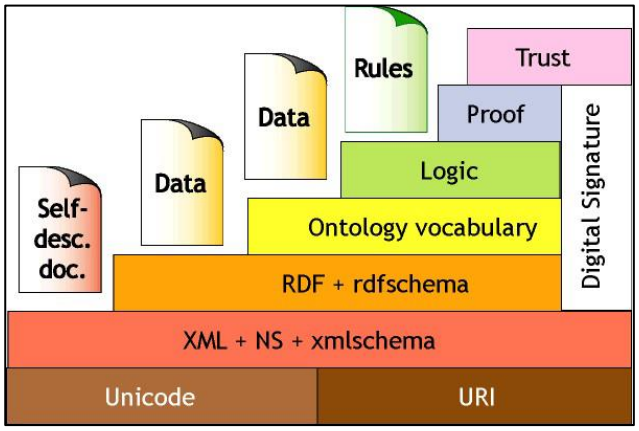
7. Semantic Web

Remember that **the real power of the web is the aggregation of information** and not its distribution. The aggregation in fact is more fragmented: there are several sources of information, but they are all separated → search engines try to aggregate all these sources.

Many years ago, it was thought that the evolution of the web would be using **automatic agents**, which locally interfaced with the web and carried out user requests (example, the user asks the agent: "*send a rose to my girlfriend*" and the agent do that).

The main problem is that many years ago only HTML was used which is simple but limited, so the agent could not understand the requests at 100%. For example, without having a precise explanation of the term "rose", as a flower, the agent could search in the web for anything that was described as pink and send the one with the lowest price (for example a sadomasochistic book!).

To solve this problem, different technologies begin to appear, trying to facilitate automatic aggregation of information and even more, trying to enable **automatic reasoning on such information**. This can be done adding **semantics** (meaning) in the appropriate way, so to enable information understanding and reuse.



This leads to **The Semantic Web Tower**.

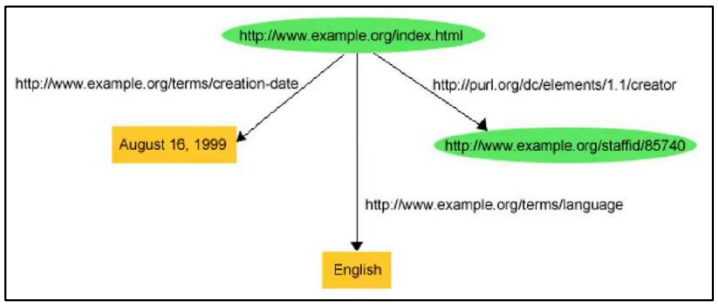
At the bottom levels we have all the elements that shape the web we know and over these levels we have other elements that help the web to do what it does nowadays.

Let's explore these levels.

7.1 RDF

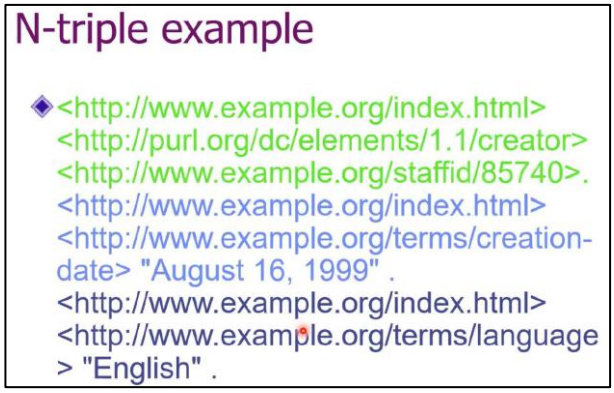
RDF stands for "**Resources Description Framework**", a framework that helps machines describing the resources. It describes relationships and concepts. The idea is to use the RDF language to write things on the web in a way that also the machines can understand them.

Base grammar: a "sentence" is made by ***subject + predicate + object***. How can we use this basic grammar with the machines?

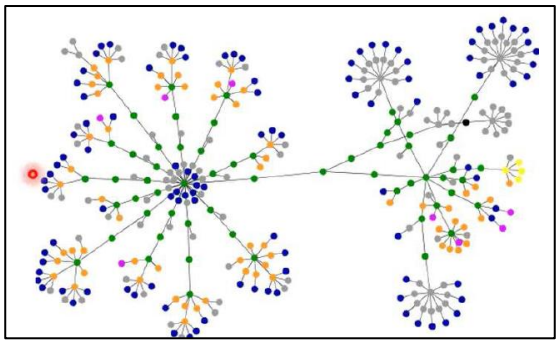


We can think about the **RDF as a graph that connects information to write sentences**. For instance, here we have that a certain page has been created on a certain date, by a certain author and that's written in English.

RDF is a model which can be written in different ways, using different standards. The two most used are: **XML** (specific dialect) and **N-triples**.



Different colors represent different sentences.



RDF enables easy aggregation of information sources. Because essentially, merging graphs gives again graphs which become *forests*. **The graphs don't always stay distinct, but they can melt together → we can use URLs to distinguish different forests, so different concepts**. We are using same names on same resources!

Anyway, this is not enough; RDF gives the base layer (a merging model to aggregate information) but we need more. We also want to **classify information** to have a better map of information.

7.2 Ontology vocabulary

Ontologies represent the technologies used to classify information. This technology gives us the information of "type X", where "type" is a ***semantic type*** (the meaning of an object). Note that there is also the ***syntactic type***, which is the one used in computer science to classify objects datatype (integer, string, etc).

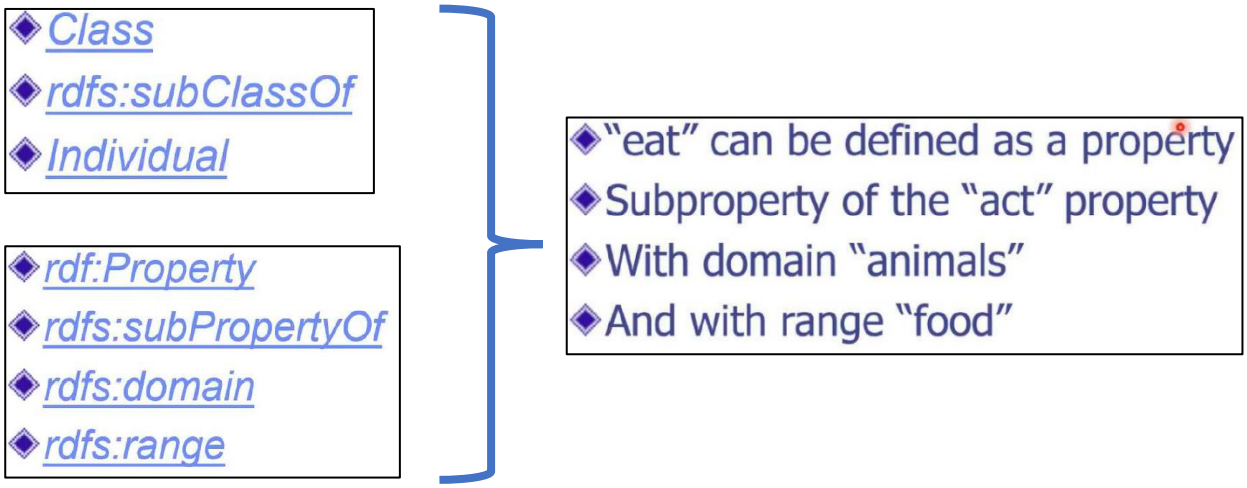
The semantic types, the more important ones, are more commonly called ***classes***. So, an **ontology** is basically composed by a collection of classes. Each class can contain objects and so these objects belong to the same class.

Example: the wine ontology contains red wines, white wines, pink wines, Merlot, Cabernet, etc. The class *red wines* can contain the objects *Merlot bottle from 1999*, *Cabernet bottle* and so on.

The interesting thing is that an ontology can also have an internal structure (so, not just being a "flat" collection of classes). The easiest structure is the so-called ***hierarchical structure***: a class can be contained into another class and so on. In general, the hierarchical structure is provided by a containment relationship among classes, that can be true or false.

More structure means more power. For instance, we can now perform ***information integrity check***, and also ***deductions*** (reasoning) → "I want to find a wine that is ...." → we can do that now.

We can support ontologies through RDF Schema:



RDF-Schema (RDFS) provides basic support: what is called the ***taxonomic layer***. Taxonomies are the first step, that can be further enriched.

The big advantage of RDF is reducing almost to zero the complexity of information aggregation. **How do we establish connections between information? Is it so easy via URLs (the names of the web)?**

We need some rules. Tim Berners Lee invented some axioms that are global rules at the base of the logic of the web:

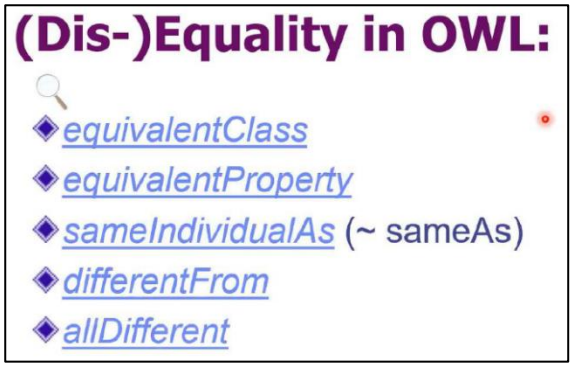
- **Axiom 0**: any resource anywhere can be given a URI and machines know how to manage URIs.
- **Axiom 0a**: any resource of significance should be given a URI.

Remember that is not so easy to correctly name a URI → **URI Variant problem**: in general, there can be many variants (URIs) for the same concept, the web is decentralized (for someone the best URI for a person is his/her homepage, for others his/her homepage). **Usefulness of URIs decreases exponentially with the number of variants**.

Axiom 1: it doesn't matter to whom or where you specify that URI, it will have the same meaning.

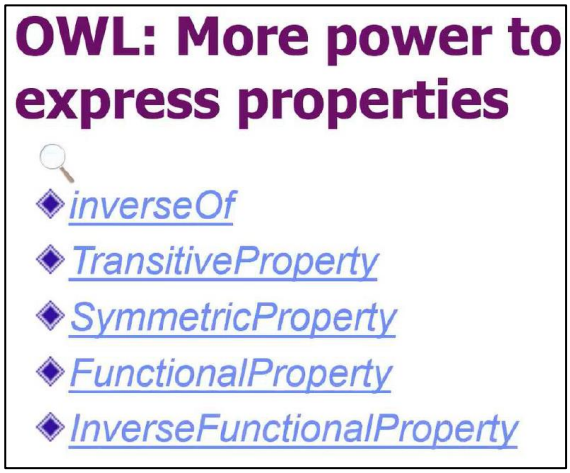
7.3 OWL (Web Ontology Language)

The basic support provided by RDF-Schema has been extended with a specific layer in the semantic web tower: **OWL (Web Ontology Language)**.



If two words are syntactically different, they can be semantically equal! OWL technology can manage the equality/inequality of two words.

OWL allows to **reduce the problem due to URIs variants**, establishing relationships among different ontologies. We can map (translate) from ontology to ontology.



7.4 SPARQL

When we have more information, we can make more reasonings about that. We can use relationships and more to define a ***logic***: what we would like is then to do automatic calculations → so having an ***executable logic***. But already the super-simple first order logic (∀ for all, ∃ exists) is not decidable → in computer science terms, it just means the corresponding program might not terminate!

How can we deal with that? Think of other contexts: for instance, databases: **SQL**. Providing the right query, we can check for relationships and find what we want to find → a computable program is created from a query. In its base version, like SQL-92, every SQL query (program) terminates!
SQL is not Turing-complete: it has limited expressive power but terminates → we are not able to describe any type of relationship, but at least SQL generates programs that terminates. **SPARQL is the reference query language for RDF**.

IDEA: think of the triples that build the knowledge graphs: *subject verb object*. The basic tool of SPARQL is provided by ***graph pattern matching*** using triples, for instance "***?*** *verb object"* → we want to know, through a certain query, what is ***subject*** related to a given *verb* and *object* (the outputs will be all the subjects that have a relationship with the given verb and object).

Structure of a basic query:

- Similar to SQL:

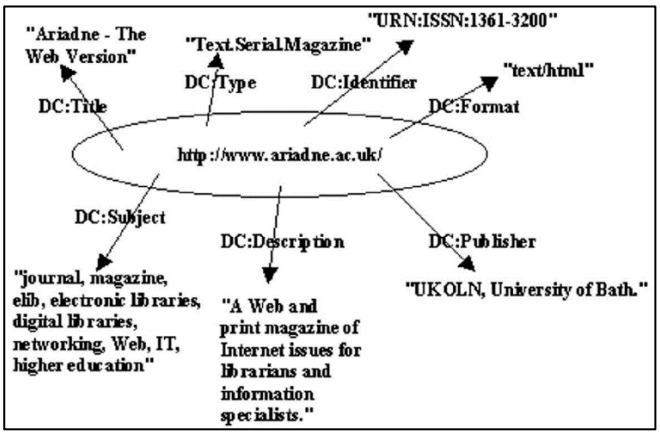
```
PREFIX ...
SELECT ...
FROM ...
WHERE {
  ...
}
ORDER BY ...
```

Let's model some data to understand how to manage the information. For instance, let's see **DC** and **FOAF**.

DC (DUBLIN CORE)

The Dublin Core is one of the first attempts to structure the web in a semantic way. The standard to describe the basic properties of documents. 15 information elements about documents:

- Title Creator / Subject / Description / Publisher
- Contributor / Date / Type / Format
- Identifier / Source / Language
- Relation / Coverage / Rights



What is the power of DC? It can extract all these document properties from a webpage.

This is helpful for search engines inspections because DC helps them to understand better the content of a site.

We can write them with HTML in this way:

```
<meta name="DC.title"
      content="What is OWL?" >
<meta name="DC.subject"
      content="Semantic Web" >
<meta name="DC.author"
      content="Massimo Marchiori" >
```

FOAF (FRIEND OF A FRIEND)

FOAF is the standard ontology for the Social Web.

How to describe a person

- The class **Person**, with properties Name / title / firstName / surname / nick / schoolHomepage / workplaceHomePage / workInfoHomePage / myersBriggs / phone / webpage / weblog

Example

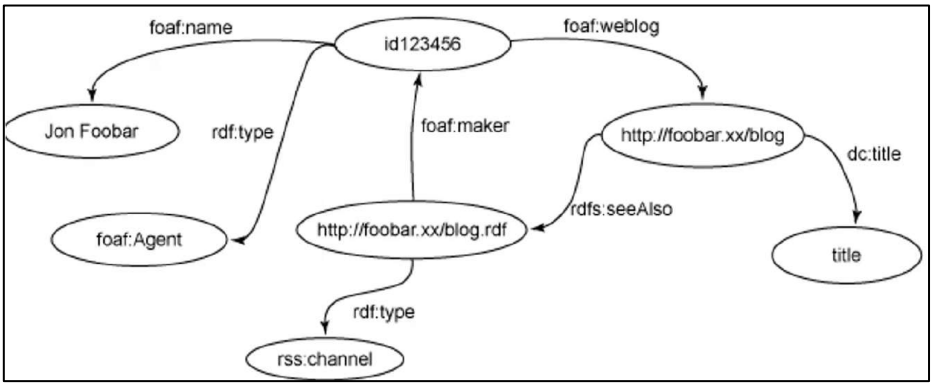
- <foaf:Person>
  <foaf:name>Massimo Marchiori
  </foaf:name>
  <foaf:mbox
  rdf:resource=mailto:massimo@w3.org />
  </foaf:Person>

  mm foaf:name "Massimo Marchiori".
  mm foaf:mbox "mailto:massimo@w3.org".

In this way we can write FOAF objects that must be characterized with some properties, like name, resources, etc.

- The basic property is **foaf:knows**
- Example:
  <foaf:Person>
  <foaf:name>Massimo Marchiori
  </foaf:name>
  ....
  <foaf:knows>
  <foaf:Person>....</foaf:Person>
  <foaf:knows>
  ...
  </foaf:Person>

PlanetRDF has been one of the first "fully semantic" websites. Let's see a little piece of its structure (knowledge graph) about bloggers within the site.

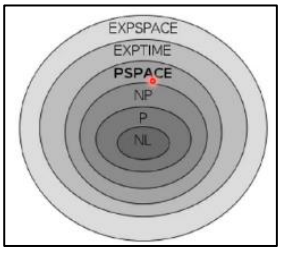


In the above picture we can see part of PlanetRDF structure. Everything has been written with the principle of the semantic web. From this knowledge graph we can query some information: we want to search for the web address of "Mark Twain" 's blog:

```
PREFIX foaf:<http://xmlns.com/foaf/0.1/>
SELECT ?url
FROM <http://planetrdf.com/bloggers.rdf>
WHERE {
  ?someone foaf:name "Mark Twain" .
  ?someone foaf:weblog ?url .
}
```

In the Semantic Web, knowledge graphs originate from multiple sources, and can anyway contain partial information → **SPARQL allows to deal with partial information** by using the OPTIONAL command:

```
PREFIX foaf:<http://xmlns.com/foaf/0.1/>
SELECT ?name ?picture
WHERE {
  ?someone foaf:name ?name .
  OPTIONAL {
    ?someone foaf:depiction ?picture
  } .
}
```



Back to the main question: is SPARQL language decidable? YES, but how much is it efficient?

SPARQL is decidable and stays in the PSPACE problems. Remember that is the same as SQL.

COMPUTATIONAL COST

What about the OWL technology? OWL is more powerful than RDF (that use SPARQL language), but more power means more difficult to compute. We need to choose between decreases its expressive power, making it decidable, or keep the expressive logic but sacrificing computability.

SOLUTION: 3 OWL layers according to what are the specific needs:

- **OWL Lite**: decidable, every query always gives a result (near EXPTIME).
- **OWL DL**: decidable, more powerful than the lite version → a good compromise (EXPTIME).
- **OWL Full**: no restrictions and so it is undecidable.

Note that also the first versions are near exponential time of computation. Anyway, the complexity class is just a statistical measure, just the top of the iceberg. **What really matters is the *average complexity* and the *context* → the exponential class is just the worst case!**

For instance, Google and Bing use EXPTIME algorithms that can take, in the worst case, days to execute the users' queries!!

7.5 Applications

LINKED DATA

Beyond the "logic" levels seen above, the semantic web uses linked data to fill knowledge graphs.

LOD (Linked Open Data) are available for free, and they can be classified from one to five stars:

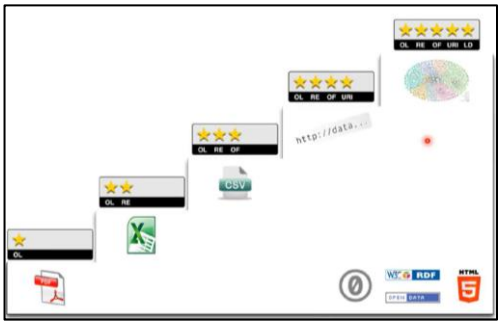
★: data available on the web but with an open license (open data)

★★: as above, with data in a machine-readable structured format

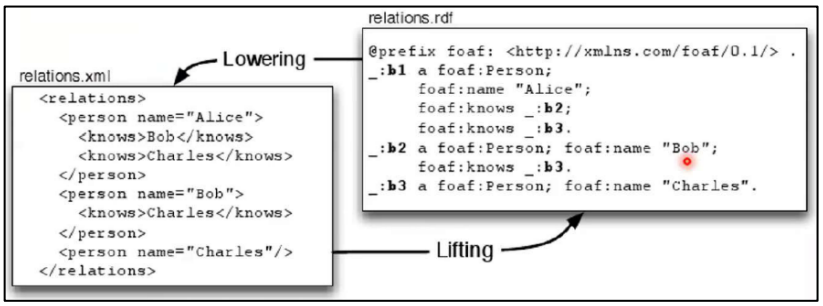
★★★: as above, using a non-proprietary data format

★★★★: as above, in Semantic Web format (RDF*,OWL)

★★★★★: as above, with data linked to other data sources to provide context



There are interesting *middle zones* in LODs. In particular, the ★★★ layer (non-proprietary data), not necessarily RDF*/OWL format. We can keep a 3 stars information and power up to 4-5 stars: going from the non-semantic web world to the RDF world is called ***lifting***. The other direction is instead called ***lowering***.



Lowering and lifting are important to do ***mashup***: join the XML/XHTML/HTML world, and all its amount of information, with the RDF / Semantic Web world, and vice versa.

Example of lifting: from SQL to RDF → ***D2RQ*** (d2rq.org) allows to automatically pass data from SQL to RDF. Can work also as a server, giving immediate web access to data as RDF. Also, ***Triplify*** and ***Openlink Virtuoso*** can do that.

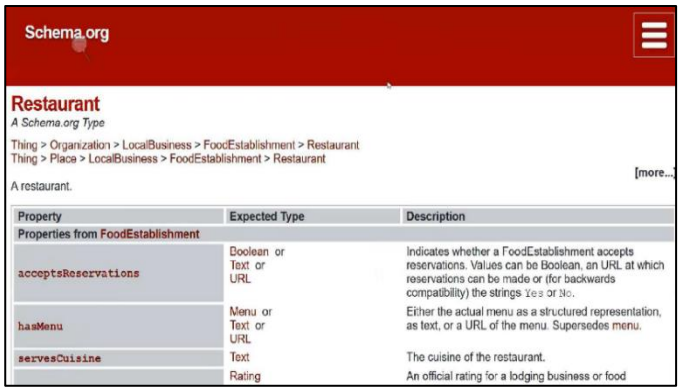
For the other data (web pages, text, pdf, word, etc) we need **NLP (Natural Language Processing)** tools that analyse (to understand) textual/hypertextual information and allow automatic lifting. One of these types of tools are ***Open Calais***.

THE REAL POTENTIAL: An example of how much potential these tools offer is provided by website www.wikido.com. This site takes from a set of selected sites information which with *Open Calais* is transformed into high-level information level obtaining structured data. Google does the same with queries. Wikido is lifting the information from simple HTML documents to a semantic web version to increase its potential.

To have at least 1 LOD star data must be on the web, but how? We could just make the data available as a file in a certain URL address: in case of 3 LOD stars this is called ***RDF dump***. The problem is that *dumping* data is just not very practical (think of Big Data).

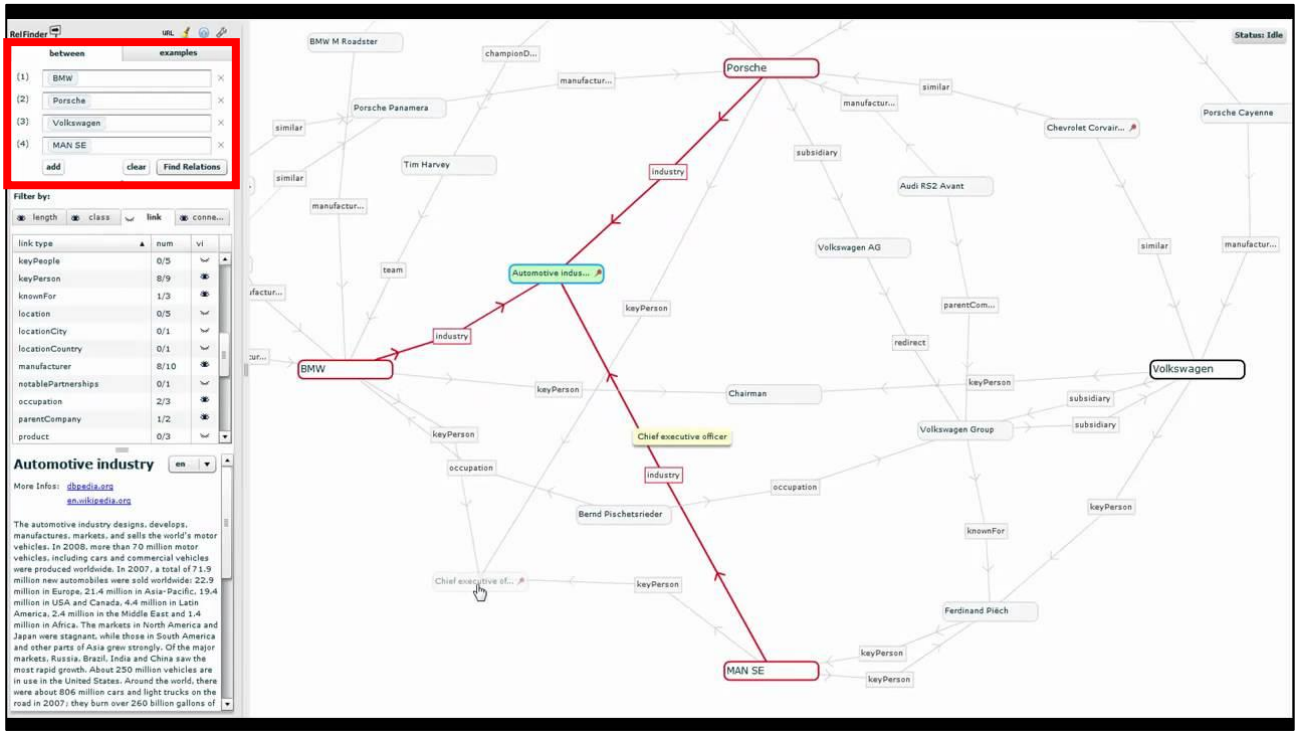
The other way is to offer a more structured access, for instance via an interface. Instead of providing a raw file, we could offer a SPARQL ***web service***: much more powerful and flexible. In this case we talk about SPARQL ***endpoints***: a connection where users can filter data. We can access to endpoints via GET: we can do an HTTP GET to the SPARQL endpoint, passing various parameters in the query part of the URL.

DBPedia is essentially the semantic version of Wikipedia. DBPedia has its own ontology, and also reuses the reference web ontologies, so to be interoperable. More than 4.8 million resources, also with endpoints. It uses ontologies provided by Schema.org, which defines the most important/used ontologies for the web. For instance, about *people*, *products*, *events*, *places*, etc.

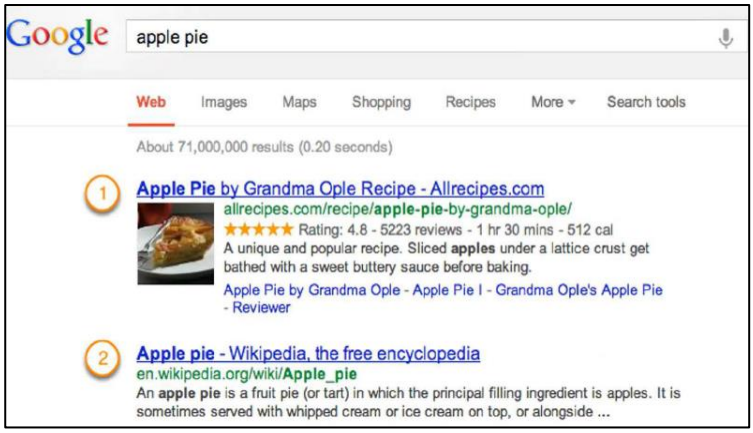


This represents the ontology for ***restaurants***. In the web we can find these kinds of information, but they are all separated!

With the power of the semantic web, we can power up websites → search engines can enter to restaurants websites and take this kind of information to collect and give them to the user.



[RelFinder] tool shows us the real potential of the interconnected information. Given 2,3 or more objects (car, place, whatever), this tool provides the relationship between the selected object taking the information from DBPedia.



Note that every state usually makes open data available. For instance, in Italy (dati.gov.it).

Search engines love this type of data. Google has full support for the semantic web. It understands all Schema.org. And search engines, like Google, reward sites that has support the semantic web putting them up in the results ranking.

All this is integrated with the much bigger knowledge graph (used also for ranking): so, users only see a very limited amount of information. Every time we make queries to a search engine, we get the answer, and this answer is taken from this enormous knowledge graph.

FREEBASE CASE

We see what is LOD but big companies, like Google, do make open their data or not? **Freebase** is the main source of open data sponsored by Google. So, Google doesn't just use other public data but also offer its set. In Freebase data are offered as N-Triple RDF, also with endpoints (so we can query the resources).

In 2015 the quantity of data offered by this platform was around 3.1 billion of triples (almost 250GB). But now if we search for the site, it doesn't exist. Google moved to Wikidata project which was a better idea because data were already uploaded in the right way, and anyone can contribute: in this way Google hadn't to spend only its resources to maintain a site like freebase but it could involve everyone to contribute in collecting data.

8. Mobile

Now let's talk about the **mobile apps**. Apps are not web, but they are anyway related (UI) and constitute a big market. There are some parallelisms with the desktop usability, especially because in the app the main thing to consider is the UI but there are also some differences with the desktop world.

8.1 Differences with desktop

The main causes of the difference between the classic and the mobile world is related to a **different execution model**, this is due to the three base components of mobile:

1. **Being mobile**: small and portable device.
2. **Screen size**: one of the main differences.
3. **Fingers**: different from the mouse.

Note: **beware of the target**. Facebook, not so well known, has three mobile version:

1. *m.facebook.com*: for non-touchable devices → smartphones are not used in all part of the world.
2. *touch.facebook.com*: for touchable screens, the most used by smartphones.
3. *0.facebook.com*: it's a super-fast version, limited-bandwidth version. For instance, images/links aren't inline and need an extra click to be seen. This version of Facebook tries to always use text only. Speed is higher but users are less happy because more clicks are needed. What is the sense of this? 0 version is offered free in all those places where connections are slow and quite costly. Facebook is a way to use connection for free → in this way Facebook took the whole market before the market started to appear!

8.1.1 Being mobile

Being mobile means that connection type changes. In fact, 3G/4G/LTE networks are on 5-40% slower than desktop connections. So, every site pays a ***time price*** when is seen on a smartphone. We know that already, having spoken about the importance of time for users → so, if a page takes 40% more, it takes more time out of any timer.

But the problem is more serious: it's not only a matter of session timer of global timer, but this is also an added delay for every single page. And so, users compare that with the average loading times.

In the desktop case, users wait for a maximum time ***2 seconds*** per page. Beyond this time, they have a delay perception, and so corresponding discomfort to the site → local loading time of every page is an important aspect to consider: every page should be faster enough.

In the mobile case, it is the same, always 2 seconds! So, we have to be careful: adapting the visual layout is not enough, we have also to be careful of the possible penalties of the mobile network: if a site takes almost 2 seconds in the desktop page, how much time will take in the

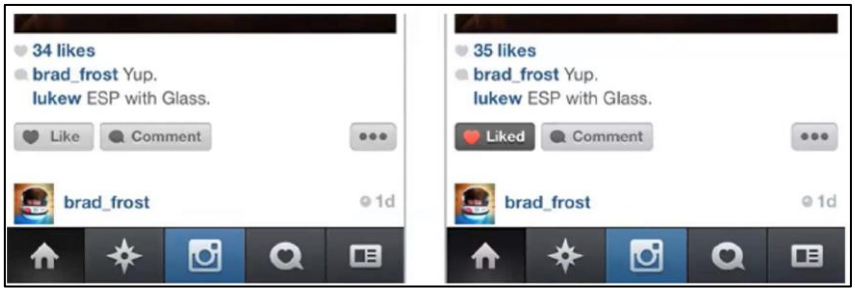
mobile environment with a slower connection? The solution is to have **light version for mobile versions**.

A fundamental factor for the app is ***responsiveness***. Delay is not so bad if it's justified from the context (for instance, photo upload, etc.). This is always valid for any action that needs a network connection (even if it's not our fault).

A solution could be "copy desktop behaviour" in these scenarios: progress bar, spinner → we are warning user about the time. But this doesn't really work on mobile because progress bars are signalling (make explicit) a problem, and **the delay is perceived more when users see a progress bar in action**.

TRANSITIONING

There are other techniques that work better, like ***transitioning***: keep the user busy, for instance with animation and so on. A particular case of transitioning is the ***skeleton screen***: if we don't know the final layout of an action, we can start drawing it even if we didn't receive all the data yet.



Example: for instance, think about the action to put like on facebook/instagram posts, a red heart appears **immediately** → user has immediately a response to his/her action. If a lot of people make like at the same time the servers cannot process the like immediately but we don't want users know this, so we'll make them happy with an immediate response to their action.

PREEMPTIVENESS

There is also the scenario where a user has to upload a photo (maybe an avatar). The classic approach here is to let the user choose the photo, then ask for a description, then upload. We can do better by loading the photo as soon as it's been chosen → instantaneous upload and users are happy.

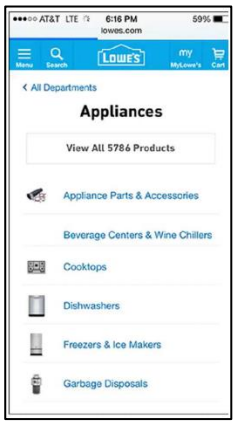
8.1.2 Screen size

The main issue of having a small size is that **a page will be seen hardly without scroll**. But when the scroll is bad for mobile? Horizontal still bad for mobile, like desktop. Vertical is the best scroll for mobile and often it is not so bad → this is related to the physical and mental computational effort which is much lower on mobile. In fact, scrolling is done via ***gestures***, without using the mouse in specific zones.

But beware, **vertical scroll may also be bad** for users, when it's used to offer choices (menus, list of products). In such cases, the user has to keep in mind the content of what above until the choice is completed, and this generates mental fatigue, proportional to the size of the hidden information!

Note: the problem is more serious on mobile, due to the smaller screen size.

Consequences: it's better to minimize vertical scrolls for choice lists, and so for instance in some cases even avoid images.



Images are justified only when they are related to the final products (for the same reasons already seen for the classic ecommerce case). This anyway can lead to the memory efforts mentioned before, and so the best solutions are anyway those that minimize scrolling operations.

A middle way could be using text with small images that sounds good for categories/menus but not for the final product.

To remedy for the small size, one solution can be to use **icons rather than text for buttons**. The problem is the same as for the desktop: this works if users know that the icon means (e.g., it's ok for the search lens). Unless the icon is super well-known, users always prefer text to icons.

So, if we want to use icons the best is to also use text and if we really want use icons without text we need to respect the ***explainability*** principle: keeping pressed the icon the user can obtain textual information on its action.

Moreover, a companion principle should be used, ***escapability***: a user can always "escape" from an action just by moving away the finger from the icon. This principle is also valid, in general, for any touch action.

ADVERTISEMENTS

Having a smaller screen also implies a redefinition of what being "invasive" means for an advertisement banner. Two main categories of size for banners: fullscreen (interstitial ads) and non fullscreen size (smart banners). Users hate both of them but they hate more the smart banners because they remain permanently there while the counterpart of interstitial once they are closed they are gone.

Smart App Banner are used to publicize the site app, but this is supper annoying for users, it is the equivalent to the pop-up for the desktop. It is an obstacle to what the user wants to see.

It's not a bad idea to publicize your app but not in this way.

8.1.3 Fingers

Fingers have pros and cons. For instance, unlike the desktop case, ***drag*** is not problematic: there is no muscle effort to keep pressed a mouse. This aspect makes gestures particularly appealing to users.

With fingers we need to be careful about actions activated by fingers. Pressing should be dependent on duration, but just distinguish between *tap* and *drag* (prolonged press) → users don't know about this threshold. Some mobile user interfaces/apps instead assign different actions to different press durations: this causes errors and high user stress.

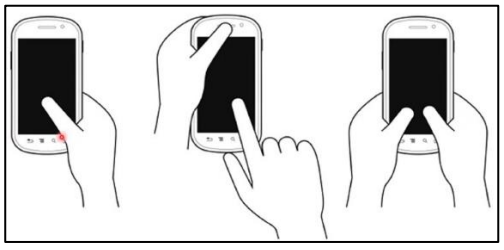
The major problem is that fingers are bigger than a mouse pointer, they are usually referred in the mobile world as "fat fingers": is that often fingers are the cause of clicking problems, for instance, for pressing really tiny buttons. The average finger is 11 millimetres wide, 8 for children. So, any clickable must be big enough so to be well centred with a finger.

Consequences: the minimum size of any clickable is 7x7 millimetres, 9x9 to make users happy. Around, we need a padding safety zone wide at least 2 millimetres.

It's amazing the number of mobile websites and apps that don't respect these basic guidelines, and so has severe usability problems. To partly dimmish these problems, we can use a principle which is anyway important in itself: ***reversibility*** → every action taken should be reversable, even more in situations of potential errors due to the fat fingers.

8.2 Fitts law for mobile

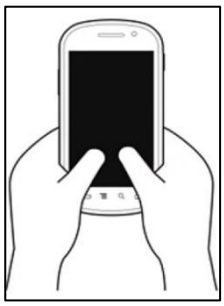
Remember that Fitts law let us to quantify how much effort is required from the user to do a certain action. But all that we see about Fitts is true also for mobile? There's something that changes, due to the different environment.



For instance, in Fitts for mobile, the size of the object (like buttons) matters, but also the imprecision due to the finger and there is also another complication: ***mobile grasp***.

There are five classes of possible grasp, **different types of interactions by the users** that need to be consider when a mobile site is designed: the two-handled symmetrical case (last case in above image), two-handed with one active hand, one-handed and the two left-handed cases.

For instance, if we are using just one hand (as in the first case of the above image) there are some areas that are not reachable by users → too much effort, we need to take care about it. The main error is to change grasp to reach different area of the site.



Each of this fives cases behave in a different way. For instance, using thumbs make the fat finger problem worst → clickable areas should be +2mm wider.

Note that also if the button to press is the right area it is difficult to press if too small, this is due to the thumbs. If we are working with tablets the problem getting worst.

The best interface must take into account all these constraints, and for instance keep all the controls in the lower part of the screen → that's where the basic smartphone commands already are.

Another note is that both on smartphones and tablets, the screen shape, and so the reachable areas, change also when passing from normal lo ***landscape*** (horizontal) position. In these cases, liquid layout can be problematic: size of objects that scale accordingly with screen size.

Remember: we can use the magic places, the borders! The advantage over a desktop web site is that now the "windows" is already maximized, and so we can always use borders, for instance via a ***fan menu***.

Note that the area under the finger used to activate a fan menu is hidden and we need to take into account this when we design fan menus.

8.3 Apps

Apps are similar to the website interfaces, so the above considerations are still valid with apps but we can go deeper. We all know the success of apps and this is a consequence of computational effort minimization: **apps minimize access time to the functionality of interest**. Moreover, the centralized app stores minimize the effort to find the right app → compared to the desktop apps that are everywhere on the web.

Many users, more than a fourth, use apps more than 60 times a day and just for this they win over mobile web → **usability always wins**, is what is important for users. Smartphone users pass on average 86% of their times on apps and 14% on the web.

The most attractive points for users on apps are **games (32%)** and **social sites (28%)** → note that this trend is still growing.

Apps are so trendy; we might be tempted to use them as fast recipe for success, everyone should design an app! That's good, but be aware that just because of their success, they also have a **very competitive environment**. Apps can enter in a ***death spiral***: 26, 13, 9 → 26% of

apps are used just once, 13% twice, 89% only three times, etc. This means that an app can lose the 26% of user just at the first usage!

Apps are like butterflies they have a very short lifetime: from 4 months to 1 year. For instance, games (the most attractive) paradoxically have an average lifespan of 4 months.
Important signal on duration: if users' growth last more than the critical period of 3 months, then the app will likely last much longer, otherwise it will die quickly. Just for this reason, it's crucial for an app to be found by users.

Apps can be found via the app store and so with a search engine, not Google or Bing, the proprietary engine of the app store.
The corresponding problem for mobile apps is called **ASO (App Search Optimization)**. Similar to the SEO but with differences. It also works via keywords, but where we can put them? There are not so many places where to put keywords: the app description and in the best place of all, the **name**.

Google and Apple also use a lot of information coming from the ***global social system***. For instance, downloads, usage time, ratings and reviews, uninstalls, brand and also positive/negative parts coming also from other systems (web and email). Google and Apple look for these things and they assign a number to each of these features: apps with great numbers get rewards.
Note that those inspected aspects are not independent from each other: if an app has a lot of downloads but the usage time is low, it's not a great app.