
SAPIENZA
UNIVERSITÀ DI ROMA

**Facoltà di Ingegneria dell'Informazione, Informatica e Statistica**
**Dipartimento di Informatica**

# Interazione Uomo Macchina

**Autore:**
Simone Lidonnici

24 novembre 2024

# Indice

# 1 Interfacce

Un'**interfaccia** è un posto dove due sistemi indipendenti e solitamente non correlati si incontrano ed agiscono comunicando tra di loro.
Un'**interfaccia utente (user interface o UI)** in IUM è uno spazio dove avviene l'interazione tra l'umo e la macchina. Lo scopo di questa interazione è di permettere un efficiente controllo della macchina da parte dell'uomo, mentre la macchina manda indietro informazioni.
La user interface contiene elementi hardware e software e fornisce:

- Input: permettendo all'utente di manipolare la macchina
- Output: permettendo al sistema di indicare gli effetti delle manipolazioni dell'utente

## 1.1 Storia delle interfacce

La storia delle interfacce ha alcuni step molto importanti:

- Memex di Vennar Bush: Un sistema proto-hypertext con un apparecchio elettromagnetico per leggere una libreria interna e aggiungere link o note
- Sketchpad di Ivan Suntherland: Usava CRT e un dispositivo a penna, è stato il primo esempio di **Graphical User Interface (GUI)**. Precursore della moderna **Computer-aided Drafting (CAD)**
- Interfacce WIMP (Window Icon Menu Pointer): interfacce dei Computer
- Interfacce Post-WIMP: Interfacce che includono realtà virtuale, basate sui gesti o sulla voce, ecc...

### 1.1.1 Tipi di interfacce Post-WIMP

Le interfacce **Post-WIMP** possono essere di diversi tipi:

- Touch Screen: che può a sua volta essere creato in diversi modi:
  - Resistivo: usato in ospedali, fabbriche per la sua grande resistenza ai liquidi
  - Capacitivo: sfrutta il fatto che il corpo umano è un conduttore di corrente e toccando lo schermo causa un cambiamento di capacità nel campo elettrostatico dello schermo
  - Infrarossi: vengono usati dei raggi a infraroddi che sono interrotti dal dito dell'utente quando tocca lo schermo
- Interfacce basate sui gesti

- Interfacce basate sulla voce
- Interfacce tattili: basate su applicare forza e movimento all'utente
- Interfacce tangibili: danno forma fisica ad informazioni digitali
- Interfacce naturali: sono interfacce invisibili o che lo diventano quando l'utente impara ad usarle. L'aggettivo "naturali" si riferisce al fatto che l'utente quando ci interagisce si sentirà come "naturale" nell'usarle
- Interfacce organiche: computer in ogni modo e forma
- Interfacce cinetiche: basate sui movimenti
- Interfacce immersive: reltà virtuale, ecc...

# 2

# Needfinding

In un processo di creazione **Human-Centered**, la prima cosa da capire è cosa vuole l'utente in quella determinata situazione su cui ci stiamo concentrando. Questo implica capire:

- Chi sono gli utenti?
- Come stanno facendo quella cosa ora?
- In quale contesto la stanno facendo?

Gli utenti non sono rappresentati né dal progettista, né dal cliente ma bisogna vedere gli effettivi utenti che poi utilizzeranno il sistema.
I metodi per effettuare il **needfinding** sono diversi:

- **Osservazione**
- **Diari**
- **Interviste**
- **Inchieste contestuali**

## 2.1 Osservazione

L'osservazione ci permette di ottenere dei dati come l'ambiente degli utenti o il loro linguaggio, però ha dei rischi come il male interpretare una cosa che si osserva o interrompere il normale svolgimento dell'attività. Dall'osservazione dovremmo imparare cosa fanno gli utenti, quali obbiettivi hanno, similitudi e differenze tra di loro e altri tipi di contesti. Inoltre si può capire la differenza tra il processo e la pratica:

- Processo: come le cose andrebbe svolte teoricamente
- Pratica: insieme di trick, operazione e informazioni che si imparano sul campo e fanno parte delle attività quotidiane

Quando si osserva non si deve partire con già una soluzione in mente, perché questo potrebbe limitare l'innovazione.

### 2.1.1 Tipi di osservazione

Ci sono diversi modi di osservare gli utenti:

- **Osservazione controllata**: in un contesto laboratoriale
  - Facile da riprodurre: se si usa un approccio quantitativo è facile ottenere gli stessi risultati
  - Facile da analizzare: dati quantitativi sono facili da analizzare
  - Veloce da condurre
  - Effetto Hawthorn: essere osservati potrebbe cambiare il modo in cui l'utente esegue l'azione
- **Osservazione naturalistica**: nel vero contesto in cui si trova l'utente
  - Più affidabile
  - Più utile per l'ideazione
  - Difficile avere un campione rappresentativo: richiede più tempo e denaro per essere eseguita
  - Difficile da replicare
  - Difficile manipolare le variabili esterne: meteo, ora del giorno ecc...
- **Mescolarsi agli utenti**: sia dievntando "parte del muro", cioè non essendo intrusivo, sia diventando "uno di loro"

## 2.2 Diari

I diari sono strumenti che richiedono all'utente di prendere appunti sulle sue azioni. Questo può essere fatto ogni volta che compie una specifica azione o ad intervalli regolari. Bisogna dare degli incentivi agli utenti e l'analisi potrebbe essere fatta successvamente dai ricercatori o in un intervista.

## 2.3 Interviste

Le interviste possono essere di due tipi:

1. **Interviste di persona**: richiedono tempo e conoscenze, possono essere strutturate o non e possono essere singole o a gruppi
2. **Questionari**: più veloci e superficiali, avendo un set di domande con risposte predefinite

Bisogna ricordarsi che l'utente non sa veramente ciò che vuole e potrebbe rispondere nel modo che il ricercatore vuole sentire e non come vorrebbe. Inoltre all'utente manca creatività e inventiva per capire il nuovo prodotto.
Per scegliere gli utenti si possono prendere:

- Rappresentanti dell'utente target
- Utenti di un sistema simile

- Non utenti (per prodotti nuovi)
- Approssimazioni con utenti simili

Inoltre è utile chiedere a degli utenti specifici come:

- **Lead user**: utenti che hanno need nuovi, prima degli altri, che sono competenti e trovano da soli la soluzioni a questi need. Possono aiutare a capire i need che gli atri avranno nel futuro
- **Extreme user**: utenti che spingono un sistema esistente al massimo, che trovano problemi difficili da individuare e hanno need che hanno anche altri ma più visibili
- **Esperti**: permettono di discutere anche problemi più difficili e astratti e danno informazioni più attendibili sul comportamento degli utenti

### 2.3.1 Interviste di persona

Quando si intervistano gli utenti di persona bisogna prima introdursi e spiegare il proprio obiettivo, poi bisogna iniziare con domande aperte e che non conducano verso un punto specifico. Le domande strutturate sono più facili da capire e nel caso di domande quantitative bisogna sempre chiedere cosa si intende specificatamente con la risposta data (per esempio un 4 in una scala da 1 a 5).
Si possono fare anche delle tipologie di interviste specifiche:

- **Storiche**: per comprendere la sequenza di eventi che spiegano il comportamento degli utenti
- **Process mapping**: chiedere di descrivere tutto il processo
- **Laddering**: continuare a chiedere ”perché?”
- **Contesto culturale**: per capire il contesto
- **Intercettazioni**: fare una singola domanda

Degli esempi di domande buone sono:

- Parlami della tua giornata tipo
- Dimmi tre cose positive riguardo a . . .
- . . . e tre cose negative
- Cosa è andato storto di recente con l’applicazione? Come hai affrontato la situazione?
- Cos’altro avremmo dovuto chiederti?

Esempi invece di domande da evitare sono:

- La funzionalità $x$ è importante per te? (Domanda ”indirizzata”)
- Cosa ti piacerebbe in uno strumento? (Gli utenti sono esperti nel loro dominio, non nella progettazione)

- Cosa ti piace in $x$? (Domanda che presuppone qualcosa: potrebbe non piacergli affatto)
- Cosa faresti in una situazione ipotetica? (Gli utenti non riescono a immaginare l'ambiente completo o una situazione ipotetica)
- Quanto spesso fai $x$?(Gli esseri umani sono pessimi nel fare stime e tendono a rispondere in modo distorto. Potrebbe essere analizzabile dai log se un'applicazione esiste già)
- Domande binarie (sì/no) (Non rivelano motivazioni)
- Raccontami una storia su di te
- Come prendi una decisione? Ti incontri con qualcuno? Qualcuno decide senza di te? ...(Evita di suggerire possibili risposte)

### 2.3.2 Questionari

I questionari online sono familiari, di basso costo e possono raggiungere un'utenza molto vasta, inoltre le risposte possono essere facilmente visualizzate in dei grafici. I problemi dei questianari è che non permettono di fare un'analisi approfondita data l'impossibilità di fare domande di specificazione. Le risposte potrebbero poi non essere del tutto veritiere se si fanno domande sulla memoria o su cose "sensibili" (soldi, emozioni). Bisogna evitare di fare domande complesse, meglio due più semplici che una complessa, evitare parole negative e non fare domande "biased". I tipi delle domanda che si possono fare in un questionario sono:

- Aperte: bisogna sollecitare informazioni specifiche, per evitare risposte troppo vaghe e richiedono una metodologia per analizzare le risposte
- Chiuse: hanno diverse scelte possibili, rappresentate su una scala

Per le domande chiuse si possono usare diversi tipi di scale:

- **Nominale**: insieme predefinito di risposte disgiunte senza ordine
  - Esempi: colore preferito, città, genere
  - Statistiche: frequenza, modalità
- **Ordinata**: risposte disposte con un ordine, non c'è una "distanza" predefinita tra le risposte; bisogna chiedere quanto si è d'accordo con un'affermazione. I valori agli estremi vengono poco scelti e un numero pari di risposte evita la risposta neutrale
  - Esempi: preferenze da 1 a 5, stelle, pollici in giù o in su
  - Statistiche: ordine, mediana, percentuale
- **Intervalli**: risposte numeriche con valore nullo e unità di misura arbitraria
  - Esempi: data, ora, temperatura
  - Statistiche: media, varianza
- **Ratio**: risposte numeriche con valore nullo fisso e unità di misura arbitraria
  - Esempi: durata di qualcosa, età

  - Statistiche: tutte

- **Assoluta**: risposte numeriche, che indicano la cardinalità di un insieme
  - Esempio: numero di impiegati
  - Statistiche: tutte

## 2.4 Inchieste contestuali

Un'**inchiesta contestuale** indica che si guarda l'utente eseguire l'azione nel suo ambiente naturale e si fanno domande per capire come e perché l'utente fa delle precise cose. Gli utenti parlano di quello che stanno facendo e sono più partecipi nella sessione. A differenza delle interviste e dei sondaggi, in cui gli utenti devono ricordarsi cosa fanno e tendono a riassumere il processo, lasciando da parte ragionamenti e motivazioni che potrebbero essere importanti, nelle inchieste contestuali gli utenti riescono a spiegare melio cosa stanno facendo in quel momento. A paragone dell'osservazione invece le inchieste premettono di capire cosa pensano gli utenti, mentre disturbano l'utente. Nei casi in cui l'utente non debba essere disturbato l'osservazione è meglio (per esempio nel caso di medici).
Nelle inchieste c'è il rischio che l'utente inizi a riassumere quello che sta facendo perchè si sente osservato, oppure pensa di dover reagire a tutto ciò che succede nel sistema. Inoltre vengono introdotti dei "bias" sia da parte del ricercatore, che non da a tutto ciò che impara la stessa importanza, sia da parte dell'utente che potrebbe cambiare le sue risposte per accordarsi con le interpretazioni del ricercatore.

# 3

# Task analysis

Dopo aver raccolto informazioni su cosa vogliono gli utenti, bisogna esplicitare questi **need**, come rappresentazione intermedia prima di ideare l'interfaccia. Bisogna rappresentare cosa l'utente vuole fare tramite il nostro design a livello astratto (per esempio mangiare bene, monitorare il sonno, ...), poi a livello concreto facendoli diventare dei **goal**. I goal a differenza dei need dipendono dal contesto, dalle risorse disponibili e più di un goal può derivare dallo stesso need. per semplicità da adesso considereremo i need e i goal uguali.
Dai need e goal bisogna poi ideare dei **task**, che indicano:

- Cosa farà l'utente
- Cosa userà
- Come compierà il suo goal

Un task è essensialmente un goal soddisfatto tramite un insieme ordinato di azioni. Le azioni sono dei task semplici (per esempio attaccare la spina di un elettrodomestico, prendere un oggetto, ...). La task analysis ci permette di capire cosa vogliono raggiungere gli utenti, cosa fanno per raggiungerlo e anche quali esperienze e conoscenze precedenti influenzano l'utente.

## 3.1 Storyboards

Gli **storyboards** sono delle rappresentazioni grafiche del sistema che si intende realizzare, senza però nessuna specifica funzionalità esplicitata. Rappresentano l'esecuzione di un task tramite una sequenza di sketch, che mostrano i punti chiave dell'esecuzione del task. Gli storyboards dovrebbero contenere l'ambiente in cui si svolge il task e le persone coinvolte, gli step dall'esecuzione del task senza nessuna UI dettagliata, il motivo del task e alla fine la soddisfazione finale con need soddisfatto spiegato.
I benefici degli storyboards sono:

- Enfatizzare il modo in cui un'interfaccia esegue un task
- Concentrare la conversazione e il feedback sulle attività dell'utente
- Accordare tutti sugli obiettivi dell'app
- Non specifica nessun dettaglio dell'interfaccia utente (pulsanti, ecc.)

## 3.2 Personas

I **personas** sono strumenti usati nel design di interfacce che rappresentano modelli astratti di utenti reali create sulla base di dati demografici, comportamentali e psicografici, permettendo di capire meglio i need e le azioni degli utenti. Hanno anche l'obiettivo di creare empatia nei confronti degli utenti e guidano le decisioni di progettazione, aiutando a ottimizzare l'esperienza dell'utente. Consentono anche di identificare le esigenze degli utenti e le sfide.
Un persona è caratterizzato da:

- Nome: dà un'identità alla persona
- Foto: permette una connessione più umana con la persona
- Caratteristiche demografiche: età, sesso, professione, istruzione, ecc.
- Obiettivi: cosa cerca di ottenere utilizzando l'interfaccia?
- Frustrazioni: quali sono le sfide e i problemi che potrebbe sperimentare?
- Comportamenti: come utilizza il sistema? quali sono le sue abitudini?

I vantaggi dei personas sono:

- **Creare empatia**: aiuta il designer ad acquisire una prospettiva simile a quella dell'utente, avendo interiorizzato gli obiettivi, i bisogni e i desideri della persona.
- **Sviluppare il focus**: rende più evidente che non è possibile progettare per tutti, altrimenti si rischia di progettare per nessuno.
- **Comunicare e raggiungere un consenso**: aiuta a comunicare le conclusioni della ricerca alle persone che non hanno potuto partecipare agli incontri con gli utenti.
- **Prendere decisioni**: vedendo il mondo dalla prospettiva dell'utente, diventa molto più semplice determinare cosa è utile e cosa è un caso limite.
- **Misurare l'efficacia**: possono fungere da sostituti degli utenti.

# 4

# Prototyping

Un **prototipo** è un modello di sistema interattivo che simula alcune caratteristiche, aspetti o funzioni ma non tutte. Un prototipo deve essere realistico solo per gli aspetti che ci interessano in quel momento e deve essere valutato con le stesse condizioni dell'interfaccia finale. Per progettare la UI per un determinato storyboards bisogna immaginare l'interazione che l'utente avrà con l'interfaccia considerando anche gli strumenti disponibili, come hardware, widget e stili di interazione. L'obiettivo del prototyping è di generare nuove idee e testarle con gli utenti con tecniche diverse in base allo stage della progettazione e dalle persone che dovranno testarlo: utenti, clienti, manager,.... Non bisogna pensare alla UI prima di concentrarsi sul task che questa deve svolgere.
Ci sono diversi approcci al prototyping:

- **Throw away**: conoscenza utilizzata, prototipo scartato
- **Incrementale**: la varie funzioni vengono aggiunte una alla volta al prototipo
- **Evolutivo**: un prototipo serve come base per il successivo, che lo migliora

La prototipazione è un ciclo in cui finché un prototipo non viene valutato positivamente, si torna a riprogettarlo. Ad ogni iterazione bisogna capire cosa è sbagliato (le cause non solo i sintomi), ipotizzare come migliorare ma bisogna avere anche un buon punto di partenza (non si ripropone un'idea iniziale scartata).
I principali problemi del prototyping possono essere:

- Tempo: fare protopiti richiede tempo e se uno viene scartato sembra tempo buttato
- Pianificazione: è difficile pianificare il processo di design e i costi durante il prototyping
- Caratteristiche non funzionali sacrificate: sicurezza, affidabilità, tempo di risposta, ...

## 4.1 Tipi di prototipi

### 4.1.1 Low Fidelity Prototypes

I **Low Fidelity Prototypes** sono dei prototipi disegnati su carta che modellano la UI, rappresentando finestre, menu, caselle di testo. I paper prototype sono utili perché facili da realizzare, da cambiare durante i test con gli utenti e non richiedono nessuna skill di programmazione.
Per testare un paper prototype il gruppo di designers deve coprire tre ruoli precisi:

1. Computer: simula il computer e cambia schermata all'occorrenza senza dare nessun feedback che il computer non darebbe

2. Facilitator: presenta l'interfaccia all'utente e gli spiega il task che deve compiere, facendo domande per capire cosa sta facendo e lo tiene sui binari del task da eseguire
3. Osservatore: sta zitto e prende appunti

I paper prototype permettono di capire se l'interfaccia fa quello che deve, se è facilmente navigabile, se si capiscono le etichette dei pulsanti e cosa è necessario che ci sia sullo schermo.

### 4.1.2 Medium Fidelity Prototypes

I **Medium Fidelity Prototypes** sono prototipi fatti al computer che renderizzano la UI e accettano degli input switchando pagina. Sono ancora composi da un design semplice, composto da poche pagine predefinite con informazioni statiche. Questo causa una ”ricerca” delle poche cose che sono veramente cliccabili nell'interfaccia, però aiuta a capire se la UI è facilmente capibile e come è l'andamento delle schermate.

### 4.1.3 High Fidelity Prototypes

Gli **High Fidelity Prototypes** sono vere e proprie applicazioni con layout, grafica e colori che rappresentano una versione finale della progettazione. Sono molto più costose da realizzare e si concentrano maggiormente sulla grafica che sull'interazione.
Permettono di capire:

- Se i colori, font e altri elementi sono scelti bene
- Se i pulsanti sono abbastanza grandi e posizionati nel punto giusto
- Se l'utente nota i feedback dell'interfaccia, come cambi di cursore

## 4.2 Tecnica ”Mago di Oz”

La tecnica ”**Mago di Oz**” viene usata per testare un'applicazione che è veramente finita, cioè con UI e algoritmi finiti, ma senza scrivere effettivamente il codice dietro. Questa tecnica si basa sull'avere una persona che simula la macchina, anche tecnologie future.
Per applicarla bisogna creare un'interfaccia che implementi parte del sistema ma permetta al ”mago” di eseguire le azioni al posto della macchina tramite un'interfaccia ”dietro”. Bisogna anche definire quando dovrebbe rispondere il ”mago” e come secondo l'algoritmo, questo può essere difficile perché bisogna tenere conto delle limitazioni del sistema e emularlo molto velocemente. Inoltre si potrebbe essere troppo ottimisti rispetto al sistema (un sistema che non sbaglia mai).
Questa tecnica ha diversi vantaggi:

- Più veloce ed economica della maggior parte dei prototipi interattivi
- Più “reale” dei paper prototype
- Facile fare variazioni
- Aiuta a identificare bug e problemi con la progettazione attuale
- Permette di immaginare applicazioni difficili da realizzare
- Fare il ”mago” permette di capire meglio i requisiti degli algoritmi

# 5

# Valutazione di un prototipo

## 5.1 Valutazione tramite utenti

Durante il test di un prototipo bisogna eseguire una valutazione mediante la partecipazione degli utenti. I partecipanti devono essere persona rappresentative degli utenti, con esperienza simile tra loro e conoscenza dei sistemi e del dominio a cui ci si sta riferendo. Lo scenario deve rappresentare una situazione potenzialmente reale in cui si immedesimano i partecipanti, ai quali si chiede di svolgere un determinato task.
Le valutazioni possono essere:

- **In laboratorio**: con sistemi specializzati, non si è interrotti ma non ha contesto e non permette di osservare utenti che collaborano. Utile se la location del sistema è pericolosa o impraticabile.
- **Sul campo**: ci si trova in un ambiente naturale, ma ci sono distrazioni o suoni fastidiosi. Utile se il contesto del sistema è cruciale per la valutazione.

### 5.1.1 Metodi di osservazione

Ci sono diversi metodi per osservare come un utente si interfaccia con un prototipo:

- **Think aloud**: si osserva l'utente eseguire il task e gli si chiede di spiegare cosa sta facendo e perché, oltre a quello che pensa stia succedendo nel sistema. Approccio semplice che permette di capire come un sistema viene effettivamente usato, ma è soggettivo e descrivere cosa si sta facendo potrebbe alterare come viene svolto il task.
- **Cooperative evaluation**: variazione del think aloud in cui anche l'utente collabora nella valutazione, con utente e valutatore che si fanno domande a vicenda, e che incoraggia l'utente a criticare il sistema.
- **Protocol analysis**: si registra l'utente che esegue il task per analizzare poi i risultati. Può essere fatta tramite:
  - Carta e penna: facile, ma limitata dalla velocità di scrittura
  - Audio: buona per think aloud, ma difficile da integrare con altri tipi di protocolli
  - Video: accurata e realistica, ma richiede attrezzature specializzate ed è invadente
  - Computer log: automatico e non invadente, ma genera una grande quantità di dati difficili da analizzare
  - Appunti dell'utente: sono soggettivi, ma utili per studi longitudinali

In pratica si utilizzano delle modalità miste.

- **Post-Task walkthroughs**: si registra l'utente che esegue il task, ma poi la registrazione viene fatti vedere al partecipante per commentarle in due momenti specifici:
  - Immediatamente: si ha ancora le informazioni in mente
  - Dopo un po' di tempo: permette all'utente di sviluppare delle domande

  Questo è utile per identificare le ragioni delle azioni e le alternative considerate. Molto utile nel caso in cui think aloud non sia possibile.

Tutti questi metodi sono però soggettivi e dipendono dalle capacità e dai bias del valutatore. Le misure prodotte da questi metodi sono quindi non numeriche ma qualitative.

## 5.2 Esperimenti

Un altro metodo di valutazione sono gli **esperimenti**, cioè delle valutazioni controllate di specifici aspetti ipotetici da testare. Vengono considerate un numero di condizioni sperimentali che differiscono solo per il valore di alcune variabili controllate. Il cambiamento nel comportamento degli utenti viene attribuito alle diverse condizioni.
Gli esperimenti sono composti da:

- Soggetti: rappresentano gli utenti e devono essere un campione sufficiente
- Variabili: cose da modificare e misurare, possono essere:
  - **Variabili indipendeni (VI)**: caratteristiche cambiate per produrre condizioni differenti (stile dell'interfaccia, numero di oggetti in un menu), tipicamente sono variabili discrete
  - **Variabili dipendenti (VD)**: caratteristiche misurate nell'esperimento (tempo richiesto, numero di errori), possono essere variabili discrete o continue positive
- Ipotesi: cosa si vuole mostrare (idea provvisoria il cui valore dev'essere accertato), possono essere:
  - **Alternative**: predirre che una variazione di una variabile indipendente causerà un cambiamento di una variabile dipendente
  - **Nulle**: predirre che non ci sarà nessun cambiamento nelle variabili indipenti, si punta a confutarle
- Design sperimentali: come si vuole fare la determinata cosa

Per eseguire un esperimento bisogna:

1. Scegliere l'ipotesi
2. Scegliere le variabili dipendenti e indipendenti (quelle indipendenti devono essere più semplici possibili)
3. Scegliere i partecipanti

4. Definire delle condizioni sperimentali: si cambia solo una variabile indipendente (VI) rispetto alla condizione di controllo (condizione base)

5. Scegliere il metodo sperimentale:

    - **Between subject**: a ogni partecipante viene assegnata una sola condizione sperimentale ed esegue il test una volta sola. Le differenze tra gli utenti possono alterare il risultato della valutazione, ma non risente del "trasferimento dell'apprendimento".
    - **Within subject**: ogni partecipante esegue un test su ogni condizione. Le differenze tra gli utenti influenzano meno il risultato della valutazione, ma risente del "trasferimento dell'apprendimento".

Il "trasferimento dell'apprendimento" indica il fatto che se si esegue prima il test sulla condizione di controllo, poi si faranno meno errori nelle condizioni sperimentali, anche se queste dovessero rivelarsi peggiori. Si può mitigare facendo svolgere a metà campione prima i test sulla condizione di controllo e all'altra dopo, nel caso di within subject.
Alla fine dell'esperimento bisogna raccogliere i dati e osservare le VD, controllando la distribuzione normale della VD, se i dati soddisfano la distribuzione normale, allora si eseguono test paramentrici (tecniche di analisi statistica standard molto robuste), sennò si eseguono test non parametrici. Alla fine si verifica se si può accettare o rifiutare l'ipotesi.

### 5.2.1 A/B testing

L'**A/B testing** è più semplice tipo di esperimento, che comprende una sola variabile indipendente con solo due valori (A o B), si utilizza solitamente sui siti web mostrando a caso una delle due condizioni all'utente.
L'ipotesi può essere nulla, cioè che non c'è differenza tra A e B, oppure un'ipotesi alternativa da dimostrare, cioè che ci sia una differenza statisticamente significativa. Il test statistico più comune è la distribuzione Gaussiana.
I pro di questo tipo di testing sono:

- Semplice da progettare e realizzare
- Risultato facile da misurare
- Risultato molto chiaro

I contro sono:

- Adatto solo al caso di due condizioni diverse
- Difficile decidere cosa misurare in alcuni casi

## 5.3 Valutazioni tramite esperti

Coinvolgere gli utenti a ogni valutazione è molto costoso, alcune volte conviene usare degli **esperti** che valutano l'impatto di un design sull'utente tipico. Gli esperti possono individuare aspetti che possono causare difficoltà perché violano noti principi cognitivi o ignorano risultati empirici accettati.
Gli esperti sono utili perché poco costosi e utilizzabili in ogni fase del progetto e dello sviluppo,

hanno però il problema di valutare solo l'aderenza ai principi conosciuti e non il vero uso del sistema.
Ci sono diversi tipi di Expert based evaluation:

- **Cognitive walkthrough**: valuta il design in base a quanto bene supporta l'utente nell'imparare il task. Viene eseguito da esperti in psicologia cognitiva e consiste nell'eseguire passo passo il task, rispondendo ogni volta alle seguenti domande:
  - L'effetto dell'azione 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**: valuta il progetto secondo delle euristiche, cioè delle linee guida da seguire. In particolare, si basa su 10 euristiche proposte da J. Nielsen e R. Molich:
  1. Visibilità dello stato del sistema
  2. Corrispondenza tra il sistema e il mondo reale
  3. Libertà e controllo da parte degli utenti
  4. Coerenza e standard
  5. Prevenzione degli errori
  6. Riconoscere piuttosto che ricordare
  7. Flessibilità ed efficienza d'uso
  8. Design minimalista ed estetico
  9. Aiutare gli utenti a riconoscere, diagnosticare e correggere gli errori
  10. Aiuto/guida e documentazione
- **Model-based evaluation**: si usano modelli esistenti per valutare l'interazione. Esempi di modelli usati sono:
  - GOMS (Goals, Operators, Methods, Selections): modello di previsione delle performance dell'utente con un'interfaccia.
  - KLM (Keystroke-Level Model): modello di previsione dei tempi necessari per compiti fisici di basso livello (uso di tastiera e mouse).
  - Dialog models: per valutare le sequenze dei dialoghi, stati irraggiungibili, dialoghi circolari, ecc.
- **Review-based evaluation**: si utilizzano i risultati di studi precedenti per supportare o rifiutare parti di design e assicurarsi che i risultati siano trasferiti al nuovo design. Richiede un esperto per assicurarsi che vengano fatte delle corrette assunzioni.

# 6

# Sviluppo di un sistema

## 6.1 Modello waterfall

Il modello **waterfall (a cascata)** è un metodo di sviluppo che consiste in 6 fasi lineari:

1. **Specifica dei requisiti**: il designer e il cliente cercano di capire cosa il sistema dovrebbe offrire.
2. **Design architetturale**: si descrive ad alto livello come il sistema offrirà dei servizi e come questi soddisfino i requisiti sia funzionali che non funzionali.
3. **Design dettagliato**: raffinamento dei componenti architetturali per identificare moduli da implementare separatamente. Il raffinamento è governato dai requisiti non funzionali.
4. **Programmazione e testing**
5. **Integrazione e testing**
6. **Operazioni e manutenzione**

Alcune volte il metodo waterfall fallisce per software perché le specifiche potrebbero cambiare durante lo sviluppo. Il design iterativo sopera alcuni dei problemi dati dai requisiti incompleti.

## 6.2 Metodo Agile

Il metodo **agile** è un metodo di sviluppo che incoraggia:

- Ispezioni frequenti
- Adattamento
- Collaborazione e auto-organizzazione
- Rapido rilascio di software di alta qualità
- Allinea lo sviluppo con i bisogni dell'utente

I metodi agile sono adattivi, al posto che predittivi, cioè si adattano ai cambiamenti e sono orientati verso l'utente più che verso il processo.
Peculiarità del Metodo Agile:

- **Pair Design Programming**: Due persone lavorano insieme, alternandosi per identificare e correggere errori.
- **Co-located Teams**: Diversi team lavorano fisicamente vicini per facilitare la comunicazione e la collaborazione.

### 6.2.1 Metodo Agile UCD

Il metodo **Agile UCD (User-Centered Design)** risolve i problemi di poca vicinanza agli utenti del metodo agile e di "early release" e collaborazione del metodo UCD. Si compone di cicli ben definiti:

- **Ciclo 0 (Ricerca e analisi)**: si analizzano gli obiettivi del sistema e si eseguono delle ricerche contestuali per poi creare dei personas e degli scenari.
- **Ciclo 1**: si definiscono la quantità di design e di test da fare nei cicli successivi. Dura da 2 a 4 settimane con una data finale fissata.
- **Cicli successivi**: il developer e il designer si danno feedback a vicenda, in particolare:
  - **Developer**: implementa l'iterazione attuale basandosi sul design della precedente.
  - **Designer**: testa l'iterazione precedente, esegue il design dell'iterazione attuale e conduce ricerche per la successiva.

Questo metodo permette di far collaborare team multidisciplinari, di poter cambiare gli obiettivi e le specifiche in qualsiasi momento e di rilasciare il software velocemente e spesso.

# 7
# Modi

In questo capitolo si utilizza una specifica terminologia:

- **Content**: l'insieme delle informazioni che ci sono nel sistema e sono utili per l'utente.
- **GID (Graphical Input Device)**: meccanismo per comunicare al sistema una particolare locazione o la scelta di un oggetto (tipicamente la posizione del cursore).
- **GID button**: bottone principale del GID.
- **Tap**: azione di premere e rilasciare un tasto (che ritorna automaticamente alla sua posizione originale).
- **Click**: posizionare il GID e fare tap sul GID button.
- **Double Click**: posizionare il GID e fare tap sul GID button due volte senza spostare il GID tra i due click.
- **Drag o Swipe**: premere il GID button e, senza rilasciarlo, muovere il GID per poi rilasciarlo.
- **Gesture (Gesto)**: una sequenza di azioni che viene completata automaticamente una volta avviata (ad esempio scrivere una parola comune o premere "invio").

## 7.1 Interazioni con l'interfaccia

### 7.1.1 Luogo dell'attenzione

Il "**Luogo dell'attenzione**" è l'oggetto fisico o l'idea alla quale stiamo pensando attivamente, c'è un solo "luogo dell'attenzione" alla volta. Questo "luogo" indica che se qualcosa accade al di fuori di esso verrà notata più difficilmente dall'utente, anche se c'è un indicazione dello stato del sistema. Per esempio nel caso del caps lock, alcune volte ci si accorge di averlo cliccato dopo alcune lettere anche se c'è un led che ci indica che è attivo, questo perché il led è fuori dal "luogo dell'attenzione". Questi errori capitano indistintamente sia ad esperti che a principianti, ai primi perché hanno sviluppato delle abitudini, i secondi per l'inesperienza.

### 7.1.2 Tipi di interazioni

Le interazioni con un'interfaccia possono essere di tipo:

- **Noun-verb (o object-action)**: in cui si seleziona prima l'oggetto su cui si vuole eseguire l'azione e poi l'azione stessa (prima si clicca il pennello di paint e poi si disegna). Questo tipo di interazione sposta una volta sola il luogo dell'attenzione (dall'oggetto all'azione) e l'interruzione non richiede un'altra azione.
- **Verb-noun (o action-object)**: in cui si seleziona prima l'azione e poi l'oggetto su cui la si vuole eseguire (prima si seleziona un testo e poi si fa diventare in grassetto). Questo tipo di interazione generadei **modi** sposta due volte il luogo dell'attenzione (prima dall'oggetto all'azione e poi di nuovo sull'oggetto) e nel caso di distrazione ci saranno errori dati dai modi.

Quindi in generale è meglio un'interazione noun-verb, che non crea modi.

## 7.2 Modi

I **modi** sono come l'interfaccia risponde ai gesti. Dato un gesto l'interfaccia è in un modo se l'interpretazione di quel gesto è sempre la stessa, quando viene interpretato in maniera diversa l'interfaccia si trova in un altro modo. Vengono generati solo da un'interfaccia con interazione di tipo verb-noun. Prendiamo per esempio una sveglia con un solo tasto di accensione, che se viene premuto quando la sveglia è accesa la spegne e viceversa. Al buio non si può vedere in che stato è il sistema e quindi non si può stabilire l'effetto della pressione del tasto, causando degli errori di modo.
Per minimizzare gli errori dati dai modi si possono non avere modi o assicurarsi che i comandi richiesti dai diversi modi siano diversi tra loro, in modo che un comando eseguito in un modo sbagliato con causi problemi (non avrà effetto).
Esistono dei modi particolari:

- **Modi temporanei**: un modo che svanisce dopo l'uso
- **Quasi-modi**: un modo che viene ottenuto mantenendo fisicamente un controllo (tasto "shift" per le maiuscole)

### 7.2.1 Interfacce modali

Un'interfaccia è **modale** rispetto ad un gesto quando:

- Lo stato corrente dell'interfaccia non è il luogo dell'attenzione dell'utente
- L'interfaccia risponderà al gesto con una tra N possibili risposte, in base allo stato del sistema

Un'interfaccia può essere modale rispetto ad uno o più gesti e non modale rispetto ad altri, si dice che un'interfaccia è **non modale** se non è modale rispetto a qualunque gesto. Si può anche misurare quanto è modale un'interfaccia in base alla probabilità che un gesto non modale venga usato, quindi 0 sarà completamente modale e 1 completamente non modale.

## 7.3 Affordances e Signifiers

L'**affordance** è la relazione tra le proprietà di un oggetto e la capacità dell'utente di capire come quell'oggetto potrebbe essere usato. La forma, grandezza e apparenza di un oggetto

aiutano l'utente a capire cosa fare con esso. L'esistenza dell'affordance dipende dall'utente e dall'oggetto.
I **signifiers** sono degli indizi che aiutano l'utente a capire cosa fare con un determinato oggetto (frecce, scritte, simboli), sono utili soprattutto nel caso in cui l'affordance non è ovvia e permettono di rendere le azioni intese per quell'oggetto più esplicite.

# 8

# Progetto di applicazioni mobili

## 8.1 Contesti d'uso e caratteristiche

Quando si progetta un'applicazione mobile bisogna considerare il contesto d'uso, cioè che l'utente:

- Non è concentrato a lungo perché in movimento.
- Ha poco tempo a disposizione e va di fretta.
- Viene frequentemente interrotto da eventi esterni.
- Potrebbe avere la necessità di non disturbare le persone intorno.
- La sua attenzione potrebbe essere richiesta da altre attività, avendo mani o vista impegnati.
- Potrebbe trovarsi in condizioni di illuminazione scarsa e potrebbero esserci rumori intorno.

A differenza dei desktop, gli smartphone hanno più piccole dimensioni, ma un'alta risoluzione che permette una buone leggibilità. Bisogna omettere elementi non essenziali all'interfaccia per non causare un'affollamento. Inoltre bisogna eliminare i memory leaks e ridurre i file al minimo per via della memoria limitata. Le schermate vengono gestite in maniera sequenziale. L'applicazione potrebbe essere interrotta in qualsiasi momento, quindi non deve perdere i dati o lo stato e deve ripartire velocemente uguale a come si trovava.
Gli aiuti sullo schermo devono essere minimali, perché l'utente non ha tempo o voglia di leggerli, e non devono togliere spazio sullo schermo. Bisogna usare controlli standard che gli utenti già conoscono e usare una sequenza di schermate logica, con bottoni back per tornare indietro.

## 8.2 Stili di applicazioni

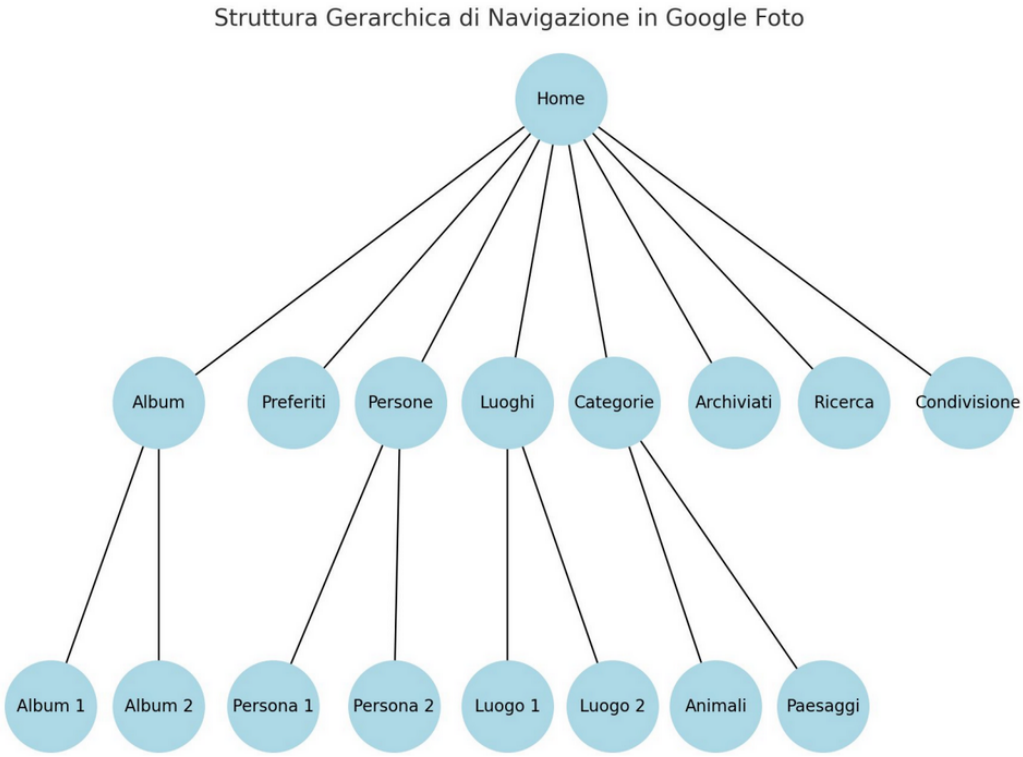
### 8.2.1 Productivity application

Le **productivity application** sono applicazioni che permettono di svolgere compiti basati sull'organizzazione e manipolazione di informazioni dettagliate. Hanno dei task importanti e focalizzano la user experience sul task, con poche informazioni e più iterazione.
I dati sono organizzati gerarchicamente in cui le iterazioni tipiche sono:

- Organizzare la lista
- Aggiungere e togliere elementi dalla lista

- Scandere a livelli di dettaglio successivi e poi eseguire operazioni sul livello scelto

La UI per una productivity app contiene un'interfaccia semplice e pulita, con una schermata per ogni livello di dettaglio e controlli standard. L'enfasi è sulle informazioni e sui task, non sull'ambiente e sull'esperienza, con la possibilità di impostare preferenze per ridurre le informazioni e le scelte ripetitive.
**Esempio:**
Google Foto ha una gerarchia di questo tipo:




Le productivity application usano solitamente un approccio noun-verb in cui scelgo un oggetto e poi eseguo l'azione, ad esempio cliccare su una foto per aprirla o cliccare sul pulsante per eliminare la conversazione. Le animazioni fanno capire che l'azione è avvenuta con successo, senza bisogno di altri feedback. L'approccio noun-verb permette di non creare dei modi che aumentano gli errori. Nonostante ciò l'aggiunta di un oggetto alla lista è comunque verb-noun (scrivere una mail), cliccare il bottone dell'azione apre una vista modale (sono in un modo per scrivere una mail) che sale dal basso andando a coprire la vista precedente. Terminata l'aggiunta (premo invia) la vista modale si chiude, scendendo verso il basso e nella vista precedente verrà aggiunto un elemento alla lista tramite un'animazione.
La vista figlia entra da destra e si sovrappone alla vista padre ed ha un bottone back per tornare indietro, che se cliccato mostra la vista figlia scorrere via verso destra, mostrando di nuovo la vista padre. Questo fa sembrare come se le viste fossero implilate, quindi l'ultima entrata sarà la prima a uscire.
Molte app utilizzano una vista di default, che è la radice dell'albero gerarchico, questo (insieme all'approccio noun-verb) permette di dare all'app rapidità, semplicità d'uso e limita il numero di errori.

### 8.2.2 Utility application

Le **utility application** sono applicazione he consentono di svolgere un task semplice con poco input da parte dell'utente, senza nessuna struttura gerarchica e con molte informazioni e poca interazione. Solitamente permettono di leggere informazioni essenziali su alcuni argomenti, verificare lo stato di qualcosa, ...
Nelle utility app si utilizzano diverse liste, non gerarchicamente collegate, l'interazione tipica comprende:

- Leggere le informazioni
- Cambiare le impostazioni
- Cambiare la sorgente di dati

Le preferenze sono frequenti cambiamenti nella configurazione, quindi è necessario poterle cambiare da dentro l'applicazione.
**Esempio:**
App del meteo

### 8.2.3 Immersive application

Le **Immersive application** sono applicazioni che consentono di svolgere compiti con un ambiente particolare, con poche quantità di testo e che richiedono l'attenzione continua dell'utente. Contengono ambienti ricchi, sono a tutto schermo e sono focalizzate sul contenuto e sull'esperienza di quel contenuto.
Le immersive application non usano controlli standard ma ne creano di nuovi, cercare e scoprire il funzionamento dei controlli è parte dell'esperienza dell'applicazione. Usa spesso grandi quantità di informazioni, ma le mostra in un modo legato al contesto.

## 8.3 Design pattern

I **Design pattern** sono un modo di comunicare problemi di design comuni e le relative soluzioni, non devono essere né troppo generali né troppo specifici. Sono considerati come un "linguaggio condiviso" che fa da refernce base per i designers, permettono di dibattere su delle alternative e possono essere letti anche da non esperti.

### 8.3.1 Design pattern per la UI

Alcuni design pattern per la UI generici sono:

- **Accordion Menu**: menu che si apre scorrendo da destra verso sinistra (menu di google drive o gmail)
- **Dropdown Menu**: menu che si apre scorrendo dall'alto verso il basso (quello delle notifiche del telefono)
- **Cards**: interfaccia divisa in rettangoli con bordi arrotondati di varie forme
- **Breadcrumbs**: serie di menu ordinati sequenzialmente i cui si sa a che punto si è e dove si deve arrivare
- **Hamburger**: tre linee orizzontali parallele che aprono un menu

### 8.3.2 Dark pattern

I **dark pattern** sono dei design volutamente usati per ingannare l'utente e avvantaggiare il sistema. L'esempio maggiore sono gli **Attention-Capture Damaging Patterns (ACDP)**. Gli Attention-Capture Damaging Patterns sono dei pattern ricorrenti che i designer usano per sfruttare le debolezze psicologiche degli utenti e catturare la loro attenzione, solitamente facendogli perdere traccia del proprio obiettivo e del tempo. L'obiettivo è quello di massimizzare l'utilizzo continuo e le interazioni.
Le vulnerabilità sfruttate sono di diversi tipi:

- Notifiche: con le vibrazioni e i led
- Possibilità di ricevere commenti o like: si continua a stare in allerta (ricompenze variabili)
- Ricezione di commenti o like: genera dopamine e ci paragona agli altri (influenza sociale)

L'impatto di questi pattern ha un effetto negativo sugli utenti, perché:

- Promuove la "dipendenza digitale"
- Diminuisce l'attenzione e la produtività
- Diminuisce il senso di autocontrollo
- Risultano dopo in un senso di rimpianto

Alcuni esempi di ACDP (tutti basati sulle ricompenze variabili):

- **Guilty-Pleasure recommendations**: sono raccomandazioni basate sulle interazioni precedenti (content-based) e sulle interazioni di altri utenti (collaborative filtering). Sono molto utili se l'obiettivo della piattaforma è lo stesso dell'utente (value-alignment problem) e fanno rimanere più tempo l'utente sulla piattaforma senza la sua volontà.
- **Neverending autoplay**: un video inizia automaticamente quando uno finisce, senza tempo per riflettere. Questo prolunga le sessioni dell'utente usando la vulnerabilità delle ricompense variabili e riduce l'autonomia dell'utente.
- **Casino pull-to-refresh**: scrollando verso il basso si attiva un ricaricamento animato che potrebbe rivelare dei nuovi contenuti da vedere. Questo porta l'utente a "refreshare" continuamente.
- **Infinite scrolling**: quando si scrolla verso il basso viene sempre aggiunto nuovo contenuto, caricando continuamente la pagina dal basso. Diminuisce lo sforzo per cercare dei nuovi contenuti e promuove una sessione "infinita".

### 8.3.3 Design pattern per Android

Alcuni design pattern per la UI su Android sono:

- **Toolbar**: barra in alto con azioni, informazioni e tasto back.
- **App bar**: barra in alto delle applicazioni con menu apribili e azioni.
- **Tabs**: barra in alto che permette di cliccare diverse sezioni.

- **Navigation drawer**: menu a sinistra che permette di aprire delle altre schermate.
- **Scrolling e paging**: scorrere verticalmente per una lista e orizzontalmente per le pagine.

Fatto fino alla slide 19tris