# IUM

Federico Ferdinandi

January 2023

## 1 Need Finding.

Un task è un compito che l'utente vorrebbe poter svolgere usando un interfaccia, un applicazione .
Per arrivare a una lista di task c'è prima una fase di **ricerca dei bisogni**. (need finding)

*La ricerca dei bisogni/need è un'attività fondamentale nella progettazione dell'interfaccia utente* poichè ci consente di **compredere i nostri utenti e le loro esigenze-problemi-frustazioni-modi di fare-necessità-obiettivi** e di pensare e progettare un prodotto adatto a chi lo utilizzerà.

Per trovare i bisogni di un determinato utente ci sono due tecniche di osservazione: **interviste** e **questionari**.

Durante la fase di need finding **osserviamo cosa fanno le persone, quali sono i loro obiettivi, che strumenti usano, il contesto e le attività che svolgono**.

Bisogna **osservare senza avere già in mente quello che stiamo cercando** ed essendo **aperti a scoperte impreviste**; **prima bisogna trovare tutti i problemi, la soluzione alla fine**.

**Dobbiamo osservare senza coinvolgere l'utente**: passare del tempo vicino senza disturbarlo.

**Mettersi nei panni dell'utente, chiedere o svolgere task insieme**, facendo questo si potrebbero trovare delle necessità in quello che l'utente fa tutti i giorni. Prestare attenzione alle pratiche: trucchi, scorciatoie (comandi veloci tastiera), modifiche, uso di ausili (es. per la memoria) e notare gli errori.

Bisogna evitare di influenzare gli utenti: non dire la tua approposito di quello che pensi, ma farsi dire dagli utenti, tralasciare i propri problemi e obiettivi e lasciar parlare gli utenti, non dire ad esempio: "che ne pensi di questo, . . . ".

La percezione dell'utente spesso inconsapevolmente errata, bisogna **prestare attenzione a ciò che gli utenti fanno e alle loro emozioni,paure e frustrazioni**.

Evitiamo di confondere needs e soluzioni poichè dall'osservazione (interviste e dai questionari) vogliamo trarre i need, le necessità. ***I need aprono nuove possibilità, mentre le soluzioni limitano l'innovazione.***
Sbagliato partire dalla soluzione ma anche dal need (cioè il need deve essere relativo all'utente e non a noi stessi), lo deduciamo guardando l'utente (quindi con interviste e questionari).

*I sviluppatori non dovrebbero chiedere esplicitamente cosa vorrebbero gli utenti dei videogiochi per migliorare l'esperienza dei task, ma in questo caso i giocatori, sono esperti e specializzati, sono raffinati nel grado di comprensione e del design del gioco e quindi la richiesta va bene.*

## 1.1 Interviste.

Le *interviste* sono informali, economiche e **permettono di approfondire e fare delle scoperte inaspettate**.

Bisogna stabilire un tempo e un luogo confortevole per gli utenti dove ci possiamo presentare e spiegare l'obiettivo. Iniziare con le domande aperte e dare tempo per le domande.

Si consiglia *time consiming*: rassicurare le persone e dire che l'intervista dura poco .

Intervistiamo sia persone rappresentative degli utenti, utenti di un sistema simile e persone che non sono utenti potenziali perchè anche da loro si trovano delle necessità e da chiunque si può trovare qualcosa interessante da dire.
Per trovare i partecipanti possiamo utilizzare i social network, amici di amici eccetera.
Se si stanno facendo interviste ad un negozio possiamo sia intervistare clienti che commessi.

### 1.1.1 Linee guida.

Possiamo preparare le domande ma non tutte per eventuali sorprese. Sarebbe opportuno registrare l'intervista e incoraggiare l'intervistato ad approfondire le risposte date.

Si deve chiedere il consenso per privacy, per i dati personali e rimanere neutrali rispetto alle posizioni espresse dall'intervistato.

### 1.1.2 Attenzione ad alcune domande

- Quelle la cui risposta è scontata, quelle troppo generiche e che contengono già la rispota.
- Quelle che chiedono all'utente di sostituirsi al designer
- Quelle basate su scenari ipotetici e che chiedono quanto spesso fai qualcosa. Indicare esattamente quando (non lontano nel tempo). Ad esempio chiedere quando sei andato l'ultima volta.
- Utilizzare domande aperte ed andare sul concreto
- É la funzionalità [x] importante per te? Che cosa vorresti in un'app? Che ti piace in [x]?
- "Come è stata presa la decisione? Avete fatto un incontro? Qualcuno ha deciso senza di te?..."

### 1.1.3 Interviste specifiche ad utenti spceifici.

**LEAD USER**: interviste fatte ad utenti che fanno uso particolare di app già esistenti .
Questi sono utenti che hanno dei need nuovi (presenti nel prossimo futuro) che scoprono prima di tutti gli altri trovano da soli le soluzioni ai loro need beneficiandone.
Ad esempio nel passato gli utilizzatori di Mac erano pochi, questi utilizzavano il mouse a differenza degli altri (tastiera). Questo vuol dire anticipare i need (oggi utilizzano tutti il mouse)

**EXTREME USER**: utenti che spingono un sistema esistente all'estremo, stressano i sistemi . Trovano needs amplificati. I loro needs sono i needs anche di altri, ma più visibili.
Utenti che visualizzano molte email, trovano need per non visualizzare e-mail che sono inutili.

**INTERVISTE AD ESPERTI**: intervite fatte ad esperti per sfruttare le conoscenze di esperti del settore .
La loro esperienza offre informazioni aggregate attendibili sul comportamento degli utenti.

### 1.1.4 Interviste particolari

- **history interviews**: *per comprendere sequenze di eventi che spiegano comportamento degli utenti.*
- **process mapping**: *chiedere all'intervistato di descrivere tutto il processo.*
- **laddering**: *Chiedo di spiegare perchè quella persona fa così.*
- **cultural context**: *per conoscere il contesto culturale.*
- **intercepts**: *una sola domanda per puntare alla quantità.*

Di solito quello che si fa è fare prima delle interviste e da lì passare ai questionari sui punti interessanti emersi dalle interviste.

## 1.2 Questionari.

I *questionari* sono set di domande uguali e specifiche per tutti gli utenti (come google form) e sono effettuati in tempi veloci per ottenere numeri e quantità di risposte.
Possono essere analizzati più rigorosamente, si possono fare statistiche e sono quindi orientati all'analisi .
Le domande possono essere:

- **aperte**;
- **chiuse**: risposta singola, multipla, scala di valore ("da 1 a 10 quanti fai x...", ranking ("metti in ordine di preferenza ...")

Sono meno flessibili rispetto alle interviste e vanno accuratamenti testati prima su pochi utenti (test pilota) e quando si ricevono feedback positivi possono essere girati a tante persone.
Ci sono alcuni tool per realizzarli come Survey Monkey, TypeForm, Client Heartbeat, Survey Planet, e anche Google forms.

### 1.2.1 Rischi questionari.

Questi questionari sono buoni per una vista superficiale ma non vanno bene per un'analisi "profonda".
Perciò se sono attuati dopo le interviste sono molto efficienti, altrimenti ci sono anche problemi. Ulteriori problemi se le domande sono basate su dati "sensibili" o sulla memoria dell'utente.

## 1.3 Diario.

Il *diario* è importante per il need finding per **raccogliere informazioni su attività sporadiche e studi longitudinali che durano nel tempo**.
Ad esempio posso tenermi un diario per per registrare quando ho mal di testa. Vado a registrami anche quello che mangio per vedere se è collegato al mio mal di testa ,...

## 1.4 Pager studies e Camera studies.

Con questa tecnica si **chiede all'intervistati di fermarsi in un momento particolare della giornata e in un determinato posto di prendere nota di quello che sta facendo**. Se si parla di camera studies si deve utilizzare una telecamera per far vedere cosa si sta facendo
Si è asincroni rispetto all'attività dell'utente.
*Ad esempio dire a che gioco si sta giocando in qualsiasi momento.*

# 2 Storyboard.

Dalla ricerca dei bisogni(interviste e questionari) produciamo un elenco di bisogni, che vogliamo soddisfare con la nostra interfaccia/applicazione.

Da questi bisogni si estraggono quelli più interessanti , da cui si ricavano compiti.

**L'individuazione del task è un'azione che comprendere di trovare i compiti che l'utente deve poter svolgere con quella particolare interfaccia per eseguire l'attività generale e soddisfare i need principali/selezionati**.

In questo caso ci interessa il ruolo della UI, chi è coinvolto(utenti), il contesto in cui si svolge il task e i task.

Non ci interessano le funzioni e neanche come progettata la user interface, non ho bisogno di sapere dov'è il bottone, il widget, eccetera

I *storyboard* sono i disegni dei task principali della nostra user interface e sono realizzati tramite **schemi e vignette**.
Quest'ultimi li realizziamo a mano libera con disegni semplici e chiari, con poco testo e poche vignette (2,4) spiegando solamente il contesto.

Non dobbiamo disegnare le schermate perchè ci vincolerebbero nella fase successiva di prototyping.

Esistono alcuni tool che permettono di realizzare i storyboard, ma non sono necessari per il nostro corso.

Per ogni task disegniamo uno storyboard. In totale 2-4 storyboard sono sufficienti.

Bisogna considerare solamente i task prinicpali perchè fare tutto è un errore comune.

Si utilizzano schemi e vignette per:

- **efficienza**: risparmio di tempo;
- **comprensibilità**
- **comunicazione dei task ad altri designer del team**.

## 2.1 Vantaggi storyboard.

I vantaggi dell'utilizzo di uno storyboard sono:

- Mostrano un sistema e il suo contesto d'uso.
- Permette la condivisione con gli altri.
- Non lega a una particolare interfaccia.
- Può essere disegnato e corretto in pochi minuti.
- Permette la discussione e il miglioramento delle scelte.

# 3 Prototyping.

Il *prototipo* è un **modello di un sistema interattivo** (bozza dell'aspetto grafico) che simula o anima alcuni aspetti/caratteristiche/funzioni (proprietà (forma, peso) di un prodotto), ma non tutti.

Dobbiamo realizzare la UI per ogni storyboard considerando tutti gli "strumenti" disponibili come:

- *dispositi hardware;*
- *widget*
- *stili di interazione.*

## 3.1 Obiettivi e approcci prototyping.

Utilizziamo questi prototipi perchè vogliamo validare un'idea o un concetto e **facciamo delle prove con gli utenti per arrivare a una versione finale dell'interfaccia**.

Per capire se il progetto va bene dobbiamo farci delle domande come ad esempio:

- la UI progettata è completa? è comprensibile? sarà facile da usare?
- UI buona al primo tentativo? improbabile! → proprio per questo si fanno i prototipi.

Ci sono vari approcci al prototyping:

- **Throw away**: viene utilizzato per testa una funzionalità che non è del sistema finale, che andrà buttato via. Il prototipo in questo caso non somiglia al sistema finale.
- **Incrementale**: le varie funzioni vengono aggiunte una alla volta al prototipo fino a quando non si arriva a un prototipo finale
- **Evolutivo**: un prototipo serve come base per il successivo, che lo migliora.

## 3.2 Iterazione e prototipiazione.

I progettisti non sono infallibili perciò è probabile che il primo prototipo non sarà mai perfetto e si dovranno fare delle iterazioni.

Se il prototipo non va bene (gli utenti non sanno come utilizzare il sistema) ricomincio da 0, ma devo cambiare punto di partenza e parto da un prototipo diverso.

La cosa importante è avere un buon punto di partenza, capire cosa è sbagliato e ipotizzare come migliorare.

Ci sono anche dei problemi legati alla prototipazione:

- **Tempo**: realizzare prototipi richiede tempo e se questi vengono scartati, può sembrare tempo sprecato. Per questo si usa il rapid prototyping : sviluppo rapidamente, ma lo testo con calma senza nessuna valutazione affrettata.
- **Pianificazione**: è difficile pianificare processo di design con prototyping ed è difficile valutarne i costi, perchè questi dipendono dalle iterazioni, dai prototipi, ecc. . .
- **Caratteristiche non funzionali** come sicurezza, affidabilità, tempo di risposta.

Fare attenzione anche all'inerzia del design: capire le cause dei problemi.
Se vedo che qualcosa va male non insistere, ma cambiare direzione con un nuovo punto di partenza e prototipo.

# 4 Paper prototyping.

Dato che con i prototipi ci potrebbero esseri problemi legati al tempo si utilizzano nelle prime fasi i paper prototyping.

Nei paper prototyping si disegnano le interfacce sulla carta con penna. Questa tecnologia rende tutto il più rapido possibile.

Con questa tecnica si disegnano widget, bottoni, menu,... **nelle proporzioni dell'interfaccia reale**.
Non si deve tener conto dell'estetica perchè l'utente è concentrato solamente sul task.

## 4.1 Vantaggi paper prototyping.

- **Facili e rapidi da creare**;
- **Possono essere modificati i prototipi al volo prima/durante i test**;
- **Poco costoso cestinare e ridisegnare un prototipo che non andava bene**.

## 4.2 Come testare i paper prototyping.

Dobbiamo testare questi paper prototyping a degli utenti potenziali e abbiamo bisogno di 3 attori principali:

- **Administrator** che somministra il test al partecipante;
- **Observer** è il designer che prendi nota di quello che fa l'utente;
- **Human computer** che cambia le schermate man mano che l'utente clicca le cose.

Permettono di testare **interazione, navigazione, workflow (flusso di lavoro) e funzionalità.**

Si può utilizzare una tecnica chiamata **wireframe** (balsamiq) che consiste nel sostituire tutte le immagini con dei box vuoti e tutte le scritte con scritte senza senso.
Questo permette di **focalizzare l'attenzione** solamente su *posizioni, margini e sugli allineamenti.*

I **mockup** sono una *bozza di un oggetto o di un sistema, priva delle funzioni del prodotto finale.*

## 4.3 Paper prototyping aiutato dal computer (tool).

É possibile farsi aiutare da alcuni tool per realizzare il paper prototyping.
In questo caso si fotografano le interfacce realizzate in carta, si immettono nei tool e si rendono cliccabili alcuni elementi. Con questa tecnica si elimina il lavoro dell'human computer.

## 4.4 Mago di Oz.

Interfaccia che simula il funzionamento reale mediante l'intervento umano.
Il sistema da valutare è solo un prototipo, non funziona realmente e esteticamente è abbastanza credibile, **l'interattività è resa da un altro umano che manovra il sistema attraverso un altra interfaccia.**

Fare questo deve essere meno costoso che realizzare l'intero sistema, e questo prevede un interfaccia per l'utente e una per il Wizard. Questo è nascosto o remoto e fornisce input "traducendo" l'inpuit dell'utente.

I vantaggi del mago di Oz sono il fatto che l'applicazione contiene poco codice da sviluppare, è più coinvolgente e realistica rispetto al paper prototyping e si presta bene a iterazioni.
Ma realizzare il Wizard è spesso faticoso e senza allenamento potrebbe essere incoerente. Inoltre la tencologia potrebbe non esistere ed essere inperfetta.

# 5 Observational Methods.

Dopo lo storyboard si creano dei prototipi (di solito in carta) per simulare come sarà l'applicazione finale, ma non sono funzionanti perchè vogliamo ottenere mokup nel minor tempo possibile.

Chi lo dice se un prototipo è buono o meno buono? **L'Observational Methods.**
Dò all'utente il prototipo e gli chiedo di eseguire uno dei task degli storyboard e vedo se li riesce a svolgere.

Accade quasi sempre che il primo prototipo non va mai bene!

Per la valutazione del prototipo è necessario ovviamente il prototipo.
Testiamo **l'usabilità e la funzionalità del sistema** e valutiamo sia la progettazione che la realizzazione.
Vediamo se la persone riesce a svolgere il task richiesto senza problemi e nel minor tempo possibile.

## 5.1 Partecipanti.

Sarebbe ottimale avere participanti **con conoscenza di quel tipo di sistemi e del dominio ed esperienza simile tra loro** (omogenei), altrimenti si avranno risposte ai test non adatte.
Secondo Nielsen sarebbe meglio avere *3/5 partecipanti per ogni iterazione.*

## 5.2 Scenario e task.

Lo *scenario* descrive una **situazione potenzialmente reale nella quale si immedesimano i partecipanti alla valutazione**. Es. "Immagina che devi comprare i mobili da Ikea",...

I *task* sono i **singoli compiti che si richiede di svolgere ai partecipanti.**
É possibile anche svolgere valutazioni di uso libero del sistema, senza task.
Dare il prototipo all'utente e si chiede di usarlo. Di solito questo si fa in un sistema finale/avanzato quasi funzionante. Ad esempio un app per le prenotazioni del volo, gli si chiede di provare a usarla secondo i suoi bisogni. Non diamo un task specifico.

## 5.3 Obiettivo valutazione.

- Valutare la portata della funzionalità del sistema.
- Valutare l'effetto dell'interfaccia sull'utente.
- Identificare problemi specifici.

Per una valutazione di un interfaccia, è necessario dover adoperare varie **tecniche**, (dette appunto di valutazione) che ci consentono di *avere un feedback sull'interfaccia stessa.*

# 6 Tecniche di valutazione.

Le tecniche di valutazione che vengono utilizzate si distinguono in 3 grandi categorie. Ovvero in quelle:

- **Basate sull'utente qualitative.**
- **Valutazioni sperimentali (quantitative)**
- **Basate su esperti.**

## 6.1 Valutazione con Participazione utente qualitative.

### 6.1.1 Stili di valutazione.

Ci sono due stili di valutazione:

- **In laboratorio**: con dei studi in laboratorio ho la possibilità di avere un equipaggiamento speciale, posso mettere videocamere e registrare schermate dello smartphone per controllare cosa fa l'utente e avere un ambiente senza interruzioni.
  Peccato che c'è mancanza di contesto.
- **Sul campo**: con dei studi sul campo c'è un ambiente naturale con ambiente mantenuto, di solito è utilizzato per studi longitudinali: avvengono in un periodo di tempo non illimitato, come diari,...
  C'è però distrazione e rumore che possono risultare come **svantaggi**.

### 6.1.2 Metodi di osservazione

- **Think Aloud - Pensare ad alta voce**: il designer dà in mano all'utente il prototipo e gli chiede di descrivere cosa sta facendo e perché, cosa pensa stia succedendo ecc.
  Ad esempio scorro una lista e dico "ma come sono ordinati i prodotti?", eccetera.
  Ci sono sempre i 3 attori: amministratore, observator, human computer,...
  É semplice e può fornire informazioni utili, ma soggettivo dal punto di vista dell'utilizzatore e osservatore.

- **Cooperative evaluation - Valutazione cooperativa**: sia l'utente che il valutatore/sperimentatore possono porsi domande a vicenda. Ad esempio valutatore chiede all'utente "perchè sei tornato indietro?", "volevi veramente tornare indietro?",...
  É meno vincolato e più facile da usare e l'utente è incoraggiato a criticare il sistema.

- **Protocol analysis - Analisi del protocollo**: viene fatta registrazione della sessione di valutazione in carta e matita o audio o video o registrazione del computer oppure taccuini dell'utente.

- **Post-task walkthroughs - Procedure dettagliate post-attività**: si fa la registrazione o trascrizione di quello che è successo e dopo si riguarda insieme (utente e valutatore) dopo l'attività, chiedendo perchè ha fatto una determinata cosa, eccetera.
  É utile per individuare le ragioni delle azioni e le alternative considerate.
  Viene utilizzato nei casi in cui non è possibile pensare ad alta voce. Ad esempio se ho fatto un app per raccogliere opinioni al cinema, dopo aver visto un film, o durante un concerto.

Questi metodi sono soggettivi e dipendono dalle conoscenze e capacità del valutatore. Il valutatore può essere aiutato anche da altri valutatori e l'obiettivo è riconoscere i problemi e capire cosa fa l'utente.

Ci sono anche degli *esperimenti quantitativi*, **servono a dire numericamente se una cosa è vera o no e fare delle statistiche**.
Quando si lavora in azienda si dovrebbero utilizzare queste tecniche, ma di solito non vengono fatte per mancanza di risorse, sopratutto tempo.

## 6.2 Valutazione sperimentali quantitative.

La *valutazione sperimentale* è **controllata da alcuni aspetti specifici del comportamento interattivo** come ad esempio grandezza del font, colore, ...
Il valutatore sceglie l'ipotesi da verificare.
Per l'ipotesi si cambiano alcuni aspetti sull'interfaccia e si vede se l'interazione migliora.

Bisogna far attenzione anche a delle condizioni sperimentali che cambiano con il valore di alcune variabili controllate.

### 6.2.1 Fattori sperimentali

I fattori sperimentali sono:

- *i soggetti (partecipanti);*
- *le variabili che sono delle cose da modificare (label del widget, del bottone, del menu, ...) e da misurare (se la label è meglio con l'icona o la scritta);*
- *l'ipotesi che è cosa vorrei mostrare;*
- *il design sperimentale che è come lo farò.*

### 6.2.2 Variabili.

Ci sono 2 tipi di variabili:

- **variabili indipendenti da modificare** sono una modifica sulle condizioni dell'esperimento come stile dell'interfaccia, ...
  Ad esempio se faccio un ipotesi per vedere se è meglio un app con il font 15 o con il font 20, la variabile indipendente è la dimensione del carattere. Nel primo variabile:15, nel secondo variabile:20.
- **variabili dipendenti da misurare** è ciò che suppongo che sia influenzato dalle variabili indipendenti .
  Nell'ipotesi di prima vado a misurare in numero di secondi come gli utenti svolgono il task x e la variabile indipendente è il tempo. Minore il tempo è, meglio è.

### 6.2.3 Ipotesi.

L'**ipotesi** è la *previsione che un cambiamento della variabile indipendente causerà una differenza nella variabile dipendente.*
Per esempio. "il tasso di errore aumenterà al diminuire della dimensione del carattere".
L'ipotesi potrebbe essere anche nulla : **cioè non ci saranno variazioni delle variabili dipendenti**.

Ad esempio dò a delle persone un vaccino e ad altre un finto vaccino con finta fisiologica. La mia ipotesi sarà che le persone che hanno ricevuto il vaccino si ammalano di meno rispetto a chi non l'ha ricevuto. L'ipotesi nulla è che non ci sarà differenza tra chi riceve il vaccino e chi non lo riceve in termini di numeri.

Quando si fa una valutazione experimental design si deve innazitutto scegliere l'ipotesi, variabili indipendenti e dipendenti, scegliere i partecipanti e il meotodo sperimentale: between subjects o within subjects.

### 6.2.4 Condizioni di controllo e sperimentali

Quando confronto due o più setup ci saranno diverse condizioni:

- **condizione di controllo**: è la condizione di partenza.
- **condizione sperimentale**: solo una variabile indipendente è modificata rispetto alla condizione di controllo.

É possibile avere diverse condizioni sperimentali che si distinguono da quella di controllo per una variabile indipendente.

### 6.2.5 Metodi sperimentali.

I test possono essere assegnati in maniera differente:

- **between subjects**: a ogni partecipante viene assegnata solo una delle condizioni e quindi ogni partecipante esegui soltanto un test.
  Per questo è necessario avere più partecipanti ed è possibile che le diverse valutazioni saranno diverse.
  Il vantaggio di questa tecnica è dato dal fatto che non risente del trasferimento dell'apprendimento.
- **within subjects**: a ogni partecipante vengono assegnate due o più condizioni, quindi il partecipante esegue il test due o più volte.
  In questo caso non ci sarà esigenza di molte persone e le valutazioni saranno sempre simili poichè stessi partecipanti. Lo svantaggio è che questa tecnica risente del trasferimento dell'apprendimento.

La metodologia più utilizzata è la within subjects e per questo c'è un meotodo per risolvere lo svantaggio di questo metodo.

Per mitigare il trasferimento dell'apprendimento cambiamo l'ordine delle due condizioni per confondere i partecipanti.
Metà dei partecipanti esegue il test dapprima sotto la condizione di controllo, poi sotto la condizione sperimentale, l'altra metà esegue il test dapprima sotto la condizione sperimentale, poi sotto la condizione di controllo.

### 6.2.6 Analisi dei dati.

Con questa valutazione può avvenire **un'analisi statistica dei dati**, che possono essere rappresentati su un grafico e si può ricercare un outliers, che sono valori fuori dalla scala normale.

Ci sono variabili discrete (valori presi da un insieme: di solito quelle indipendenti) o continue positive (maggiori di 0) e variabili indipendenti tipicamente discrete.

## 6.3 Valutazioni basate su esperti: EXPERT BASED.

Invece di andare a cercare utenti specifici per fare valutazioni qualitative e quantitative prendo degli esperti e faccio delle valutazioni di tipo expert based.

Queste valutazioni le posso fare in loop finchè non sono soddisfatto, prima le faccio e meglio (possibilmente nel paper prototyping) è poichè prima correggo l'errore. Inoltre sono vantaggiose perchè non ho bisogno di utenti.

Uno o più esperti valutano l'impatto di un dato design sull'"utente tipico".
**Con questa valutazione voglio valutare non i dettagli delle interfacce, ma errori grandi che violano dei principi, per individuare aspetti che possono causare difficoltà dato che violano noti principi cognitivi.**

Queste valutazioni sono vantaggiose perchè sono poco costose, non richiedono utenti specifici e possono essere fatte in qualsiasi momento.
Sono svantaggiose poichè non vanno a valutare i dettagli dell'interfaccia ma solamenti se i sistemi rispettano l'aderenza ai principi conosciuti (linee guida).

### 6.3.1 Tecniche di valutazione:

- **Cognitive walkthrough**: solitamente viene eseguita da esperti in psicologia cognitiva e *valuta il design in base a quanto è facile per l'utente imparare a svolgere un task.*
  L'esperto cammina attraverso il design del prototipo per cercare dei problemi.

  Per svolgere questa tecnica sono necessari la descrizione del task, dei passi per svolgere il task, dell'utente specifico del sistema, del contesto di utilizzazione e delle specifiche del sistema.
  A ogni passo del compito da svolgere l'esperto si farà delle domande:

  - l'effetto dell'azione (in questo passo) è lo stesso dell'obiettivo dell'utente?
  - l'utente può vedere che l'azione è disponibile?
  - l'utente capirà che quella è l'azione da eseguire?
  - dopo averla eseguita, capirà il feedback ricevuto?

- **Heuristic Evaluation**: 10 euristiche per scoprire problemi di usabilità dei sistemi interattivi.
  Viene utilizzato per guidare le decisioni di progetto o per criticare decisioni già prese.
  Le 10 euristiche sono:

  - Visibilità dello stato del sistema.
  - Nessuna barriera linguistica tra il sistema e il mondo reale.
  - Libertà e controllo da parte degli utenti.
  - Coerenza e standard.
  - Prevenzione degli errori.
  - Riconoscere piuttosto che ricordare.
  - Flessibilità ed efficienza d'uso.
  - Design minimalista ed estetico.
  - Aiutare gli utenti a riconoscere, diagnosticare e correggere gli errori.
  - Guida e documentazione.

- **Model-based evaluation**: Ci sono già dei *modelli esistenti che permettono di valutare l'interazione*: GOMS (Goals, Operators, Methods, Selections), KLM (Keystroke-Level Model), Dialog models.

- **Review based evaluation** invece, è un *altro metodo di valutazione che si basa su risultati sperimentali e prove empiriche della letteratura*, in modo tale da rifiutare o accettare parte della progettazione dell'interfaccia.

# 7 Design e designer.

Vogliamo **l'utente al centro di tutt**o: design, progettazione e realizzazione.
Questo non succede quasi mai nella realtà, perchè di solito il committente e il cliente non sono utenti tipici di un sistema a differenza degli stakeholders che sono persone che beneficiano dell'app o del sistema.
CHIAVE DEL DESIGN: lasciare tutto in mano dell'utente.
Fare design significa progettare. Ci saranno obiettivi, vincoli e compromessi.

Per gli **obiettivi** ci chiediamo per quale scopo stiamo progettando questo sistema interattivo? Per chi stiamo progettando? Sapere quali sono gli utenti del sistema è molto importante e perché i nostri utenti vogliono il nostro design?

Per i **vincoli** ci chiediamo quali materiali/standard/framework,platform(iOs, Android, ...) utilizzare.
Ci facciamo delle domande sui costi e sui tempi di implementazione.

Per i **compromessi** dobbiamo acettare di dover accettare compromessi. Evitare di progettare per un solo goal e poi cercare di far "funzionare" il resto.

## 7.1 Golden rule of design.

Le regole d'oro del design consistono nel **comprendere i materiali che utilizziamo**, come i computer: bisogna comprendere la loro potenza, ma anche i limiti ed inoltre dobbiamo comprendere le persone e le loro interazioni. Dobbiamo renderci conto che gli utenti fanno degli errori e lo dobbiamo tenere in considerazione. Dato che l'utente è al centro di tutto bisogna conoscere gli utenti attraverso il need finding.

## 7.2 Usabilità di un sistema.

*L'usabilità è il "grado in cui un prodotto o un sistema può essere utilizzato da utenti specifici per raggiungere determinati obiettivi con efficacia, efficienza e soddisfazione in un determinato contesto di utilizzo"*

L'obiettivo dell'interaction design è progettare per la massima usabilità in tutte le fasi del ciclo di vita .

### 7.2.1 Sub-characteristics of usabilty.

- Appropriatezza riconoscibilità - gli utenti riconoscono se un sistema è adatto alle loro esigenze.
- Apprendibilità - un prodotto può essere utilizzato da utenti specifici per raggiungere obiettivi.
- Operabilità - un prodotto/sistema ha attributi che lo rendono facile da usare e controllare.
- Protezione dagli errori dell'utente: grado in cui un sistema protegge gli utenti dal commettere errori.
- Estetica dell'interfaccia utente: grado in cui un'interfaccia utente consente un'interazione piacevole.
- Accessibilità - un prodotto/sistema può essere utilizzato da persone con caratteristiche.

## 7.3 Designer e artista.

Il designer non è un artista, poichè un artista fa per il suo piacere personale e non per gli altri.
**Il designer mette insieme i bisogni degli utenti** (s*ceglie i materiali e le tecniche, sperimenta, fa interviste, questionari, tiiene conto della componente psicologica e tiene conto dei costi*) e produce qualcosa che non interessa a lui, ma agli altri. Il designer lavora per gli altri, mentre l'artista lavora per sè stesso e realizza qualcosa soltanto per le sue esigenze.

# 8 Progetto e sviluppo di sistemi interattivi.

Vediamo alcuni modelli per la progettazione del software.

## 8.1 The Waterfall model - il modello a cascata.

Il **modello a cascata** è una metodologia di sviluppo che è nata 50 anni fa (negli anni 70') ed è tutt'ora in uso.

Il modello a cascata si basa su una serie sequenziale di passi:

- **Specifiche dei requisiti**: il progettista e il cliente cercano di catturare ciò che ci si aspetta che il sistema fornisca, che può essere espresso in linguaggio naturale o in un linguaggio più preciso.
- **Progettazione architettonica**: descrizione di alto livello del modo in cui il sistema fornirà i servizi richiesti, suddivisione componenti principali e legami.
- **Progettazione dettagliata**: perfezionamento dei componenti architettonici e delle interrelazioni.
- **Programmazione e testing.**
- **Integrazione e testing**.
- **Operazioni e mantenimento**.

Questa metodologia non è funzionante tanto che anche l'inventore disse che questa era solo un'idea iniziale da ottimizzare.
Questa metodologia di sviluppo è nata per l'ingegneria civile e non per lo sviluppo software.

Per quanto riguarda l'ingegneria civile c'è una fase di progettazione in cui il design è difficile da prevedere e richiede persone costose.
La fase successiva è quella della creazione facile da prevedere in termini di tempo, ma richiede persone molto abili manualmente. Molto più grande sia in termini di costi che di tempo rispetto alla progettazione e alla pianificazione è la fase di creazione.

Per lo sviluppo software il tempo e i costi alti sono dovuti alla fase di progettazione.

La waterfall model spesso fallisce per vari motivi:

- Dato che la fase di progettazione richiede molto tempo, in questa fase soltanto i designer lavorano, mentre i programmatori devono aspettare.
- Viene separata la progettazione dallo sviluppo e quindi fallisce perché le specifiche di progettazione cambiano/o vengono introdotte durante lo sviluppo per vari motivi:
  - cambiamento di contesto;
  - migliore comprensione del problema o nuove idee;
  - nuovi requisiti aziendali;
  - scoperta di un uso inaspettato;
  - l'evoluzione della tecnologia;

  In questo caso dato che la **waterfall model** è una metodologia **predittiva** piuttosto che adattiva c'è un continuo tornare indietro nelle diverse fasi creando un *dialogo scoordinato*.

Il progetto software è meglio dunque se è di tipo iterativo (come il prossimo modello che vedremo) piuttosto che di tipo sequenziale.

## 8.2 Progettazione iterativa.

La **progettazione iterativa** supera i problemi legati all'incompletezza dei requisiti. (Cliente che introduce nuovi requisiti nel mezzo)

Ci sono dei prototipi durante le varie iterazione che simulano alcune caratteristiche del sistema finale. Questi prototipi possono essere: throw away, incremental e evolutive.

I problemi di gestioni (tempo, caratteristiche non funzionali, contratti e pianificazione) vengono superati.

## 8.3 Agile.

**Agile** è una metodologia di sviluppo che è basata su alcuni aspetti fondamentali:

- ispezioni frequenti del codice, del prototipo, ...
- l'adattamento
- il lavoro di squadra, l'auto-organizzazione.
- la consegna rapida di software di alta qualità.
- allineare lo sviluppo alle esigenze del cliente, noi in questo caso parliamo di user. (cliente ed user sono differenti, l'user ha dei bisogni, . . . )

L'Agile "spezzetta" in più cicli quello che di solito viene fatto in un unico ciclo.
Quindi le fasi di ***Analisi, Progettazione, Programmazione, Testing e Rilascio*** vengono svolte solo per un piccolo gruppo di funzionalità, per poi essere ripetute per i cicli che seguono. Ognuna di queste fasi è chiamato nella terminologia Agile: **sprint**.

Altre caratteristiche del metodo agile sono:

- **Pair Design Programming**, ovvero una metodologia in cui si programma in due, alternandosi. (uno vede gli errori dell'altro ed eventualmente li corregge)
- **Close Proximity Teams**, dove Team diversi lavorano vicini (fisicamente parlando).

### 8.3.1 Differenze con Agile.

A differenza del modello a cascata, agile è una tecnologia di tipo **adattivo**, cioè accoglie il cambiamento (introduzione cambiamenti e nuovi requisiti) anche a costo di cambiare se stessa.
Un'altra differenza consiste nel fatto che agile è orientato alle persone piuttosto che ai processi, e quindi è importante avere un ambiente sereno nel team con persone che non sono sostituibili.

## 8.4 User-Centered Design: UCD.

Esiste anche un altro modello iterativo abbastanza vicino all'Agile, ovvero il **modello UCD** che è decisamente *più vicino all'utente* e cerca di capire di cosa l'utente ha bisogno.

In questo modello non deve essere l'utente ad adattarsi al design del prodotto, ma il design del prodotto deve essere coerente all'utilizzo dell'utente.

Da solo l'UCD non gode delle caratteristiche importanti dell'agile come gli sprint, i team, i pair programmaing e pair design,... ed è proprio per questo che nasce il metodo Agile UCD.

## 8.5 Agile UCD.

Questo **nuovo modello "ibrido"**, ha dunque il vantaggio che riesce a colmare le principali lacune dei due metodi, come l'early release (per l'UCD) e la vicinanza all'utenza (per l'Agile).

Per completezza, è bene mostrare allora tutte le fasi principali di questo nuovo metodo:

- ***Ricerca e analisi***: la fase di Ricerca e analisi è fondamentale per la buona riuscita di un progetto. Oltre a definire gli obiettivi aziendali e gli scopi dell'applicazione, si pianificano le attività di ricerca per acquisire informazioni attraverso. interviste all'interno dell'azienda.

- **Sprint 0**: ***sviluppo dei personaggi, degli scenari e delle stories***: servendoci dei risultati delle ricerche possiamo, nello sprint 0, definire i personaggi e gli scenari che ci porteranno a individuare le principali stories. Durante lo sprint 0 il team di programmazione, o almeno un suo rappresentante, deve essere coinvolto nello sviluppo dei personaggi e degli scenari.

- **Sprint 1** ***Architettura informativa, task, wireframe e interfaccia grafica***: tipicamente, le stories più importanti dello sprint 1 sono quelle che portano allo sviluppo dell'architettura informativa generale, a individuare i task principali, alla progettazione dei wireframe generali e al progetto grafico dell'interfaccia.

- **Dallo sprint 2 allo sprint (N)** ***produzione***: nella fase di produzione vengono individuate stories che, di volta in volta, porstano allo sviluppo di determinate sezioni del sito web oppure a determinati task dell'applicativo web (una procedura di registrazione, per esempio).

- **Sprint (N)+1** ***pubblicazione***: le attività di questo sprint sono essenzialmente di revisione e controllo.

- **Sprint di** ***ottimizzazione e modifica*** Se il progetto lo prevede si potranno identificare degli sprint in cui inserire le attività di ricerca più idonee per valutare l'efficacia, l'efficenza e la soddisfazione d'uso.

Noi nel progetto abbiamo utilizzato Agile UCD grazie al need finding, al prototyping, ...

Per svolgere meglio il lavoro si fanno dei **meeting giornalieri/inizio-fine sprints** molto brevi e si discute cosa fare quel giorno stesso.

Si preferisce ad avere sempre più *feedback su utenti via testing.* Se sto facendo app sulla prenotazione di aula, vado dai studenti e non da chi gestisce l'app.

Bisogna suddividere il design rispetto a quello che devo fare. L'iterazione è fissa, una settimana, se il progetto è grande, farò una funzione alla volta.

# 9 Luogo dell'attenzione e principi.

Quando siamo attivi e coscienti **il luogo dell'attenzione** è *un idea o un pensiero alla quale stiamo pensando attivamente intenzionalmente (concentra la nostra attenzione).*

## 9.1 Focus and Locus

Il **focus** è la volontà di focalizzare l'attenzione su qualcosa e potrebbe essere un oggetto o un elemento dell'interfaccia.
Il focus può essere attivo in un certo istante.

Il **locus** è dato dal fatto che noi non possiamo controllare attivamente quale sarà il nostro luogo dell'attenzione poichè abbiamo sempre l'attenzione su qualcosa e questo non potrebbe coincidere con il focus dell'interfaccia.
Potrebbe essere attivo o passivo.

## 9.2 Unico luogo dell'attenzione e come spostarlo.

Possiamo concentrarci solo su una delle cose che percepiamo contemporaneamente, quindi le applicazioni devono porre l'attenzione soltanto su una cosa e non su troppe cose altrimenti potrebbero fallire.

Quando abbiamo l'attenzione su qualcosa, quindi diciamo che siamo assorti su una cosa non si presta attenzione alle altre cose che accadono.

Solamento un evento che potrebbe essere interno o esterno potrebbe spostare il luogo dell'attenzione.
Vengono fatte **delle scansioni dell'ambiente** e quando siamo assorti su qualcosa la scansione viene fatta di meno.

## 9.3 Compiti automatici.

Compiti che eseguiamo senza pensiero cosciente sono detti compiti automatici.

**Possiamo eseguire più compiti automatici insieme, ma non possiamo eseguire più compiti automatici e non automatici** (cioè quelli che riguardano il luogo dell'attenzione) ad esempio il camminare mangiando (automatico) pensando alla soluzione di un problema di matematica (non automatico).
Cercare di eseguire contemporaneamente due compiti non automatici fa peggiorare il rendimento in entrambi.

## 9.4 Abitudini.

Compiti effettuati rapidamente creano un'abitudine che permette di eseguirli senza pensarci.

Soffermare il pensiero sui dettagli dell'esecuzione può rendere impossibile eseguire tali compiti.
Risulta molto complicato interrompere un'abitudine.
**Interrompere una sequenza automatica richiede lo spostamente del luogo dell'attenzione su di essa** (attraversare la strada è il task, la sequenza di azioni è: guardo a dx, guardo a sx,...)

### 9.4.1 Abitudini con interfacce.

Dobbiamo tener conto e sfruttare il fatto che gli utenti tendono a prendere delle abitudini anche quando utilizzano le interfacce (CTRL+C, CTRL+V, sbloccare il cellulare, aprire le applicazioni , ...)
É possibile eseguire azioni automatiche in contemporanea con altre azioni.

### 9.4.2 Errori per abitudini.

*La formazione di abitudini potrebbe aiutare l'utente* ma potrebbe creare anche alcuni problemi.
Ad esempio nella cancellazione di un file è di solito presente un dialog che ci chiede se siamo sicuri di eliminare un determinato file. Dato che siamo abituati a confermare questo è possibile che ci sbagliamo ed eliminiamo un file importante.

L'interfaccia dovrebbe permettere la concentrazione sul task, dato che la concentrazione aumenta con lo stress.
Se l'esecuzione del task è difficile o delicata, l'utente tende a concentrarsi solo su alcuni aspetti essenziali, ignorando warning e help forniti dall'interfaccia.

### 9.4.3 Come sfruttare abitudini nelle interfacce.

Noi non sappiamo cosa sta guardando l'utente e se questo potrebbe avere attenzione su un altra cosa. Dal punto di vista di interfaccia si cerca di fare quello che si può, si può ipotizzare dove l'utente ha il luogo dell'attenzione e apportare cambiamenti in altre parti del sistema, senza distrarre l'utente, cioè le cose meno importanti le metto dove l'utente non ha luoghi dell'attenzione.

Il luogo dell'attenzione ne è solo uno, in un interfaccia potrebbe essere anche fuori. Se l'utente è concentrato nell'interfaccia sarà fissato solo su un luogo dell'attenzione e per questo si cercherà di ridurre l'attenzione da altre parti dell'interfaccia.

## 9.5 Cambio di contesto.

Potrebbe essere possibile che l'utente voglia passare da una schermata di un app ad un altra, avviare il telefono e usarlo.
Ci sono 2 cambi di contesto: il nostro (dell'utente) e quello del dispositivo .
Generalmente un cambio di contesto richiede 10 secondi, se questo diventa abituale potrebbe richiede meno tempo.

**Bisogna ridurre il tempo di cambio di contesto e aiutare l'utente**. Per far questo, ad esempio nel passare da un app all'altra si dovrebbe vedere una miniatura della schermata dello stato attuale di quell'app.
Per ridurre il tempo di cambio di contesto ci sono alcuni accorgimenti:

- Tornare all'ultimo task che era stato interrotto prima di averlo completato;
- L'interfaccia può aiutare l'utente a ritornare al precedente luogo dell'attenzione, mostrando la stessa schermata che si era lasciata, e posizionando il cursore nello stesso posto dove si trovava: punto di un documento, pagina web interna in un sito, etc.

Ad esempio alcuni aspetti positivi sono:

- *Firefox che quando viene lanciato riapre tutte le finestre dell'ultima sessione. Chrome carica tutte le schede, Firefox quella dove si stava.*
- *L'anteprima del mac riapre un pdf sull'ultima pagina visualizzata.*
- *Il sistema operativo riapre le finestre della directory aperte nell'ultima sessione.*
- *Sistema in stato di 'stop' anziche' spento.*
- *Autoradio riprende il cd dal punto in cui si era interrotto.*
- *Netflix, prime video, che ci ricordano a che punto della puntata ci troviamo.*

# 10 Modi e interfacce modali.

*Tutti gli utenti prendono delle abitudini involontariamente quando eseguiamo sequenze di azioni senza pensiero.*
Queste abitudini possono aituare l'utente, ma possono diventare anche una fonte di errore.
Infatti potenziali problemi nascono dalle sequenze di azioni.
Effettivamente la conferma y/n per eliminare qualcosa non è pericoloso, ma lo potrebbe essere quando sto inviando soldi, ... (quando svolgo task critico).

## 10.1 Terminologia.

- **Content**: l'insieme delle informazioni che risiede in un sistema e che sono importanti per l'utente, esempio: interfaccia grafica.
- **GID**: meccanismo per comunicare al sistema una particolare locazione o scelta di un oggetto come la locazione del cursore, freccetta,...
- **GID bottone**: bottone principale del GID, che ritorna nello stato iniziale una volta attivato. (click del mouse, del touch screen, ...)
- **Tap**: premere e rilasciare un testo
- **to click**: posizionare il GID e utilizzare il GID bottone.
- **to drag**: premere il GID bottone, senza rilasciarlo muovere il GID, e poi rilasciarlo (in un altra locazione). Tipica azione si spostare file nel cestino, ...

## 10.2 Gesti.

Un gesto è una sequenza di azioni umane che viene completata automaticamente una volta avviata.
Nel touch screen, i gesture hanno preso un importanza notevole e sono ormai noti a chiunque. Senza di questi infatti, non si potrebbe nemmeno "comunicare" con il dispositivo in alcun modo, vista l'assenza di opportuni tasti fisici.

## 10.3 Modi.

*Dato un gesto, un interfaccia è in un modo se l'interpretazione di quel gesto è sempre la stessa, se il gesto è interpetato in maniera diversa, l'interfaccia si trova in un altro modo.*

Se sto dentro un file word, selezionare Ctrl+x è un gesto, l'esito è sempre lo stesso. (modo: visuale lettura)
Il tasto return può essere interpretato come 'andare a capo' o come 'confermare l'inserimento'.

Quasi tutte le applicazioni hanno modi, l'avere modi può aiutare l'utente ma può creare anche disorientamento.

Ad esempio se si considera una sveglia con un bottone che serve ad attivare e disattivare la sveglia a seconda del fatto che la sveglia è attiva o disattiva.
Se siamo al buio, non si può stabilire l'effetto dell'azione poichè non si vede il display.

## 10.4 Interfaccia modale.

**Un interfaccia modale non permette di stabilire quale operazione si debba effettuare per raggiungere un dato risultato solamente ispezionando il meccanismo di controllo, è perciò necessario verificare separatamente lo stato del sistema.**

Un interfaccia può avere i modi, ma se ha i modi non è detto che è modale. Se ha i modi e non sussistono entrambi le condizioni (quelle dette prima) allora non è modale.

*Ad esempio un interfaccia modale potrebbe essere la sveglia descritta prima, l'interruttore (doppio) della luce, il freno a mano delle nuove automobili. Un altro esempio potrebbe essere quella del CAPS LOCK che crea modi che non si possono verificare solamente ispezionando il meccanismo di controllo.*
Le persone continuano con questo errore perchè lo stato del sistema non si trovano nel luogo dell'attenzione.

## 10.5 Minimizzare i 'mode errors'.

**Non avere i modi**, **assicurarsi che i comandi richiesti da differenti modi non sono gli stessi (ogni modo attivato da un differente shortcut)** e **rendere evidente che ci troviamo in un modo**.
La perfetta non sovrapposizione fra i gesti di modi diversi comporta che non ci siano più modi.

Un'interfaccia è modale rispetto a un dato gesto quando:

- lo stato corrente dell'interfaccia non è il luogo dell'attenzione dell'utente e
- l'interfaccia risponderà al gesto con una tra n possibili risposte, a seconda dello stato.

L'interfaccia è modale se ha i modi e se non riesco a capire in che modo sta senza esaminare esternamente.

## 10.6 Quasi modi e modi temporanei.

- **Modo temporaneo**: un modo che svanisce dopo l'uso ( *ad esempio: pennello di word, attivato con un solo click).*
- **Quasi-modo**: un modo ottenuto attivando e mantenendo fisicamente un controllo: *es. tasti maiuscole, tasto CTRL + .. (combinazioni)*

## 10.7 Noun-verb and verb-noun.

Eseguire azioni (verb) su oggetti (noun).

- **Noun-verb**: si seleziona prima l'oggetto e poi l'azione da effettuare. *Seleziono il testo (prima l'oggetto) e modifico.*
- **Verb-noun**: si seleziona prima l'azione e poi l'oggetto da modificare. *Seleziono la gomma su photoshop e dopo cancello. Questo introduce un modo.*

**Lo stile verb-noun crea un modo.** Per evitare modi, privilegiamo lo stile noun-verb. Nello stile noun-verb:

- Si seleziona l'oggetto mentre esso è nel luogo dell'attenzione, quest'ultimo poo si sposta sull'azione da compiere e a quel punto si attiva il gesto che esegue l'azione.
- Interrompere l'azione non richiede un'altra azione (cancel o escape).

# 11 Progetto di applicazioni mobile.

Andiamo a vedere le applicazioni per i dispositivi mobile come smartphone, tablet, smartwatch, ...

## 11.1 Contesto d'uso.

Le ***applicazioni mobili*** sono *differenti rispetto a quelle del computer* poichè hanno c**ontesti di uso diverso, cambia il posto (contesto fisico) in cui uso l'app e il tempo (contesto temporale).**

Un utente può utilizzare un dispositico mobile anche in movimento, eseguendo dei compiti in tempi brevi e di fretta. L'utente può essere interrotto a causa di eventi esterni, ...

Un dispositivo mobile può essere utilizzato in contemporanea ad altre azioni, ad esempio utilizzo un cellulare con una mano perchè con l'altra sto tenendo la spesa, ...

## 11.2 Differenze rispetto al desktop.

### 11.2.1 Schermo piccolo.

A differenza del desktop un dispositivo mobile ha uno scermo più piccolo, questo comporta **minor leggibilità, maggior risoluzione e poco spazio per inserire elementi non utili all'interfaccia.**

### 11.2.2 Memoria limitata.

Nei dispositivi vecchi, oltre a problemi di schermo e risoluzione, c'era anche il problema della memoria, visto che non sono molto capienti sotto questo aspetto. (anche la RAM, CPU etc sono quello che sono.)
Questa problema dunque, ha portato ad avere sui dispositivi mobili file con le dimensioni ridotte al minimo indispensabile. (Cosa ovviamente non vera su PC.) Tuttavia tutto ciò si avverte sempre di meno, viste le crescenti capacità dei smartphone attuali.

### 11.2.3 Multitasking e schermate.

Nei dispositivi mobile non è presente il multitasking. *L'utente può accedere alle schermate sequenzialmente dato che non ci sono finestre.*
Un'applicazione potrebbe interrompere qualsiasi altra app, come quella delle telefonate e in questo caso, una volta effettuata la telefonata, si dovrebbe ritornare all'app precedente con tutti i dati salvati e dallo stesso stato in cui si trovava.

### 11.2.4 Help minimale.

Gli utenti di un dispositivo mobile non hanno tempo di leggere l'help poichè quest'ultimo può togliere spazio sullo schermo.
Proprio per questo bisogna usare controlli standard: le app mobile devono essere il più possibile standardizzate (gli utenti già li conoscono)
Dotare le schermate di bottoni back per tornare indietro. Deve essere sempre fatto questo.

# 12 Stili di applicazioni mobili.

Ci sono *3 tipi più comuni di applicazioni mobili.* La maggior parte delle applicazioni ricadono in un tipo o mix.

## 12.1 PRODUCTIVITY APPLICATION.

Coprono la maggioranza.
Questa tipologia di applicazione mobile **permette di svolgere compiti basati sulla organizzazione e manipolazione di informazioni dettagliate**, come mail, gestione foto, messaggistica, social media, ...
User experience focalizzata sul task, niente di bello da vedere, meglio un design minimal.

Ci sono più schermate organizzate **gerarchicamente**.
Un interazione tipica si compone delle seguenti azioni:

- Organizzione a lista o albero.
- Aggiungere e togliere elementi dalla lista. A uno o più livelli ci sono delle liste.
- Scendere a livelli di dettaglio successivi e poi eseguire operazioni sul livello scelto.

Ci sono alcuni standard:

- Una vista (schermata) per ogni livello con un interfaccia semplice e pulita (focalizzata su task).
- Uso di controlli standard.
- Enfasi sulle informazioni e sul task, non sull'ambiente o sull'esperienza.
- Impostazione di preferenze per ridurre informazioni e scelte ripetitive

## 12.2 UTILITY APPLICATION.

Le utility application sono delle app con dei task semplici che richiedono poco inpit dall'utente. **Vengono utilizzate per leggere informazioni essenziali su qualche argomento e verificare lo stato di qualcosa**.
Ci sono poche schermate con poche informazioni senza struttura gerarchica.
Le preferenze sono soggette a frequenti cambiamenti alla configurazione.
*Ad esempio app per le previsioni meteo, controllo telecamere, ...*

## 12.3 IMMERSIVE APPLICATION.

Le immersive application sono utilizzate per **compiti che presentano un ambiente particolare e non mostrano grandi quantità di testo.** Richiedono l'attenzione continuata dell'utente che non può distrarsi. In un gioco può mettere in pausa in gioco, ma rientrando si troverebbe male.
*Fanno parte di questa tipologia ad esempio i giochi.*

Le caratteristiche di queste applicazioni sono: **ambienti ricchi di suoni, a tutto schermo e focalizzate sul contenuto o esperienza di quel contenuto**.
Un'immersive app non usa i controlli standard, cercare i controlli e scoprirne il funzionamento è parte dell'esperienza di un'immersive application.
Usa spesso grandi quantità di informazione, ma la mostra in un modo particolare legato al contesto.

Esempi di queste applicazioni possono essere :

- La *livella* che fa parte di questa categoria perchè mentre controllo non posso distrarmi quando la utilizzo.
- *Stellarium* (si vedono stelle e costellazioni), *app per realtà virtuale, youtube*, *netflix*, ...

Nella lezione 18, il professore fa notare che su classroom i controlli per l'eliminazione dei commenti sono divisi in due: uno per l'utente stesso e uno per gli altri utenti.
Ad esempio se io voglio eliminare un mio commento troverò la scritta delete al secondo posto: mentre se voglio eliminare i commenti di altri lo trovo al primo posto.
Questa cosa andrebbe evitata, c'è un errore di automatismo, essendo abituato con gli altri a eliminare all'inizio, con me stesso mi confondo.

# 13 Android: criteri.

Vediamo alcuni prinicipi da tenere in considerazione quando creiamo delle interfacce. In questo caso parliamo del sistema operativo Android.

## 13.1 LEVA SULLA CONOSCENZA DEGLI UTENTI.

### 13.1.1 STANDARD WIDGETS.

Sono tutti quei *componenti grafici* che per via del loro utilizzo, sono **diventati dei veri e propri standard**. Bisogna cercare di sfruttare quello che l'utente già sa, usare widgets di android. *Ad esempio la barra sotto per cambiare schermata e tornare indietro, ultimo dialog modale e per la ricerca, mappa, pallino per trovare la posizione attuale.*

### 13.1.2 STANDARD GESTURES.

I "gesti" effettuati sull'interfaccia, che ormai sono consolidati. *In questo caso parliamo di touch singolo, doppio, swipe, drap, scroll, ...*
*Parliamo anche di zoom in, zoom out, ...*

Ci sono anche delle metafore che si usano per descrivere qualcosa che si conosce dalla vita reale, e sarà per questo di facile comprensione.
Ad esempio parliamo dello switch, della lente di ingrandimento per cercare qualcosa, icona del segnalibro.

## 13.2 DISCOVERABILITY.

**Le cose sull'interfaccia devono essere facili da scoprire e non impossibile da trovare.**
Se c'è infatti qualcosa che non si vede nella nostra applicazione, è importante farla notare in qualche modo all'utente.
Per far questo è vietato **NASCONDERE I CONTROLLI**, si possono nascondere dei task secondari, solamente se sono standard.

## 13.3 ACTION AND REACTION.

**Qualsiasi operazione effettuata su un interfaccia dovrebbe dar luogo ad una reazione** poichè l'utente deve rendersi conto del fatto che ha effettuato un azione.

*Un esempio è quando ad esempio prendiamo e spostiamo un oggetto.* Quando faccio queste azioni, voglio che l'app risponda in tempo reale!
*Quando selezioniamo una data sul calendario deve evidenziarsi.*

## 13.4 MEMORY.

*Ad esempio quando andiamo nei contatti bisogna far sì che vengano mostrati prima i contatti che chiamo più frequentemente, ...*

## 13.5 USERS IN CONTROL.

**Gli utenti devono sempre sapere dove si trovano e cosa stanno facendo**.
Su Android questo si fa grazie a dei menu , il cui titolo è nella cosidetta Action Bar.
Una action bar è un oggetto presente nella parte superiore di ogni schermata, e che in genere continua ad essere presente in ogni momento in cui usiamo l'app.

**BACK**: Ci deve essere sempre la possibilità di tornare indietro grazie al "back" .
Ci sono due tipi di navigation:

- quello sopra mi porta nella view gerarchicamente precedente. (all'interno dell'app)
- quello sotto mi porta nella view precedente. (all'interno del sistema)

**CANCEL**. Ci deve essere sempre la possibiltà di cancellare qualcosa che si sta facendo e ancora non è stato compiuto.

**PROGRESS BAR**. Si tratta di una barra che è necessaria se l'operazione impiega più di quanto l'utente è tollerante ad aspettare. (in genere un secondo). Vediamone le varie tipologie:

- Determinate: ovvero sò quanto manca.
- Indeterminate: Non sò quanto manca.
- Buffer Tipica nei video.
- Query Determinate e Indeterminate

**UNDO**. Permette di cancellare un operazione dopo averla eseguita .

*La differenza tra cancel e undo è che cancello un operazione prima (cancel) o dopo (undo) averla eseguita.*

### 13.5.1 CUSTOMIZE.

*Ogni utente ripete certe azioni, e queste sono spesso diverse da utente a utente.* Sarebbe **importante quindi raccogliere informazioni dall'uso dell'utente, per apprenderne le preferenze**. (e di conseguenza aiutarlo con dei futuri suggerimenti) ... Quello che si fa per questo campo è:

- suggerire sulle ricerche.
- mettere icone generali, come quelle per la condivisione , per i like,...
- Permette di customizzare qualsiasi cosa all'utente: lo sfondo, la posizione dei widget, ...

### 13.5.2 CONSISTENCY.

**SAME GESTURES, SAME ACTION.**
É importante avere Coerenza in questa cosa, in quanto vogliamo che ***ad una determinata azione corrisponda sempre lo stesso comportamento***. (Stesse fun- zioni, stessi widget)
Inoltre è importante evitare le modalità che potrebbero confondere l'utente.
Utilizzare sempre le stesse funzioni e i stessi widget.

Dare la possibilità all'utente di cambiare la modalità scura o chiara.

## 13.6 MATERIAL DESIGN ANDROID.

**Material Design è un sistema adattabile di linee guida, componenti e strumenti che supportano le migliori pratiche di progettazione dell'interfaccia utente.**
I principi per quel che riguarda il Material Design, (approfondibili Qui) sono essenzialmente 3:

- ***Material is the metaphor***: i "Materiali" sono molto importanti in questo contesto. Questi vengono riproposti nell'interfaccia rispettando i principi della fisica.
  Il fatto che tutti gli element idell'interfaccia hanno una distanza dal fondo dello schermo, mi fanno sembrare che ci sia una terza dimensione.
  Ad esempio l'App bar ha un altezza 4dp, il floating action button ha un altezza 8dp, le card hanno un altezza di 2d.
  Quando scorro una lista di elementi, la lista passa sotto l'app bar e sotto il floating aciton button. Cambiano anche le diverse ombre.
- ***Bold, graphic, intentional***: il modo in cui sono espressi i font, non comportano semplicemente un miglioramento grafico, in quanto creano gerarchie di importanza, focus e significati diversi in base al contesto. Interfacce con elementi senza bordi, con colori che contrastano tra di loro e disegni non realistici.
- ***Motion provides meaning***: se c'è movimento si svolge un azione,...

### 13.6.1 List and Cards.

- ***Le liste, sono Items (omogeni tra di loro) disposti su più linee in una disposizione verticale, che danno l'impressione di essere un elemento unico.***
- Gli elenchi basati su Cards, sono invece dei ***pezzi di "carta" che contengono dati distinti tra di loro e potrebbero essere non sono omogenei tra loro*** (a differenza dei tipi precedenti).

### 13.6.2 Alerts and Bottom sheets.

Gli *alert*, come sappiamo, sono **messaggi di "allarme",** e sono implementati tra- mite pop-up o dialogue box. Sono utilizzati per informare su situazioni critiche e richiedere agli utenti di prendere decisioni .
La maggior parte sono modali e introducono un cambiamento nel significato dei gesti che io faccio sull'interfaccia.

Le *bottom sheets* invece, presentano una lista di azioni (più dettagliate) possibili.
Questi tipi di alert hanno un nome specifico, e vengono chiamati nel gergo Bottom Sheets.
Come abbiamo detto sono "user solicited", in quanto seguono una determinata azione eseguita dall'utente.
Questi sono **modali** perchè in un caso se clicco il contatto si apre il bottom sheets e se da qui clicco sui contatti si toglie il bottom sheets.

# 14 iOS: criteri e strumenti.

In **IOS** c'è *Human Interface Guidelines* che contiene le guide, i irinicipi di funzionamento generali e le miglior pratiche per la progettazioni di interfaccia IOS.

## 14.1 Criteri iOS.

### 14.1.1 Authentication.

**Si tende a posticipare il sign-in il più possibile**: non è ben visto scaricare un'applicazione e non vedere neanche una vista per la registrazione.
Prima di passare al sign-in si spiega il perché dell'app e i benefici una volta scaricata.

### 14.1.2 Data Entry.

- Si vuole **minimizzare il più possibile data entry**: se mi devo registrare utilizzare altre forme come google, facebook, apple id,... oppure posso utilizzare solo i campi obbligatori come email e password.
- Sarebbe **preferibile la selezione alla scrittura** perchè quest'ultimo è noioso e soggetto ad errori.
- C'è anche una caratteristica di **validazione dinamica**: quando si sceglie una password si mette in rosso finchè non va bene per i requisiti. (metti numeri, lettere maiuscole, . . . )
- Inoltre c'è l'**hints**. ( help per l'informazione richiesta )

### 14.1.3 Gestures.

Si devono **utilizzare gesti standard** come *singolo click ,swipe, scroll, ...*
Evitare di associare azioni non standard a gesti standard.

### 14.1.4 Modality.

Le *modality* sono **viste (schermate) che focalizzano l'attenzione dell'utente**.

Essendo **modali** producono dei modi e funzionano solo dalla vista modale che si sovrappone alla vista sotto. Ad esempio con il dialog (che è modale) non si può spostare sulla mappa , . . .
I gesti che funzionano sono quelli del dialog. La vista sottostante è sbiadita e fa capire che è stato introdotto un modo. Questi possono occupare lo schermo in parte o del tutto e mostrano controlli per il completamento o annullamento.
La vista modale è solo una finestra, non può avere ulteriori viste (come android)

### 14.1.5 Navigation.

**Mantenere chiaro il percorso** che può essere *navigato attraverso nav bar, tab bar, page control.*
In IOS la navigazione è particolarmente gerarchica .

### 14.1.6 Branding.

Il *branding* dovrebbe essere **non inavsivo, incorporato nella grafica e coerente in tutta l'app**. Potrebbe essere un logo.

## 14.2 Launch.

*Quando lancio l'applicazione si deve visualizzare qualcosa anche se non si è totalmente caricata*, perciò si deve dare il tempo necessario all'app / all'utente.
Deve riprendere il lavoro interrotto in qualsiasi contesto .

**LAUNCH SCREEN**: quando si avvia l'app ci deve essere ad esempio un logo netflix, disney, primevideo, sempre identico e senza testo.

## 14.3 Strumenti iOS.

### 14.3.1 3D Touch.

Strumento in fase di disattivazione. Introduce una terza dimensione per il tocco. (si va nella profondità dello schermo). Ci sono 2 livelli: un tocco leggero e uno pesante.

Si tratta di un gesto: pressione sul touch screen e in risposta a questo gesto ci potrebbe essere un menu, del contenuto aggiuntivo, una preview, un'animazione, etc.

Quello che si può ottenere con il 3D touch è quello che si ottiene con un long press di Android.
I motivi per cui il 3D Touch non ha mai preso piede è per il fatto che il 3D Touch è stato implementato solo su iOS e quindi questo dà delle difficoltà a rendere standard questo gesto.

Il **quick action** è una *pressione su un'icona in home screen in un click leggero.*
Facendo un click leggere viene mostrato un menu di azioni rapide relative a una particolare applicazione.

### 14.3.2 Peek and Pop.

Distinguiamo il peek e pop.
Il **peek** è l'*azione che si ottiene alla pressione leggera.* (tengo premuto, ottengo vibrazione e viene visualizzata una preview di qualche tipo).
Se *faccio una pressione più forte quello che viene prodotto* è un **pop** (accesso all'item vero e proprio).
Dopo aver fatto il peek posso fare uno swipe verso l'alto per visualizzare le azioni da svolgere.

*Se ho un link posso con un peek avere una preview della pagina su cui andrei. (almeno se non è la pagina che volevo non ho bisogno di tornare indietro)*

### 14.3.3 Live photos.

Effettuando un pop su una foto vedo il **live della foto**.

### 14.3.4 Notifiche.

Sono **locali** o **push**: *Le notifiche locali o push esistono anche in Android.*

Per le **notifiche locali** si intende una notifica generata da un app perchè è avvenuto un particolare evento .

Le **notifiche push** non sono direttamente per l'utente, ma *sono per il dispositivo.*
Le app se vogliono ricevere delle notifiche si registrano al servizio (firebase per android oppure push notification service) per un certo tipo di notifiche e quando una delle informazioni per le quali l'app si è registrata è

disponibile l'app viene notificata (dall'app al dispositivo). Dopo che la notifica avviene l'utente può vedere a sua volta una notifica.

Ad esempio l'app per le previsioni del tempo deve tenere aggiornate le informazioni , ma vorremmo che l'app in background non spreca troppa batteria e troppa banda anche se deve comunque dare le informazioni aggiornate. Per fare questo l'app delle previsioni del tempo si registra al server per farsi mandare delle notifiche quando si hanno le informazioni aggiornate su una città. (firebase o pns)

Questo è un meccanismo ***inverso al client-server***, è il server che quando ha nuove previsioni notifica il client che è l'app. L'app delle previsioni del tempo in questo caso non notifica all'utente, ma altre app come quella di app di messaggistica notificano l'utente. Questo di solito funziona per i widget di determinate applicazioni.

Caratteristiche notifiche:

- Possono avere *titolo, testo, suono e logo app.*
- Possono essere *badge, payload, banner, alert, lock screen.*
- **L'apertura delle notifiche** avviene con un tap sul banner o swipe in lock screen.
- **L'annullamento** attraverso swipe up o attesa sparizione.
- **L'expanded detail view dinamica** avviene con 3D o swipe dow.

#### 14.3.5 Widgets.

Vengono utilizzate per **mostrare informazioni o funzionalità essenziali all'app**.
Possono essere semplicemente campi con testo, immagini, ...
Se ci sono più widget si scelgono diversi nomi.
In Android si possono scegliere e posizionare sullo sfondo dello smartphone, per iOS c'è un opportuna vista.

Questi widget possono approfittare delle notifiche push e devono mantenere le informazioni aggiornate.
Alcuni widget hanno bisogno di notifiche push, altre no (come le app che lavorano in locale (sveglia,..)).
Un altro esempio su cui non utilizzare notifiche push è ad esempio il fatto di avere un app per i passaggi degli autobus. Questo perchè se prendo l'app una volta al giorno carica tutto il giorno le informazioni e sprecherebbe banda e batteria.

#### 14.3.6 Siri.

Tutte le app possono integrarsi con Siri .
Bisogna definire i task supportati, validare l'input, eseguire il task e restituire un feedback.
Siri risponderà rapidamente.
Viene utilizzata Siri per minimizzare l'interazione (richiedere chiarimenti offrendo opzioni per l'utente)

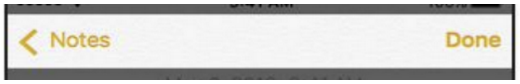
A differenza dell'assistenza google ha un integrazione con le app: tutte le app possono integrarsi con Siri.

## 14.4 Bars, Views and Control.

### 14.4.1 Status bar.


La *status bar* **mostra lo stato corrente**. Viene fornita automaticamente dal sistema operativo e si trova in alto.
Può essere utilizzata anche per mostrare progress bar indeterminato di attività di rete nelle operazioni lunghe.
Può essere nascosta se si gioca a un videogioco.

### 14.4.2 Navigation bar.

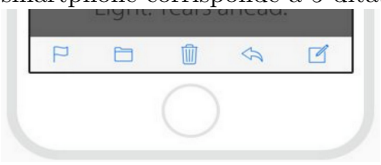



**Costituita da un pulsante per tornare indietro e per eseguire un azione opposta al back**, c'è anche un titolo e potrebbe trattarsi di un segmented control (bottoni che permettono di cambiare la visualizzazione della lista).
Non serve se non si può navigare dalla vista corrente e si può togliere temporaneamente.

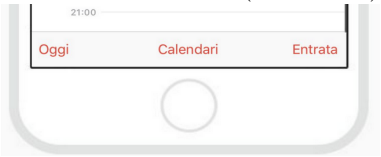
### 14.4.3 Toolbar.

Si trova in basso.
Contiene **bottoni o label per azioni rilevanti nella vista corrente**. Ci sono:

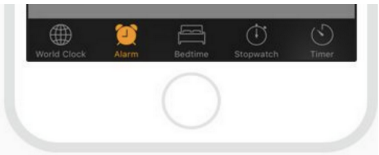
- max 5 icone da scegliere in un insieme predefinito senza didascalia. Sono 5 perchè la larghezza dello smartphone corrisponde a 5 dita.



- max 3 label di testo (dinamiche). Queste sono 3 perchè la label è più larga.

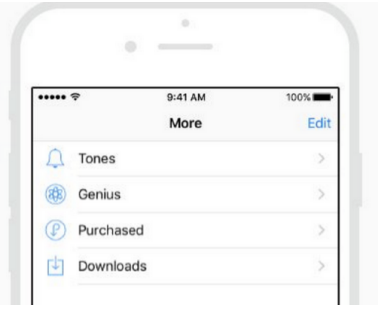



### 14.4.4 Tab bar.




Si utilizza per **navigare tra le diverse sezioni** ed è *sempre presente*.
E' costituita da 5 tab, poi c'è more. In questo caso non c'è solo l'icona ma anche la scritta.
La differenza con la tab bar è che le icone non sono standard e quindi ci sono anche delle scritte.

### 14.4.5 Tabelle.




*Corrispondono alle liste e cards di Android* e **mostrano contenuto, utili per la navigazione**.
Sono costituite da una singola colonna (elementi raggruppabili) e sono scrollabili.
Si aggiornano spesso.

Le righe delle tabelle potrebbero essere:

- *default*;

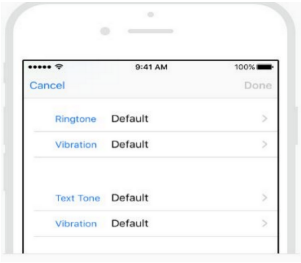

default

- *subtitle*;


subtitle

- *value1 or value2*: rappresentano tabelle con valori all'interno di una tabella.


value1

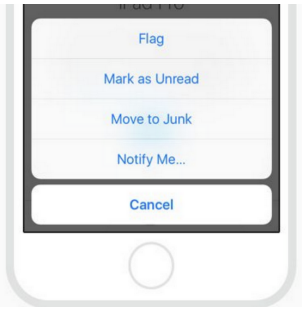



value2

### 14.4.6 Alerts.

Sono **messaggi del sistema o dell'app con titolo e testo brevi in cui l'utente l'utente può dover prendere decisioni anche annullando**.
Soo asincroni e interrompono il task in corso.
Questi alerts sono modali perchè se clicco fuori dal dialog non vengono effettuate azioni sulla vista generale ma solo sul dialog.
Per non far prendere abitudine all'utente si possono scambiare i cancel ,. . .

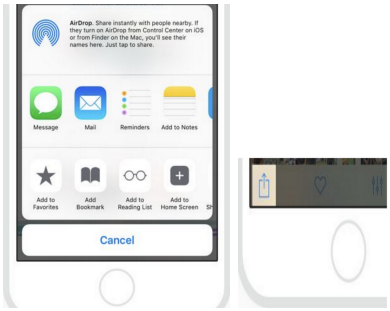
### 14.4.7 Action sheets.




**Fornisce opzioni di scelta per l'esecuzione di un azione**, ed appare in seguito ad un'azione da parte dell'utente.
Queste vengono usate quando ho azioni che:

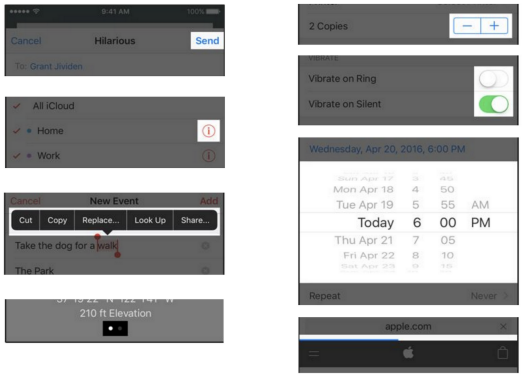
- Prevedono più possibilità.
- Sono potenzialmente "Distruttive".

### 14.4.8 Activity views.




Usando action button che significa condividi o esporta può portare a queste activity views.
Sono personalizzabili e solitamente sono fornite di default e possono essere utilizzate **per eseguire un task utile nel contesto attuale**.

### 14.4.9 Controls.




- Bottoni predefiniti come send o cancel.
- Icone standardizzate come info, ...
- Azioni predefinite quando si seleziona un testo.
- Page control per scegliere le viste.
- Step che permettono di settare dei valori interi.
- Switch, pickers per selezionare giorno e ora.
- Metafora di lucchetti di combinazione.
- Progress bar, ...

### 14.4.10 Controls: text fields.


Bisogna ***usare la tastiera più appropriata per quel testo***.
Per il numero di testi sulla tasiera ci sono 10 tasti e per questo ci sono sistemi accorgimento,...
Per il testo numerico infatti ci sono delle tastiera più grandi solo per i numeri.

# 15 Affordances and Signifiers.

**Gli oggetti di uso quotidiano riflettono problemi di progettazione dell'interfaccia**: *maniglie delle porte lavatrici telefoni, ...*
Adattare le nozioni di convenienza, metafora, modello concettuale all'HCI fornisce regole di progettazione.

## 15.1 Affordances.

**Capacità di azione di un oggetto: tutte le azioni che è fisicamente possibile fare con questo oggetto**.
Le offerte esistono indipendentemente dal fatto che siano percepite o meno dell'utente.
Anche in IUM c'è affordances.

## 15.2 Perceptual Learning - Eleanor Gibson.

**Bisogna imparare a riconoscere affordances**.
*"percepiamo per imparare, così come impariamo a percepire".*
Tutto quello che vediamo lo valutiamo in termini di uso.

## 15.3 Signifiers - Don Norman.

Riguarda la percezione di affordances di un oggetto: **le percezione di possibili capacità di azione su un oggetto come la forma, le dimensioni e l'aspetto dell'oggetto suggeriscono cosa si può fare con esso**.
*"gran parte della conoscenza quotidiana risiede nel mondo, non nella testa" (Norman, 1988)*
I signifiers ci dicono come posso usare un oggetto, le *affordances* sono i **possibili usi di un oggetto che sono intrinseci di un oggetto**.
C'è un affordances che non conosco perchè non c'è nessun signifiers che mi suggerisce che quell'affordances esiste.

## 15.4 Meaningful in HCI.

Un *indicatore percettibile* (visivo, uditivo) che **spiega l'atteggiamento adottare nei confronti di un oggetto**. Più importante delle offerte per interagire con un sistema