

Appunti di Programmazione di Interfacce

Simone Pepi¹ e Francesco Iannelli²

¹Stesura appunti

²Riscrittura in L^AT_EX

a.a. 2019/2020 Prof. Mazzei

Indice

1 Introduzione	3
1.1 XX Design	3
1.2 UX Designer	4
1.3 UI Designer	4
2 User Interface	5
2.1 Diversi tipi di interfacce	5
2.2 Categorizzare le CUI	6
3 Good and Bad Design	7
3.1 Design of Useful Things	8
4 Human Centered Design	9
4.1 Desing Thinking vs HCD	10
5 Principi Fondamentali dell'Interazione	11
5.1 Affordance	11
5.2 Significante	12
5.3 Mapping	13
5.4 Feedback	14
5.5 Modello concettuale	14
5.6 Immagine di Sistema	15
6 Constraints, Discoverability e Feedback	16
6.1 Vincoli e mapping	17
6.2 Funzioni Obbliganti	18
6.3 Activity-Centered Control	18
7 How do people do things	19
7.1 I Golfi dell'Esecuzione e della Valutazione	19
7.2 I sette stadi dell'azione	20
7.3 Tre livelli di Processing	21
7.4 I sette Principi Fondamentali della Progettazione	22
7.5 Disruptive Innovation	24
7.6 Agile	24
8 Progettazione	25
8.1 Personas	25
8.2 Requirements	26
8.3 User Stories	26
8.4 Scenarios	27
8.5 Use Cases	28

9 Human Error	29
9.1 Root Cause Analysis	29
9.2 I cinque perchè	30
9.3 Definizione di errore	31
9.4 Prevenzione dell'errore	32
10 Pretotyping	34
10.1 Thoughtland, il mondo dei pensieri	34
10.2 Thoughts without datas are just opinions	35
10.3 Pretotype vs Prototype	35
10.4 Flusso del Pretotyping	36
10.5 Tipi di Pretotyping	37
10.6 Minimum Viable Product	39
11 UX for connected devices	40
11.1 Industry 4.0	40
11.2 Products and Services in the 4.0 era	42
11.3 Real world context	43
12 Natural Language	44

Capitolo 1

Introduzione

Che cos'è il design?

Il design è la **pianificazione o la specifica per la costruzione di un oggetto o sistema, o per l'implementazione di un'attività o un processo**. Si trova quindi agli antipodi della scomposizione del problema in sottopassaggi, cioè del pensiero computazionale.

Il design parte dalla base del problema e **identifica soluzioni per la causa del problema**. Famosa è la frase: *"Se vogliamo che agli utenti piaccia il nostro software, dobbiamo progettare le applicazioni come se fossero persone con cui ci piacerebbe uscire."*

Due delle caratteristiche più importanti di una buona progettazione sono **visibilità** e **comprendibilità**.

- **Visibilità:** è possibile indovinare quali azioni sono possibili e come eseguirle?
- **Comprendibilità:** cosa significa tutto questo? Come va usato? Cosa significano tutti i vari comandi?

Nei dispositivi complessi la visibilità e la complessità richiedono l'uso di manuali d'istruzioni, ma questo lo accettiamo solo se il dispositivo è davvero complesso, ma dovrebbe essere del tutto superfluo per le cose semplici.

1.1 XX Design

Fin dai primi tempi dell'industria del design, la parola *design* normalmente farebbe pensare a qualcosa riguardante la grafica. Il lettore potrebbe sentirsi confuso riguardo a tutte le sfumature di significato che assume questa parola nel mondo del lavoro.

Con l'evoluzione della tecnologia anche l'industria dei media si è evoluta, dalla stampa ai media sul web, mobile e software, così come l'industria del design.

I settori principali della progettazione che andremo a toccare sono: il **design industriale**, il **design dell'interazione** e il **design dell'esperienza utente**. Nessuno di essi è rigidamente definito ma è diverso il punto focale di ognuno:

- **Design industriale:** creazione e sviluppo sia di concetti che di specifiche per ottimizzare la funzionalità, il valore e l'aspetto di prodotti e sistemi, con reciproco vantaggio per gli utenti e i produttori
- **Design dell'interazione:** l'attenzione è concentrata sul modo in cui le persone interagiscono con la tecnologia; lo scopo è migliorare la loro comprensione di ciò che si può fare, ciò che succede e ciò che è appena successo, basandosi su principi psicologici, tecnici ed estetici.

- **Design dell'esperienza utente:** progettazione di prodotti, processi, servizi, eventi e ambienti, mirando soprattutto alla qualità e alla piacevolezza dell'esperienza complessiva.

Per rispondere chiaramente a qualsiasi domanda, dobbiamo innanzitutto illustrare le differenze tra **Graphic Design**, **User Experience Design (UX Design)**, **User Interface Design (UI Design)**.

1.2 UX Designer

La **User Experience** è il modo in cui le persone si sentono **a livello psicologico** quando usano un prodotto. Ogni prodotto utilizzato da qualcuno ha una user experience: giornali, bottiglie di ketchup, poltrone reclinabili, maglioni di cardigan ecc...

Quindi l' **UX design** si occupa di come il prodotto viene recepito, la sensazione che dà, ed è molto diverso dal tradizionale design grafico.

I progettisti UX lavorano in modo da conferire al prodotto la migliore esperienza d'uso possibile, in modo da trasmettere, all'utente che ne farà uso, una sensazione di soddisfazione e non di frustrazione. A tal fine vengono usati una varietà di strumenti tra i quali: analisi competitiva, interviste e sondaggi, il tutto per arrivare a costruire tipologie di *personas* tra gli utenti del prodotto.

Non si può progettare la User Experience, ma si può progettare per la User Experience, ecco perché, per soddisfare appieno gli utenti che useranno il prodotto, è importante comprendere quali sono le loro necessità.

1.3 UI Designer

Dallo studio della UX si crea un abbozzo dell'interfaccia. Non si crea subito il wireframe finale, ma si parte da un analisi dei casi di studio. Esistono più tipi di casi di studio ed ognuno è specifico per delle personas, infatti, personas differenti hanno capacità differenti.

L'UI Design è un procedimento differente dal front-end developing: la materia progetta le guidelines che istruiscono il developer su come creare al meglio una UI.

Possiamo considerare la UI Design come **sotto area** della UX Desing.

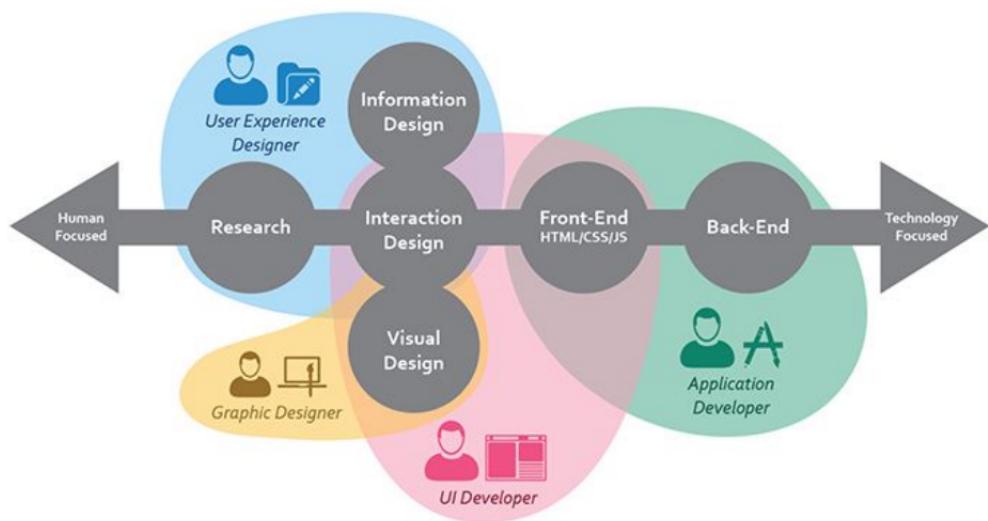


Figura 1.1: Le varie aree del design.

Capitolo 2

User Interface

Quando parliamo di **User Interface (UI)**, in italiano Interfaccia Utente, parliamo dello spazio di un sistema dove avviene l'interazione uomo-macchina: il monitor, il mouse, le casse audio e quant'altro.

L'obiettivo di questa interazione è far sì che l'utente possa controllare e far funzionare la macchina in modo efficace, mentre la stessa macchina reagisce simultaneamente fornendo informazioni che aiutano il suo processo decisionale tramite **feedback**.

In generale, l'obiettivo dell' UI design è quello di produrre una UI che renda facile, efficiente e divertente l'uso di una macchina o di un programma, in modo da massimizzare la User Experience dell'utente finale.

Il termine **user-friendly** non può essere omesso in questa trattazione: tra un'app facile e piacevole da utilizzare e una solo facile da usare, l'utente medio preferirà sempre la prima.

Le interfacce sono strutturate in uno o più layer. L' **HID o Human Interface Device** è la periferica con cui l'utente interagisce con il sistema; come ad esempio mouse, monitor, gamepad, ecc...

Lo **HMI o Human Machine Interface** è un concetto che astrae dall' HID, con HMI, infatti, ci si riferisce allo strato che separa un essere umano che sta utilizzando una macchina dalla macchina stessa. Un device che implementa un HMI è un HID. Quando la macchina in questione è un computer, HMI diventa **HCI o Human Computer Interface**.

2.1 Diversi tipi di interfacce

Una persona umana possiede cinque differenti sensi che possono essere "mappati" in cinque categorie di interfacce possibili, più una categoria introdotta con l'introduzione di visori e giroscopi:

- **tactile UI** (touch)
- **visual UI** (sight)
- **auditory UI** (sound)
- **olfactory UI** (smell)
- **equilibrial UI** (balance)
- **gustatory UI** (taste)

La composizione di più UI prende il nome di **CUI (Composite User Interface)**. Le più comuni CUI sono le **GUI o Graphical User Interface**, le quali sono composte da interfacce grafiche e tattili. Se aggiungiamo anche il suono diventano **MUI (Multimedia User Interface)**.

È bene sottolineare che **aggiungere più interfacce per poter interagire con una macchina utilizzando più sensi non è sempre una buona idea**. Giusto per fare un esempio, prendiamo in esame i video di Facebook: i video venivano riprodotti con l'audio attivo, ma gli ingegneri di Facebook si sono accorti che la maggioranza delle persone che visualizzavano i video, mutavano immediatamente il suono per varie ragioni (e.g. privacy o utilizzo di Facebook in momenti non opportuni), quindi hanno ben pensato di far partire la riproduzione automatica dei video con il suono mutato e introducendo i sottotitoli per le parti del video parlate. Questo oltre ad essere un ottimo esempio di MUI riprogettata in GUI è anche un esempio di tecnica ideata per le utenti disabili e riusata per poter far fruire il prodotto a quelle personas che lo utilizzano in momenti in cui non possono usufruire dell'audio.

2.2 Categorizzare le CUI

Le CUI possono essere categorizzate in tre diverse macrocategorie:

- **Standard:** usano dispositivi standard come tastiere, mouse e monitor
- **Virtual:** schermano il mondo reale e creano un mondo virtuale. Tipicamente utilizzano dei caschi VR.
- **Augmented:** non schermano il mondo reale e erogano contenuti non completamente digitali che prendono forma nella realtà esterna che circonda l'utente, appunto espandendola.

Quando un'interfaccia utente interagisce con tutti i sensi umani viene chiamata **Qualia Interface**, secondo la **teoria Qualia**.

Le CUI possono essere anche **classificate per il numero di sensi** con cui esse interagiscono. Ad esempio, lo *Smell-O-Vision* è una CUI standard 3S (3 sensi) con display, suono e odori. Se si aggiungesse un quarto senso diventerebbe un 4S, si pensi ad esempio alle poltrone dei famosi cinema 4D.



Figura 2.1: Virtual reality



Figura 2.2: Augmented reality.

Capitolo 3

Good and Bad Design

Il buon design valido sempre e per tutti non esiste, poiché si fa design per la user experience di determinate personas. Però possiamo dare due caratteristiche importanti sui cui misurare un buon design:

- **Discoverability:** è la capacità innata di un sistema di veicolare i possibili usi e di comunicare come si usa. Non è detto che una volta capito cosa si può fare si riesca a farlo. Per avere una buona discoverability si usa tipicamente la **visibilità**: un rubinetto con i pomelli bene in vista incrementa la discoverability. In un software tale lavoro è svolto dai pulsanti.
- **Understanding:** è la capacità del prodotto di farsi usare correttamente dall'utente.



Figura 3.1: Discoverability and visibility.



Figura 3.2: Understanding.

3.1 Design of Useful Things

Quando le cose vanno bene, si dimenticano subito!

Questo perché nella psicologia umana le cose devono andare bene per definizione. Quando qualcosa va storto invece, l'amigdala crea un ricordo con un peso molto maggiore rispetto a un ricordo di una esperienza piacevole.

Il design deve quindi preoccuparsi di come funzionano le cose, come vengono controllate e della natura delle interazioni. Quando la progettazione è fatta bene, crea prodotti piacevoli e brillanti, quando è fatta mala, i prodotti sono inutilizzabili e ciò porta a una notevole frustrazione e irritazione.

Le macchine sono concepite, progettate e costruite da esseri umani. Al nostro confronto sono **assai limitate**: non conservano quella ricca storia di esperienze comuni che ci permettono di interagire grazie a un patrimonio collettivo di conoscenze.

Le macchine seguono di solito regole di comportamento rigide, piuttosto semplici. Se sbagliamo nel seguirle, anche di poco, la macchina fa quello che le diciamo, per quanto insensato e illogico sia. Noi esseri umani siamo creativi, dotati di immaginazione e pieni di buon senso, ovvero un abbondante patrimonio di sapere accumulato in anni di esperienza. Le macchine, però, ci obbligano a una grande precisione, cosa a cui non siamo avvezzi. Le macchine non hanno né flessibilità né buon senso e, spesso, gran parte delle regole seguite da una macchina è nota solo alla macchina stessa e ai suoi progettisti.

Quando non si eseguono queste sue regole segrete e bizzarre, e la macchina fa la **cosa sbagliata**, la colpa viene scaricata su chi la manovra, accusato di non capirla e di non seguirne i rigidi protocolli. Con gli oggetti di uso comune, il risultato è la frustrazione, con i dispositivi complessi o processi industriali e commerciali, le conseguenze possono essere incidenti anche mortali.

È tempo di **ribaltare la situazione, di accusare le macchine e la loro progettazione**. La colpa è delle macchine e di chi le ha progettate: sta a loro capire le persone, non a noi capire i loro dettami arbitrari e insensati.

Le ragioni delle carenze nell'interazione uomo-macchina sono numerose.

Alcune nascono dai limiti della tecnologia attuale, altre da limitazioni intenzionali dei progettisti, spesso per abbassare i costi di produzione. Ma la maggior parte dei problemi deriva dalla totale incomprensione dei principi di design necessari per un'efficiente interazione uomo-macchina. Perché questa deficienza? Perché la progettazione è opera per lo più di ingegneri esperti di ingegneria ma non di psicologia, quindi limitati nella comprensione delle persone.

"Siamo uomini anche noi" pensano, *"quindi siamo in grado di capire i nostri simili".*

I tecnici fanno l'errore di pensare che sia sufficiente la spiegazione logica: *"Basterebbe che leggessero le istruzioni e andrebbe tutto bene"*.

Gli ingegneri sono formati a un tipo di pensiero logico, di conseguenza finiscono per credere che tutti debbano pensare in quel modo e progettano le loro macchine di conseguenza.

**Dobbiamo accettare il comportamento umano
per quello che è, e non come vorremmo che
fosse.**

Capitolo 4

Human Centered Design

Le persone sono frustrate dalla complessità degli oggetti quotidiani. Dalla complessità sempre maggiore del cruscotto dell'auto, alla crescente automazione della casa, con le sue reti interne. La proliferazione di sistemi complessi per il tempo libero e la comunicazione (e.g. video, audio, giochi elettronici) e le cucine sempre più tecnologiche; la vita di tutti i giorni sembra a volte una battaglia infinita contro la confusione, gli errori continui, la frustrazione, e un ciclo interminabile di aggiornamento e manutenzione degli apparecchi.

La soluzione è il **Design Antropocentrico** o **Human Centered Desing** o **HCD**, un'impostazione che parte dai bisogni, dalle capacità e dai comportamenti umani, adattando la progettazione a quei bisogni, quelle capacità e quei comportamenti. Lo HCD è un approccio di design specificamente orientato allo sviluppo di sistemi interattivi con l'obiettivo di produrre sistemi utili, altamente usabili e che si **focalizzano sull'utente**.

Il metodo è orientato all'efficienza ed all'efficacia, per aumentare la soddisfazione dell'utente ed evitare il più possibile gli effetti negativi.

Prima l'utente, poi le features! Lo HCD mette i bisogni, comportamenti e capacità umane prima di tutto, e progetta in funzione di esse.

Il problema principale delle UI è la comunicazione, in particolare la comunicazione dalla macchina verso la persona; **una buona interfaccia sa comunicare con l'utente**.

Progettare interfacce che funzionano fintanto che le cose vanno bene è relativamente facile, ma **la comunicazione è ancora più importante quando le cose non vanno bene**. È qui che i progettisti devono concentrare l'attenzione, sui casi in cui le cose vanno storte, non su quelli in cui le cose funzionano secondo i piani. Si focalizza l'attenzione soprattutto nel **comunicare ciò che è andato storto**: bisogna guidare l'utente frustrato alla risoluzione del problema poiché, in caso di **risoluzione**, proverà una **sensazione positiva** di successo per aver capito cosa non funzionava e per aver risolto il problema. Ciò crea **empatia con il sistema**.

Evitare quindi la frustrazione e aiutare a risolvere quando insorge un problema sono i concetti chiave dello HCD. Lo HCD è una filosofia di design che parte dalla **comprensione delle persone e dei bisogni che intende soddisfare**. Questa comprensione deriva dall'osservazione e dallo studio delle persone che spesso sono inconsapevoli dei loro veri bisogni e magari nemmeno delle difficoltà che incontreranno.

Per capire l'utente serve studio e osservazione della persona stessa. Non è sempre possibile tale osservazione, per cui versioni alfa e beta di un certo sistema non servono solo a fare debugging, ma servono anche per capire che cosa fa e come si comporta l'utente: diventa utile avere statistiche sull'utilizzo effettivo del sistema (e.g. quanti click su un determinato pulsante, quante volte una determinata procedura, ecc...)

Ottenere le specifiche dello HCD è quindi una delle parti più difficili del design, al punto che il principio è quello di evitare di specificare il più al lungo possibile e procedere con ripetute approssimazioni: si esegue una specifica ad alto livello, se ne implementa una parte, si testa sull'utente finale e tramite il suo feedback, modifico la parte implementata e testo di nuovo. Quando si ritiene buono ciò che è stato implementato si congela e si passa ad implementare una nuova parte.

Il ruolo dello HCD nel design	
Experience design	Area di focus
Industrial design	Area di focus
Interaction design	Area di focus
Human Centered Design	Il processo che assicura che la progettazione incontra i bisogni e le capacità degli utenti che useranno il sistema

Possiamo progettare per il design industriale, per l'interazione e per l'esperienza utente. Lo **HCD non è un'area di focus ma è un metodo**.

4.1 Desing Thinking vs HCD

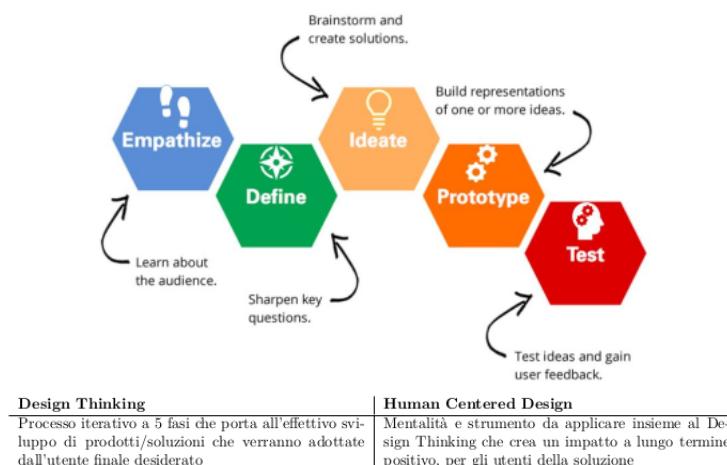
Insieme al termine HCD, spesso si può vedere il termine **Design Thinking**. I due termini vengono da scuole di pensiero molto valide ma con visioni diverse.

Il Design Thinking segue il filone Stanford (dove è nato): è un **processo di desing** con cui progettare nuovi prodotti che verranno effettivamente adottati dalle persone. Come processo è più vicino alla **disruptive innovation** che all'antropocentricità.

Si suddivide in cinque fasi iterative:

- **Empathize**: studiare il proprio pubblico, progettare il prodotto in modo che stabilisca un collegamento empatico con l'utente.
- **Define**: delineare meglio le domande chiave, cioè quali sono i bisogni a cui assolvere.
- **Ideate**: brainstorming, creare soluzioni.
- **Prototype**: costruire una o più idee.
- **Test**: testare le idee e ricevere un feedback.

Lo HCD è un mindset che viene sovrapposto al Design Thinking: identificato il modello di business, si può applicare lo HCD per assicurare che il prodotto soddisfi effettivamente le esigenze delle persone che lo andranno ad utilizzare.



Capitolo 5

Principi Fondamentali dell'Interazione

I bravi designer producono esperienze piacevoli! Ai tecnici non piace molto la parola *esperienza* poiché troppo soggettiva. Ma se si interrogasse un ingegnere sulla sua automobile preferita descriverà modello e finiture, la sensazione di potenza nell'accelerazione, la maneggevolezza del cambio e dello sterzo, ecc; queste sono esperienze.

L'esperienza è cruciale poiché determina la tonalità del ricordo che conserviamo delle interazioni con gli oggetti.

Quando la tecnologia si comporta in maniera inaspettata, proviamo confusione, frustrazione e rabbia: **emozioni negative**. Se invece comprendiamo il comportamento della tecnologia, abbiamo una sensazione di controllo, bravura e persino orgoglio: **emozioni positive**.

Cognizione ed emozione sono profondamente legate: se non mettiamo l'utente in un mood positivo farà più fatica ad apprendere l'interfaccia; più l'utente è arrabbiato e frustrato meno è predisposto a comprendere e riutilizzare il prodotto.

La **visibilità o discoverability** di un prodotto è il grado di facilità con cui un utente **scopre cosa fa, come funziona e che tipo di azione è possibile**. Tale visibilità è il risultato dell'applicazione di cinque concetti psicologici fondamentali: **affordance, significante, vincolo, mapping e feedback**. C'è anche un sesto principio, forse il più importante di tutti: **il modello concettuale del sistema**. Analizziamoli passo passo.

5.1 Affordance

Il termine affordance, letteralmente *invito*, indica la relazione fra un oggetto fisico e una persona, cioè la relazione fra le proprietà dell'oggetto e la capacità dell'utente di determinare in che modo tale oggetto può essere usato.

Una sedia appare fatta apposta per sostenere qualcosa quindi "invita" alla seduta. La maggior parte delle sedie è abbastanza leggera da poter essere sollevata e spostata da una singola persona ("invita", "permette" al trasporto), ma qualcuna è così pesante da richiedere l'intervento di più persone. Se però un certo gruppo di individui non ha la forza di sollevare una sedia, per loro la sedia non presenta l'affordance "sollevamento e trasporto".

Una affordance **non è una proprietà ma è una relazione tra un oggetto e una persona**, dipende quindi dalle proprietà sia dell'oggetto che della persona.

Si può anche parlare di **anti-affordance** nel concetto di **prevenzione dell'interazione**. Un ottimo esempio sono gli spunzoni per evitare che piccioni o altri tipi di volatili si posino in un cornicione: prevengono l'affordance di sedersi che il cornicione ha verso i piccioni. Le affordance e le anti-affordance **devono essere discoverable e perceivable**.

Questo fatto non è scontato: il vetro, famoso per la sua relativa invisibilità, occulta l'anti-affordance di precludere il passaggio.

Se uno di questi inviti o impedimenti all'uso non è percepibile c'è bisogno di qualche mezzo per segnalarne la presenza: il significante.

N.B. È assolutamente sbagliato dire "metto un affordance". Posso dire "metto un significante" ma solo se ho un'affordance.

5.2 Significante

I progettisti hanno dei problemi pratici: hanno bisogno di sapere come rendere comprensibili gli oggetti che creano. Lavorando sulla grafica degli schermi elettronici, dovevano trovare il modo di indicare quali parti potevano essere sfiorate, battute, scivolate in su o in giù o di lato, azioni che si potevano eseguire con le dite, con lo stilo o con il mouse.

Un significante è quindi **un modo per indicare dove effettuare un'azione**, dato un'affordance che determina quali azioni sono possibili.



- **Affordance:** cosa posso fare, quale azione posso compiere.
- **Signifier:** dove poso fare l'azione.

Molto spesso i significanti **sono indispensabili** poiché la maggior parte delle affordance sono invisibili. Per fare un esempio basti pensare alle porte scorrevoli: se i cardini non sono visibili, quando si vede la maniglia la prima azione che una persona tenta di fare è quella di spingere o tirare la porta, ma essa non si muoverà; è quindi necessario mettere un significante (e.g. un cartello o una scritta) che indica quale azione è necessaria per poter aprire la porta.

I significanti posso essere:

- **Voluti o intenzionali:** come un'etichetta, una stringa, un'icona.
- **Accidentali o non intenzionali:** come ad esempio un sentiero tracciato da persone che camminano attraverso un campo o delle persone in fila alla stazione.

Nel design i **significanti sono molto più importanti delle affordance**, perchè comunicano come usare il prodotto o l'interfaccia. Ma come si può associare l'affordance e il significante ad azioni reali? Nella maggior parte dei casi tramite **convenzioni**. La comprensione di un'affordance percepita è dovuta anche alle convenzioni culturali.



Figura 5.1: La scritta pull è un significante, data l'affordance della porta di essere spinta o tirata.

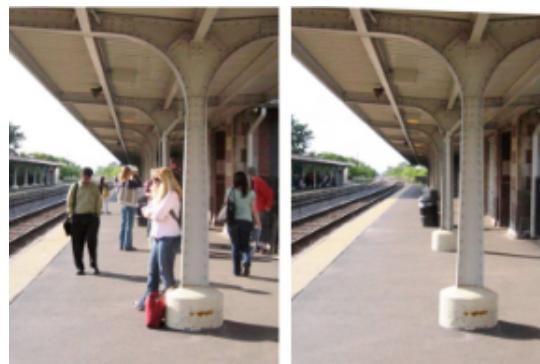


Figura 5.2: Le persone che aspettano il treno sono un esempio di significante sociale.

5.3 Mapping

Mapping è un termine tecnico, ripreso dalla matematica, che indica la relazione fra gli elementi di due insiemi.

Il concetto di mapping è di grande importanza nel progettare le interfacce e stabilire i significanti. La disposizione dei significanti può comunicare di più circa l'interfaccia e circa le sue funzionalità. Infatti quando il mapping usa una corrispondenza spaziale fra la collocazione dei comandi e quella dei dispositivi comandati, è facile capire come usarli.

Il modo migliore per applicare il mapping è quello **naturale**, perché è un'attività in cui il nostro cervello è molto bravo, i bambini imparano a fare mapping fin dai primi anni di vita. È da tenere presente che il concetto di **naturale** è ben diverso dal concetto di **universale**, poiché ci possono essere molti mapping che sembrano "naturali" ma sono specifici a una cerchia di culture.

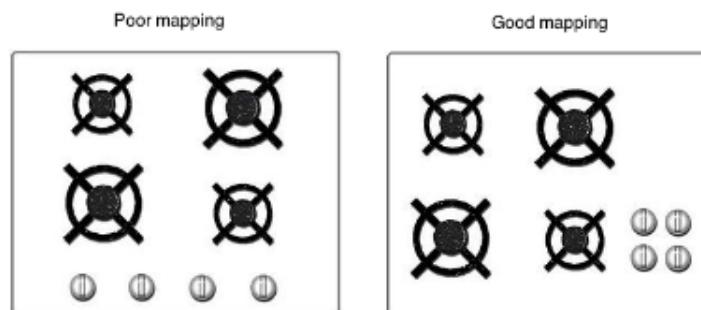


Figura 5.3: Mapping cattivo e mapping buono.

5.4 Feedback

Il feedback è la comunicazione del risultato di un'azione, di una risposta dell'interfaccia verso l'utente.

Il feedback **dove essere immediato**, anche un ritardo di un decimo di secondo può essere troppo, se il ritardo è troppo lungo l'utente potrebbe rinunciare all'attività che stava compiendo con quel prodotto e passare ad altro o addirittura non riuscire a comprendere l'origine del feedback.

Deve essere informativo, non deve portare con se troppa informazione, ma deve assolvere al proprio obiettivo, deve far capire che un'azione è in corso o che è stato prodotto il risultato che ci aspettiamo. Un scarso feedback può essere peggio di nessun feedback, perché distrae, crea confusione e quindi frustrazione da parte dell'utente.

Altro fattore importante è **la semplicità**, ovvero un feedback non deve essere pedante: troppi annunci o segnali portano le persone ad ignorarli in modo da perdere anche quei feedback cruciali e importanti. Il feedback deve essere **essenziale** e mantenere l'ambiente calmo e tranquillo.

5.5 Modello concettuale

Un modello concettuale è una descrizione altamente semplificata delle funzionalità di un sistema; non deve essere completa o accurata ma utile. I file, le cartelle e le icone che vediamo sullo schermo del computer ci aiutano a creare un modello concettuale dei dati in memoria o delle applicazioni disponibili. In realtà il computer non contiene fascicoli o cartelle: esse sono solo concettualizzazioni ideate per facilitarne l'uso.

I modelli semplificati sono preziosi e utili fintanto che le ipotesi che li supportano sono vere.

Nel Cloud Storage Sync i file sembrano essere sul dispositivo, ma in molti casi il materiale è nel cloud. Il modello concettuale è quello di un archivio disponibile sui dispositivi degli utenti. Questo modello semplificato è utile per il normale utilizzo, ma se il collegamento dei servizi si interrompe, può nascere confusione: l'informazione è sempre presente sullo schermo, ma non possiamo più salvarla o recuperare altri dati, cosa inspiegabile in relazione al modello concettuale precedentemente citato.

Il modello concettuale è **come il designer vuole che l'utente percepisca il prodotto**; sarebbe l'ambizione di progettare e comprendere la UX.

Una volta che i progettisti hanno pensato e progettato il modello concettuale si implementa l'interfaccia, in modo che il modello concettuale venga veicolato all'utente tramite affordance, significanti e mapping presenti su essa.

Quando una persona si interfaccia con il sistema o il prodotto sviluppa un suo modello mentale. Un **modello mentale** è un modello concettuale nella mente dell'utente che rappresenta il modo in cui, secondo lui, funzionano le cose. Non solo persone diverse possono avere modelli mentali diversi dello stesso oggetto, ma la stessa persona può avere molteplici modelli, pertinenti ciascuno a un aspetto diverso del suo funzionamento, e persino contraddittori gli uni con gli altri.

Più è grande la differenza tra il modello mentale e quello concettuale, più l'utente farà fatica ad usare il sistema.

L'ideale è che l'utente apprenda un modello concettuale giusto **direttamente dal device che utilizza**, senza andare a leggere manuali o istruzioni o, peggio ancora, trasmessi da persona a persona. La comprensione di un dispositivo tramite passaparola porta all'effetto del "telefono senza fili": l'interpretazione cambia da persona a persona. Per questo vi è necessità che il modello concettuale trasmesso dal prodotto sia pressoché univoco con quello mentale che l'utente si è fatto.

In questo contesto vale l'affermazione *less is more* secondo cui se una feature è difficile da veicolare allora è meglio non implementarla.

5.6 Immagine di Sistema

Le persone si creano di continuo modelli mentali di sé, degli altri, dell'ambiente degli oggetti con cui interagiscono: modelli concettuali formati attraverso l'esperienza, l'addestramento e l'istruzione.

Questi modelli ci servono da guida per realizzare i nostri scopi e comprendere il mondo in cui viviamo.

Come ci formiamo un modello concettuale adeguato dei dispositivi che utilizziamo? Non potendo parlare con il progettista, ci basiamo su tutta l'informazione accessibile: l'aspetto dell'apparecchio, cosa abbiamo imparato dall'uso di oggetti simili in passato, cosa ci dicono le pubblicità, i venditori, i pieghevoli illustrativi, il sito web e il libretto di istruzioni. **L'insieme di tutta questa informazione è l'immagine di sistema.**



Come illustrato nella figura, il progettista e l'utilizzatore finale del prodotto costituiscono i vertici scollegati di un triangolo. Un vertice del triangolo è occupato dal modello concettuale del progettista, cioè dalla sua concezione del prodotto in questione.

Una volta commercializzato, il prodotto si stacca dal progettista: lo vediamo isolato al secondo vertice del triangolo.

L'immagine di sistema è tutto ciò che si percepisce dalla struttura fisica prodotta (completa di documentazione, istruzioni, significanti e ogni informazione accessibile dal sito web o dal servizio di assistenza clienti).

Il modello concettuale dell'utente deriva dall'immagine di sistema, mediante l'interazione con prodotto, letture, ricerca online e manuali. Il progettista si aspetta che il modello concettuale dell'utente coincida col suo, ma, non essendoci comunicazione diretta fra lui e l'utente, tutto il peso della comunicazione grava sull'immagine di sistema.

Questo spiega perché la comunicazione è un aspetto importante del buon design. **Per quanto sia geniale il prodotto, se la gente non riesce ad usarlo l'accoglienza sarà cattiva.** Tocca al progettista fornire l'informazione adeguata a renderlo comprensibile e usabile. Quel che più conta è presentare un modello concettuale capace di guidare l'utente quando le cose non vanno come dovrebbero.

Un buon modello concettuale è la chiave per avere prodotti comprensibili, di facile uso e gradevole. La buona comunicazione è la chiave per ottenere buoni modelli concettuali.

Capitolo 6

Constraints, Discoverability e Feedback

In che modo si riesce a capire una cosa che non abbiamo mai visto prima?

Non c'è altro da fare che combinare l'informazione presente nel mondo esterno con quella che abbiamo in testa.

L'insieme di conoscenze che troviamo nel mondo comprende le affordance, i significanti visibili, le corrispondenze fra quelle parti degli oggetti che sembrano comandi o punti da manipolare, le azioni risultanti e i vincoli fisici, che limitano ciò che è possibile fare.

La conoscenza che abbiamo in mente comprende i modelli concettuali, i vincoli culturali, semantici e logici del comportamento, le analogie fra la situazione attuale ed esperienze precedenti.

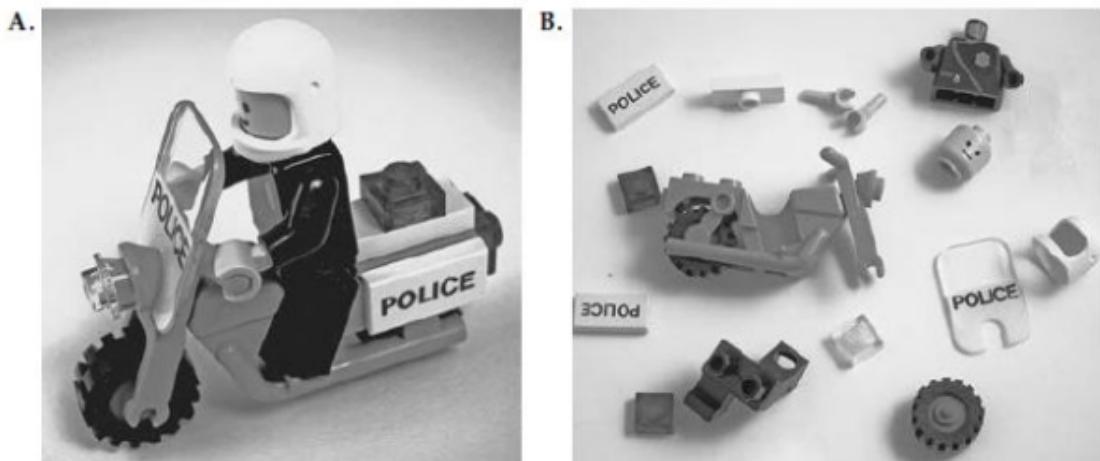


Figura 6.1: Un modellino lego.

Prendiamo come esempio il modellino Lego presente in figura: ha 15 pezzi, solo alcuni specializzati, molti altri sono di grandezza e forma uguale ma di colori diversi. Ma combinando i vincoli fisici con quelli culturali, semantici e logici si riesce a costruire il modellino senza istruzioni, mettendo ogni pezzo nella sua giusta posizione.

Vincoli fisici limitano le parti che possono andare insieme, i vincoli culturali e semantici impongono restrizioni precise a tutti i pezzi restanti e se rimane fuori qualche pezzo l'incastro è dettato dalla logica.

I vincoli sono indizi potenti, che limitano l'insieme delle azioni possibili. L'uso intelligente dei vincoli in sede di design permette alle persone di decidere prontamente il giusto corso d'azione, anche in una situazione del tutto nuova.

Possiamo categorizzare i vincoli in **quattro** classi:

- **Vincoli fisici:** si affidano a proprietà del mondo fisico, senza alcun bisogno di istruzioni o di addestramento. Nell'esempio della motocicletta Lego ritroviamo questo vincolo nei pezzi che si incastrano solo in un determinato verso.
- **Vincoli culturali:** si affidano alle abitudini culturali, sociali, comportamentali che possono cambiare nel tempo. Con vincolo culturale si intendono anche le convenzioni. Nell'esempio della motocicletta Lego ritroviamo questo vincolo nel saper determinare la collocazione delle luci: bianco all'anteriore e rosso al posteriore.
- **Vincoli semanticci:** si affidano al significato della situazione per circoscrivere l'insieme delle azioni possibili, si basano sulla conoscenza della situazione e del mondo. Nel caso della motocicletta, c'è un'unica collocazione sensata per il motociclista, deve stare seduto guardando in avanti.
- **Vincoli logici:** dettati dalla semplice e pura logica umana. Se avanzasse un un solo pezzo per assemblare la motocicletta, grazie alla logica sapremo dove collocarlo nella sua giusta posizione.

Un buon designer può sfruttare questi vincoli per veicolare l'utente verso un modello mentale del prodotto che si avvicini il più possibile al modello concettuale desiderato ed in tal modo garantirgli una UX gradevole.

6.1 Vincoli e mapping

Vincoli e mapping a volte si confondono tra di loro. Una serie di interruttori mappati in maniera opportuna danno un vincolo logico che permette all'utente di non sbagliare, si intuisce perfettamente cosa verrà azionato da quell'interruttore posto in quel determinato punto. **Mapping forti diventano quasi dei vincoli logici.** L'assenza di vincoli e mapping genera, come detto più volte, frustrazione poiché crea una interfaccia poco chiara e difficile da comprendere.



Figura 6.2: Un interruttore per le luci di una stanza che imita la piantina del locale.

6.2 Funzioni Obbliganti

Le funzioni obbliganti sono una forma di vincolo fisico: consistono di situazioni in cui le azioni sono vincolate in modo che un passaggio mancato impedisce di procedere al successivo.

Sono il caso estremo di vincoli per impedire un comportamento inappropriato.

Non tutte le situazioni permettono l'intervento di vincoli così forti, ma il principio generale si applica negli ambiti più diversi.

Esaminiamo tre di questi metodi per applicare funzioni obbliganti:

- **Interlock:** obbliga a eseguire le operazioni nella sequenza dovuta. Usati soprattutto nell'ambito della sicurezza. **Per compiere un task si deve eseguire una serie di passi.**
- **Lock-in:** mantiene attiva una funzione impedendo che qualcuno la interrompa prematuramente. Usato molto in ambito informatico (e.g. ogni tentativo di uscita da un'applicazione senza salvare è prevenuto da un messaggio di allerta che chiede la conferma dell'intenzione). **Per finire un task si deve compiere un'azione.**
- **Lockout:** impedisce l'ingresso in uno spazio pericoloso o impedisce che succeda qualcosa. Può essere considerato l'opposto del lock-in. Un esempio di stampo informatico, sono gli alert "VM 18" dove dobbiamo dichiarare la maggiore età che si possono trovare su alcuni siti. **Per iniziare un task si deve compiere un'azione.**

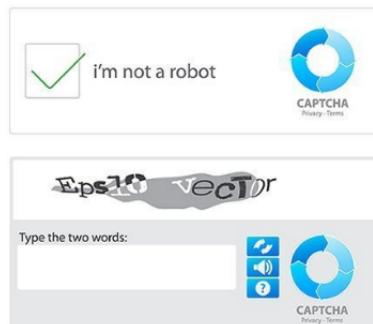


Figura 6.3: I captcha sono un esempio di interlocks.

6.3 Activity-Centered Control

Il mapping spaziale dei comandi non sempre è il più opportuno.

In molti casi è meglio avere interruttori diversi per attività diverse: **comandi centrati sulle attività.**

Quindi azionando un semplice comando si imposta una serie di oggetti per svolgere una determinata attività, senza comandarne uno per uno. In molti auditorium ci sono interruttori con indicazioni "video", "computer", "piena luce", "lezione" che impostano il microfono, le luci della sala, il proiettore e quant'altro, nel miglior modo per svolgere l'attività selezionata.

Questo schema è eccellente in teoria, ma nella pratica è difficile da realizzare bene, soprattutto è necessario valutare gli imprevisti e le possibili risoluzioni.

Il metodo è giusto, purché la gamma di attività sia scelta in modo da rispondere alle situazioni reali. Ma anche in quel caso saranno pur necessari dei comandi manuali, perché si presenteranno sempre esigenze inattese, che richiederanno una regolazione particolare dei dispositivi.

Capitolo 7

How do people do things

È facile imparare alcune azioni elementari per far funzionare un dispositivo tecnico. Ma cosa succede se le cose non vanno come dovrebbero? Come può l'utente accorgersene, e scoprire cosa fare?

Per chiarire meglio tutto questo è bene soffermarsi sulla psicologia umana e su un semplice modello concettuale dei modi in cui si sceglie e si valutano le nostre azioni. Da qui si passerà a esaminare il ruolo della cognizione e delle emozioni: piacere quando le cose funzionano senza intoppi, frustrazione quando le nostre aspettative iniziali sono bloccate.

7.1 I Golfi dell'Esecuzione e della Valutazione

Quando usiamo un oggetto, ci troviamo davanti due golfi: il **golfo dell'esecuzione**, nel quale cerchiamo di indovinare come funziona e cosa fare, e il **golfo della valutazione**, in cui si tratta di capire cosa succede. Il compito del progettista è aiutare le persone a superare i due golfi e renderli il meno profondi possibili.

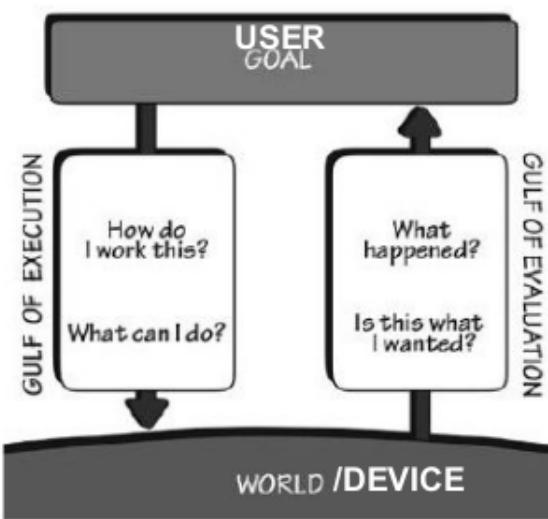


Figura 7.1: Golfo dell'esecuzione e golfo della valutazione.

Il **golfo della valutazione** corrisponde allo sforzo necessario per interpretare lo stato fisico del dispositivo e capire fino a che punto sono realizzate aspettative e intenzioni. Il Golfo è stretto quando il dispositivo fornisce informazioni sul proprio stato, in una forma facile da cogliere e interpretare, e corrispondente all'idea che abbiamo del suo funzionamento.

Quali sono gli elementi progettuali più importanti per superare il golfo della valutazione?

Il feedback e un modello concettuale adeguato.

Quali sono gli elementi progettuali più importanti per superare il golfo dell'esecuzione?

Significanti, constraints, mapping e un modello concettuale.

Entrambi i golfi sono presenti in molti apparati. Si incontrano spesso difficoltà, ma ogni volta vengono liquidate accusando se stessi. Di fronte a queste cose che ci si aspetterebbe di saper usare, si conclude semplicemente con *"sono stupido"*. Oppure, con dispositivi dall'aspetto più complicato, semplicemente ci si arrende, pensando di essere incapaci di utilizzarli. Queste spiegazioni sono entrambe sbagliate. **Le difficoltà hanno origine nel desing, non nell'utente.**

7.2 I sette stadi dell'azione

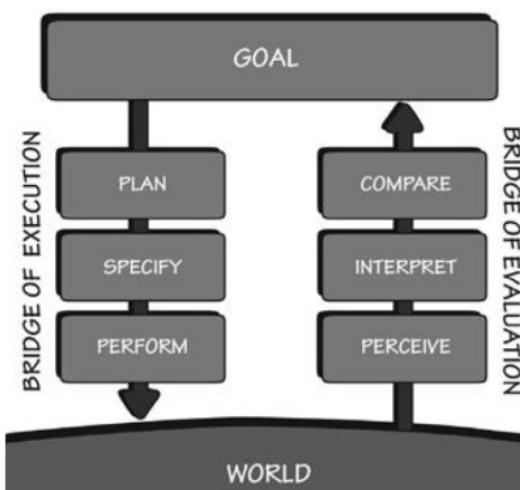
Un'azione implica due fasi: **eseguirla e valutarne gli effetti, fare e interpretare**. Sia l'esecuzione che la valutazione richiedono che si capisca come funziona una cosa e quali risultati produce. Entrambe le fasi influiscono sul nostro stato emotivo.

Le azioni specifiche fanno da ponte fra ciò che vorremmo veder realizzato e tutte le possibili azioni fisiche per arrivarci. Una volta specificato quali azioni compiere, dobbiamo effettuarle concretamente: questo è lo **stadio dell'esecuzione**. Dallo scopo discendono i tre stadi dell'esecuzione: **pianificare, specificare ed eseguire**.

La valutazione si articola anch'essa in tre stadi: **percepire, interpretazione, confrontare**.

Ecco così che abbiamo i **sette stadi dell'azione**: uno per lo scopo, tre per l'esecuzione e tre per la valutazione.

- **Scopo:** definire l'obiettivo.
- **Progettare:** l'azione da eseguire.
- **Specificare:** una sequenza d'azione.
- **Eseguire:** la sequenza specificata.
- **Percepire:** lo stato del mondo.
- **Interpretare:** la percezione.
- **Confrontare:** il risultato con lo scopo.



La maggior parte delle azioni non richiede tutti i sette stadi in sequenza, ma quasi nessuna attività si risolve in un’azione singola.

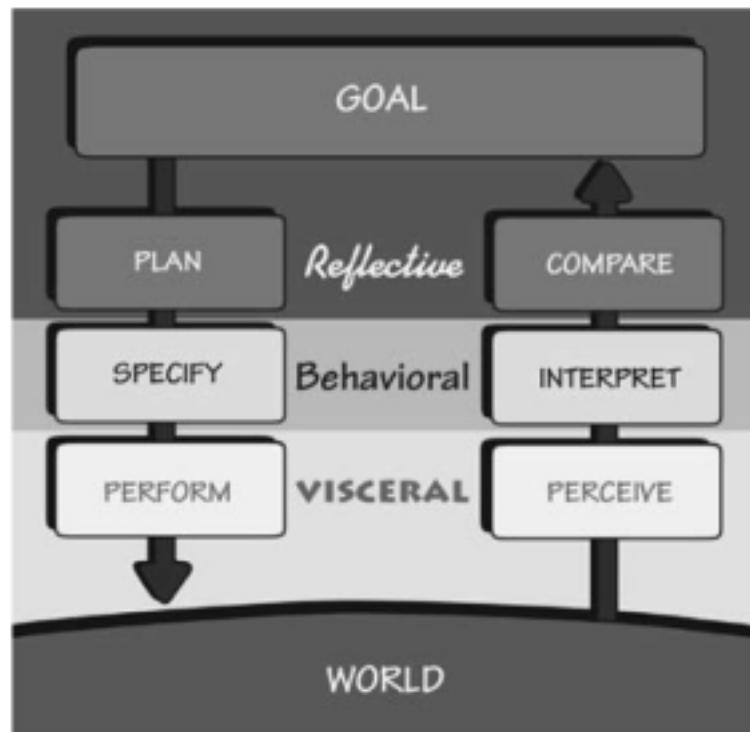
Di solito intervengono numerose sequenze e l’intera attività può durare ore o giorni. Ci sono molteplici circuiti di feedback, con cui i risultati di un’attività sono usati per indirizzare l’utente verso altre, in cui uno scopo genera scopi accessori, un progetto sotto-progetti. Ci sono attività in cui lo scopo originario è dimenticato, scartato o riformulato.

I sette stadi offrono uno schema per sviluppare nuovi prodotti o servizi. I golfi dell’esecuzione e della valutazione sono i punti più ovvi da cui partire, poiché entrambi offrono l’occasione di migliorare il prodotto. Tutto sta nello sviluppare capacità di osservazione per scoprirli.

7.3 Tre livelli di Processing

Gli stadi dell’azione possono essere associati con tre livelli di processing mentale: viscerale, comportamentale e riflessivo.

- **Livello viscerale:** è il livello più elementare che ci permette di rispondere prontamente in maniera subconscia, senza consapevolezza o controllo cosciente.
- **Livello comportamentale:** è la sede delle abilità apprese, attivate da situazioni che corrispondono al modello pertinente. In sede di design, il livello comportamentale è guidato dalle aspettative durante l’esecuzione e guidato dalle emozioni durante l’attesa di conferme di tali aspettative. Decide in che modo si compie un determinato task e in che modo si interpreta un determinato feedback.
- **Livello riflessivo:** è quello della cognizione conscia, è quindi a questo livello che si sviluppa la comprensione profonda e hanno luogo il ragionamento e i processi decisionali. Qui fanno capo i livelli più alti di emotività: è qui che avviene la soddisfazione e l’orgoglio, ma anche la frustrazione e il senso di colpa.



Per il progettista il livello riflessivo è forse il più importante dei tre. La riflessione è consci e le emozioni che si producono a questo livello sono le più durature: quelle che attribuiscono responsabilità agli agenti casuali, come colpa, vergogna e orgoglio.

Le risposte riflessive sono anche parte integrante del ricordo degli eventi, che dura assai più a lungo dell'esperienza immediata o del periodo d'uso, che sono invece ambiti del livello viscerale e comportamentale.

È la riflessione che ci induce a consigliare un prodotto, a raccomandare l'uso o magari a sconsigliarlo.

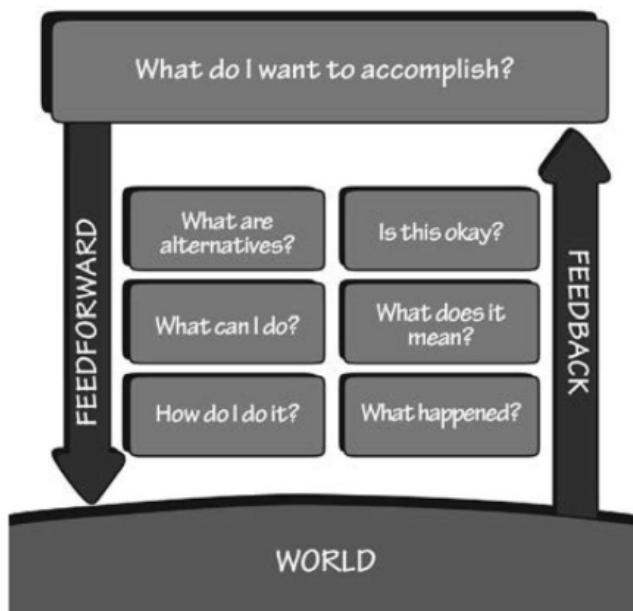
I tre livelli di elaborazione contribuiscono tutti insieme a determinare il nostro stato emotivo e cognitivo. Funzioni riflessive di alto livello possono mettere in moto le emozioni più elementari, così come queste possono stimolare attività cognitive di tipo riflessivo.

7.4 I sette Principi Fondamentali della Progettazione

Il modello in sette stadi del ciclo d'azione può essere un prezioso sussidio per il design, in quanto suggerisce una lista di domande fondamentali. In generale, ogni stadio dell'azione richiede specifiche strategie progettuali, e, viceversa, presenta occasioni tutte sue di disastro.

Derivano quindi sette domande, a cui dovrebbe poter rispondere chiunque stia usando un determinato prodotto.

- Cosa voglio ottenere?
- Quali sono le sequenze d'azione alternative?
- Quale azione posso fare ora?
- Come faccio questa azione?
- Cosa è successo?
- Cosa significa?
- Va bene? Ho realizzato il mio scopo?



Il progettista ha la responsabilità di garantire che a ogni stadio dell'azione il prodotto fornisca l'informazione necessaria per la risposta.

L'informazione che serve a rispondere alle domande sull'esecuzione è il **feedforward**.

L'informazione che aiuta a capire quello che è successo è il **feedback**.

Il **feedforward** si realizza mediante l'uso opportuno di significanti, vincoli e mapping, anche il modello concettuale ha un ruolo importante. Il **feedback** è dato dall'informazione esplicita circa l'impatto dell'azione eseguita e anche qui una parte importante è svolta dal modello concettuale.

Sia il **feedback**, che il **feedforward** devono presentarsi in una forma facilmente interpretabile da chi utilizza il sistema. La presentazione deve corrispondere alla visione che le persone hanno dello scopo che vogliono realizzare e alle loro aspettative. L'informazione deve essere congruente con le esigenze umane.

Dalle risposte relative ai sette stadi dell'azione si ricavano sette principi fondamentali del design:

- **Visibilità:** è possibile scoprire immediatamente quali azioni sono possibili e qual è lo stato attuale del dispositivo.
- **Feedback:** c'è un'informazione completa e continua riguardo ai risultati delle azioni e allo stato attuale del prodotto o del servizio. Dopo aver eseguito un'azione, è facile determinare il nuovo stato.
- **Modello Concettuale:** il design fornisce tutta l'informazione necessaria per creare un buon modello concettuale del sistema, che favorisca la comprensione e la sensazione di controllo. Il modello concettuale potenzia sia la visibilità, sia la valutazione dei risultati.
- **Affordance:** affordance corrette sono fatte apposta per rendere possibili le azioni desiderate.
- **Significant:** un uso efficace dei significanti assicura la visibilità e un feedback efficiente e intellegibile.
- **Mapping:** la relazione fra i comandi e le rispettive azioni obbedisce ai principi del buon mapping, sostenuto, per quanto possibile, dalla disposizione spaziale e dalla contiguità temporale.
- **Vincoli:** fornire vincoli fisici, logici, semantici e culturali guidando l'azione e facilitandone l'interpretazione.

Questi sette principi sono mappati uno a uno sugli stadi d'azione dell'utente.

È bene concludere con una nota la parte dedicata a strumenti, metodi ed elementi per il design dello human-computer interaction: per molte attività quotidiane , gli obiettivi e le interazioni non sono ben definiti, **sono più di tipo opportunistico che pianificato**.

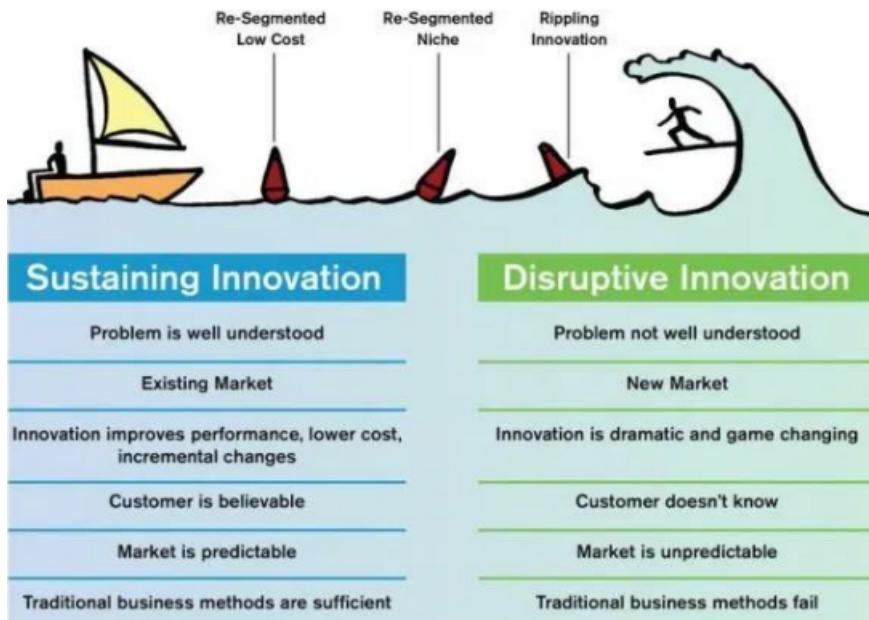
Le azioni opportunistiche sono quelle in cui il comportamento scaturito dalle circostanze prevale sulla pianificazione. Gli utenti in questi casi agiscono in maniera non controllata e quindi non prevedibile.

È difficile fare buon design per queste situazioni, anche attenendosi a tutti i principi esposti fino ad ora: l'utente che agisce in maniera opportunistica romperà questi schemi.

7.5 Disruptive Innovation

"People don't want to buy a quarter-inch drill. They want a quarter-inch hole!"

La maggior parte dell'innovazione è fatta come miglioramento incrementale di prodotti già esistenti. La **disruptive innovation** riguarda l'innovazione non lineare, consiste in un completo salto di binario. Vuole introdurre idee radicali che portano a progettare nuove categorie di prodotto nel mercato.



Ciò è effettuato riconsiderando l'obiettivo tramite un procedimento chiamato **root cause analysis**, che può essere decomposto in quattro fasi.

La **root cause analysis** è quindi fatta:

1. Descrivendo chiaramente il problema.
2. Tracciando il flusso degli eventi che ha fatto in modo che da una situazione di normalità emergesse un problema.
3. Distinguendo le cause principali dalle cause secondarie.
4. Tracciando un grafico che descrive le relazioni tra le cause principali e il problema.

7.6 Agile

Si supponga che la progettazione non sia un problema e che si possegga l'analisi completa di un processo, come fare ad innovare?

L'**Agile** è un modo di pensare che rientra nella categoria del **Design Thinking**. È un **metodo di lavoro**: il Design Thinking è un modo di porsi nella soluzione dei problemi, mentre invece l'**Agile** è il modo in cui si applicano i metodi del Design Thinking per sviluppare software, il primo infatti porta a identificare una soluzione, l'**Agile** invece è un **metodo strutturato per lo sviluppo di software**. Nell'**Agile** il focus è posto sugli individui e l'interazione con essi piuttosto che sui processi e sui tools. Il risultato è un software che funziona invece che grandi e verbosissime documentazioni. Uno dei punti fondamentali è lo sviluppare capacità di adattamento al cambiamento. Le regole non sono dettate dai programmatore o da un capo che prende le decisioni, tutto è guidato dal processo di design. Il grande punto di forza dell'**Agile** rimane comunque quello dell'**adattabilità**. La mancanza di struttura o documenti non è anarchia, anzi è fulcro di grande collaborazione e sintonia tra il team.

Capitolo 8

Progettazione

Quali processi e metodi utilizzare per portare avanti il progetto di un prodotto software?

In questo capitolo si analizzeranno diversi processi. Essi non vanno identificati come fasi ben definite e statiche, da seguirsi una dietro l'altra, bensì come **fasi dinamiche e alternabili**.

8.1 Personas

Identificato correttamente il problema mediante la **Task Analysis**, come si creano elementi individuali? Come si identificano le così dette le **Personas**?



Il primo passo da fare dopo la Task Analysis è **identificare le personas**. Una **personas** è l'**archetipo di uno dei possibili utenti**. Scrivere e identificare le personas aiuta a colmare la distanza tra il cliente e l'azienda, in modo da capire cosa vuole e cosa si aspetta l'utente dal prodotto, ma anche cosa gli crea frustrazione nell'usarlo. Esistono molte **tecniche** per fare un analisi degli utenti che aiutano i progettisti e identificare e a scrivere le personas:

- **Task Analysis.**
- **Feedback** tra i quali: analisi delle attività, interviste e focus groups.
- **Prototipazione.**

Sorge spontanea la domanda seguente: *ma quante personas è bene definire?* Per rispondere a questa domanda ci si basa sul **Princípio di Pareto**:

Concentrarsi sul 20% degli utenti che utilizzerà il prodotto per l'80% dell'uso complessivo.

8.2 Requirements

Un **requirement** è un servizio o una caratteristica che assolve un bisogno di un utente. I requirements possono essere funzioni, vincoli, regole aziendali o altri tipi di elementi che devono essere presenti per soddisfare le esigenze degli utenti. È quasi ovvio che trovare i **requirements corretti** risulta molto più **semplice** se in precedenza sono state **individuate e descritte** le **personas**.

Infatti è controproducente scrivere anticipatamente i requirements, è complesso se non impossibile descriverli tutti all'inizio di una fase di progettazione. Risulta più facile descrivere i requirements con l'avanzamento del progetto, in modo di poter capire come modificare, eliminare o aggiungere i requirements giusti, tenendo in considerazione anche le personas a cui è destinato il prodotto finale.

Il **requirements driven development** è un approccio complesso e oneroso e va in **contrasto** con il metodo **Agile** che richiede si sia sempre pronti al cambiamento. Questo approccio così poco flessibile sta andando sempre meno di moda tuttavia è bene seguirlo una volta definiti i requirements per quel 20% di software che utilizzerà l'80% delle personas.

Esistono **due tipi di requirements** per il mondo della UI:

- **Funzionali:** descrivono quali funzionalità deve avere il software. Rispondono alla domanda: *Cosa fa?*
- **Non funzionali:** specificano i tratti qualitativi del prodotto. Rispondono alla domanda: *Come lo fa?* e descrivono inoltre attributi come sicurezza, affidabilità e manutenibilità.

8.3 User Stories

Dopo aver definito **personas** e **requirements**, il prossimo passo è la scrittura delle **User Stories**.

Una **user story** è una breve descrizione che identifica l'utente insieme al suo obiettivo, determina **chi è, di cosa ha bisogno e perché ne ha bisogno**. Tipicamente ci sono una o più user stories per ogni persona, ma **non il contrario**: avere più persone per una user story significa aver descritto la medesima persona con termini differenti e quindi averne introdotta una ridondante. Se una user story non copre tutte le sfaccettature della singola persona che dovrebbe descrivere è indizio del fatto che tale persona potrebbe essere troppo **ampia**, ed è bene che venga suddivisa in più persone.

Una user story è un **requirement espresso dalla prospettiva del cliente**.

Con le user stories si ottengono due importanti risultati: si **migliorano le descrizioni delle personas**, migliorandone il dettaglio e arricchendole con una narrativa, e si ha un elemento tangibile per valutare lo sviluppo del prodotto e fare il punto della situazione. Esiste uno schema ben preciso per scrivere una user story:

As a [role], I want [feature] because [reason]

Chi scrive le user stories?

Tutti! È responsabilità del proprietario del prodotto assicurarsi che vengano scritte delle stories giuste e corrette, ma non significa che le debba scrivere lui stesso. Nel corso di un buon progetto Agile ci si aspetta che ogni membro del team scriva user stories secondo un **metodo collaborativo**.

A che livello è bene scriverle?

Una dei grandi benefici delle user stories è che possono essere scritte a **vari livelli di dettaglio**.

Possiamo avere user story di livello **epic** ovvero in grado di coprire un grande quantità di funzionalità. Tipicamente sono disutili ma possono diventare interessanti se spaccettate in user stories più piccole e quindi con maggior livello di dettaglio. Una **user story epic** può essere **divisa** in decine o centinaia di user stories più piccole.

I dettagli possono essere aggiunti sostanzialmente in due modi:

1. Suddividendo una user story in user stories più piccole e quindi più dettagliate.
2. Aggiungendo **condizioni di soddisfacimento dell'utente**, ovvero dettagli contribuenti al grado di soddisfacimento dell'utente, spacchettandole per ogni membro di soddisfacimento. Quest'ultima è una tecnica molto difficile da utilizzare nel mondo della progettazione software.

8.4 Scenarios

Gli **scenarios** sono l'**evoluzione delle user stories**. Una user story sintetizza in bravi frasi cosa fa l'utente con il software e quale bisogno deve soddisfare.

Lo scenario estende la user story andando a descrivere anche quali motivazioni hanno portato l'utente ad usare il software e come egli si comporterà nel suo utilizzo. Inoltre gli scopi o **goals** delle personas vengono estesi e descritti in maniera più ampia.

Ma a cosa serve uno scenario?

Le **user stories** sono **sintetiche**, **brevi**, dicono cosa sviluppare e poco più, sono informative e comprensibili solo agli **addetti al mestiere**. Gli **scenarios**, essendo più ricchi in dettaglio, sono più comprensibili agli **stakeholders**, quindi sono **utili per comunicare e interloquire con persone fuori dal team di sviluppo**. Servono soprattutto per allinearsi con le richieste del cliente ma sono utili anche all'interno del team per allineare le idee di sviluppo.

Un buon scenario risponde alle **seguenti domande chiave**:

- **Chi è l'utente?**
- **Quale è la motivazione che lo ha spinto ad usare il prodotto e cosa si aspetta da esso?**
- **Qual è il suo obiettivo?**

In alcuni casi uno scenario potrebbe includere anche aspetti circa *come* fare le cose, ma ciò appartiene al mondo degli use case che verranno analizzati successivamente. Per definire gli **scenarios**, è necessario **mapparli**. Bisogna partire **avendo già le personas e le relative user stories**, e individuare per ogni persona un key task che essa deve svolgere. Si possono scrivere gli scenarios racchiudendo una o più user stories relative alla persona ma tipicamente per ogni persona si descrive almeno uno scenario. È necessario **focalizzarsi sul key task** e quindi contestualizzare nel miglior modo possibile il goal dell'utente e come è influenzato dal contesto in cui egli si trova.

Quindi grazie agli scenarios **possiamo determinare**:

- I **punti più importanti** su cui concentrarsi durante il **processo di progettazione per l'UX**.
- Quali fasi del processo richiedono **ulteriore revisione e attenzione**.
- Le **principali esigenze e motivazioni** dell'utente.

Ci sono **tre metodi principali per scrivere gli scenarios:**

- **Piccoli goal or task oriented scenarios**, brevi, sono quasi una semplice estensione della dialettica di una user story.
- **Elaborated Scenarios**: sono versione più elaborata con molto più contesto.
- **Full Scale Task Scenarios**: sono talmente dettagliati che i singoli task delle attività sono praticamente scritti sotto forma di paragrafo. Da evitare, si rischia gli scenarios così ottenuti siano uguali agli use cases.

8.5 Use Cases

Gli use cases sono la naturale evoluzione degli scenarios. **Consistono della completa narrativa di quali azioni l'utente compie per svolgere uno scenario.**

Ogni **caso d'uso** è una **serie di step monolitici**, è quindi una nuova task list prodotta dal processo di design per ampliare ulteriormente un concetto analizzato e descritto in precedenza mediante la task analysis. Uno **scenario** si concentra su una situazione che coinvolge uno o più **attori**, mentre un **caso d'uso** è incentrato su una **persona**, come essa si comporta, che decisioni prende e come interagisce con la nostra piattaforma o software. I casi d'uso aggiungono valore perché aiutano a **spiegare come dovrebbero comportarsi il sistema e il processo**, inoltre aiutano anche nella fase progettuale di brainstorming e danno un'idea su cosa potrebbe andare storto. Forniscono inoltre un elenco di obiettivi che può essere utilizzato per stabilire e valutare la complessità del sistema. I casi d'uso sono **diretti allo sviluppo**.

Con dei **casi d'uso ben fatti** si può passare direttamente all'**implementazione** poiché **delimitano precisamente cosa deve fare il sistema e come si deve comportare nelle varie situazioni**.

In un caso d'uso sono **inclusi**:

- **L'utente.**
- **Cosa vuole fare.**
- **Qual è il suo scopo o goal.**
- **Gli step necessari** per arrivare allo scopo.
- **I feedback** che il sistema restituisce durante i vari step.

Non sono inclusi in un caso d'uso:

- **Qualsiasi dettaglio implementativo o di scelta tecnologica** che non sia un constraint o un requirement.
- **Dettagli riguardanti l'interfaccia utente.**

I passaggi da seguire per scrivere un caso d'uso sono:

1. Identificare le **personas**.
2. Sceglierne **una per caso d'uso**.
3. Identificare i suoi **obiettivi**.
4. Discernere i **task principali** da quelli **secondari**: applicare il principio di Pareto, quindi focalizzarsi sul basic path e non su casi particolari.
5. Considerare le sequenze alternative.
6. Cercare i punti in comune tra i vari casi d'uso e **accorparli**, riducendoli ove possibile.
7. Ripetere il procedimento per tutti gli utenti.

Questi passaggi se seguiti correttamente consentono di identificare informazioni chiave sugli utenti, utili a costruire prodotti che soddisfano tutte le possibili personas. **Ogni passo che ci avvicina all'utente è un passo verso la giusta direzione.**

Capitolo 9

Human Error

La maggior parte degli incidenti industriali è causata da errore umano: le stime si aggirano tra il 75% e il 95%.

Come è possibile che tante persone siano così incompetenti?

La risposta è semplice! Non lo sono, il problema è nella mal progettazione e nel cattivo design.

Si continuano a produrre dispositivi e software che richiedono, a chi li usa, di mantenere per ore un'attenzione e una vigilanza complete, oppure di memorizzare procedure arcaiche, confuse e usate di rado, magari anche una sola volta nella vita. Si costringono le persone a stare in un ambiente monotono senza nulla da fare per ore e ore, salvo dovere improvvisamente rispondere con rapidità e precisione. Le si sottopone ad un ambiente di lavoro complesso e sovraccarico, dove sono continuamente interrotte durante l'esecuzione di compiti simultanei. **L'interruzione è una delle cause che più frequentemente portano all'errore umano.**

Uno dei più grandi problemi è l'**atteggiamento delle persone verso gli errori** commessi. Quando un errore causa perdite economiche o, peggio ancora, danni alle persone, si istituisce una speciale commissione d'inchiesta, che quasi immancabilmente trova i colpevoli.

Il passo successivo è punirli con multe, il licenziamento o addirittura il carcere. Essi vengono incolpati e puniti o, nella migliore delle ipotesi, incolpati e riaddestrati. Tutto questo non risolve il problema: lo **stesso errore continuerà a presentarsi**. Per evitare di incorrere nuovamente nell'errore, quando esso viene commesso, è bene studiarne il motivo, per poi ridisegnare il prodotto o le procedure in modo che esso non si ripeta o, se dovesse ripetersi, che i danni siano ridotti al minimo.

9.1 Root Cause Analysis

La **Root Cause Analysis** consiste nell'indagare l'incidente finché non si trova la singola causa che ne è l'origine. Ovvero il momento nel tempo quando effettivamente qualcuno ha preso decisioni o eseguito azioni sbagliate e, una volta fatto ciò, accertare da cosa è derivato lo sbaglio. Purtroppo, troppo spesso, il processo si ferma non appena si scopre che una persona ha agito in maniera impropria, additandola come *colpevole*.

Cercare di trovare la causa di un incidente suona bene, ma ha due difetti.

1. **Primo:** la maggior parte degli incidenti non ha una sola causa. Da qui il **modello a groviera degli incidenti** di James Reason.
2. **Secondo:** Solitamente l'analisi delle cause profonde si ferma non appena trovato un errore umano.

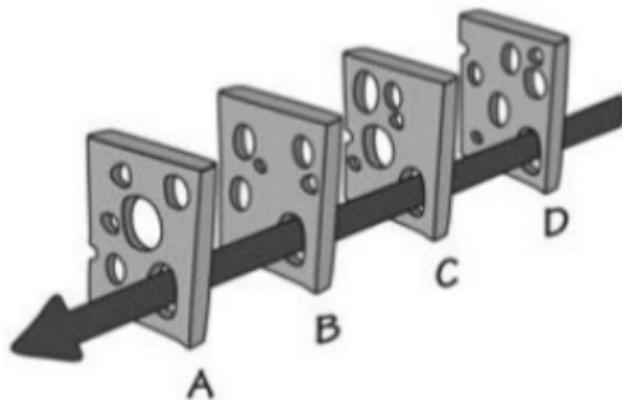


Figura 9.1: Modello a groviera.

Se una macchina smette di funzionare per un guasto o un malfunzionamento si cerca di capire come mai si è rotta o cosa l'ha portata a guastarsi. È opportuno fare lo stesso quando si scopre un errore umano: **individuarne le cause**.

Quando l'analisi delle cause profonde incontra, nella concatenazione di cause ed effetti, un errore umano, **il lavoro è appena cominciato**: bisogna capire perché l'errore è accaduto e cosa si può fare per prevenirlo.

9.2 I cinque perché

L'analisi delle cause profonde mira a determinare la causa **prima di un evento**, non la causa immediata.

In Giappone da tempo si usa a questo scopo una procedura detta "*dei cinque perché*" ideata da Sakichi Toyoda e impiegata dalla Toyota nell'ambito del sistema di controllo qualità dei suoi prodotti.

Fondamentalmente quindi quando si cerca la ragione di un evento non ci si deve fermare dopo averne trovata solo una, ma bisogna continuare ad indagare fino a che non si trovano le **vere cause di fondo**.

Va ripetuta davvero cinque volte?

No, ma chiamarla *procedura dei cinque perché* sottolinea la necessità di proseguire anche dopo aver trovato una causa apparente.

Vediamo un esempio: **il veicolo non si accende**.

1. **Perché?** La batteria è morta.
2. **Perché?** L'alimentatore non funziona.
3. **Perché?** La cinghia dell'alternatore non funziona.
4. **Perché?** La cinghia dell'alternatore era ben oltre il suo tempo di servizio e non è stata sostituita.
5. **Perché?** Il veicolo non è stato mantenuto secondo le tempistiche raccomandate.

Quando le persone sbagliano, bisogna **cambiare il sistema** in modo da evitare l'errore e, se non è possibile eliminarlo del tutto, almeno fare in modo di ridurne gli effetti.

Se il sistema lascia sbagliare gli utenti è **mal progettato**, se il sistema induce all'errore, allora è **progettato malissimo**.

Perchè le persone sbagliano?

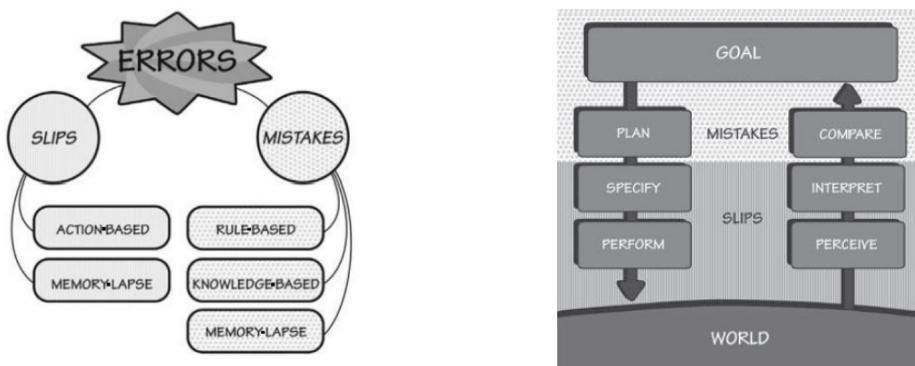
Perchè il design si concentra sulle esigenze del sistema e delle macchine, non su quelle degli utenti. Le macchine hanno bisogno in genere di comandi precisi, obbligando l'operatore a introdurre esatte informazioni numeriche. Gli esseri umani non sono adatti ad esercitare grande precisione e commettono spesso errori quando devono digitare lunghe sequenze di numeri o lettere.

Gli umani sono creativi, curiosi, costruttivi, particolarmente bravi nel creare modi nuovi di fare le cose, nel cogliere nuove opportunità. Compiti monotonici, ripetitivi, precisi contraddicono tali qualità e vi entrano in conflitto.

9.3 Definizione di errore

Si definisce **errore umano** ogni deviazione dal comportamento appropriato. Il termine appropriato è da prendere con le pinze, perché in molte circostanze si scopre quale fosse il comportamento giusto solo successivamente.

Generalmente comunque si chiama *errore* ogni comportamento che si discosta da quello generalmente accettato come giusto o adeguato. Errore è il termine generale per tutte le situazioni sbagliate. È possibile dividere gli errori in **due classi**:



1. **Lapsus o Slips:** si ha un lapsus quando s'intende eseguire un'azione e si finisce per eseguirne un'altra. Nel caso del lapsus, l'azione eseguita non è quella voluta. Ci sono **due tipi principali di lapsus: lapsus di azione e lapsus di memoria**. Nel primo caso si esegue un'azione sbagliata, nel secondo si dimentica di eseguire l'azione o di valutarne i risultati. Esempio di **lapsus di azione**: versare il latte nel caffè e poi riporre la tazza in frigorifero. L'azione è giusta ma è applicata all'oggetto sbagliato. Esempio di **lapsus di memoria**: dimenticare di spegnere il fornello dopo aver terminato la cottura. **I lapsus si hanno nelle fasi attuative e percettive dell'azione.**
2. **Errori cognitivi o Mistakes:** si ha un errore cognitivo quando è sbagliato il goal o lo scopo: da quel momento in poi le azioni, anche se eseguite a puntino, fanno parte dell'errore essendo di per sé inappropriate, in quanto parte di un progetto sbagliato. In questo tipo di errore l'azione è corretta ma l'intenzione no. Gli errori cognitivi si suddividono in: **regola sbagliata, conoscenza sbagliata e dimenticanza**. In un errore generato dall'applicazione della **regola sbagliata**, la diagnosi della situazione è giusta, ma poi viene scelto un corso d'azione sbagliato, seguendo una regola operativa errata. In un errore causato da **conoscenza erronea o incompleta**, ad essere sbagliata è la diagnosi stessa della situazione. Gli errori per **dimenticanza** si hanno invece quando ci si dimentica di qualche passaggio al momento di fissare gli obiettivi, di eseguire una procedura o di valutarne i risultati. **Gli errori si fanno nelle fasi di tipo cognitivo.**

9.4 Prevenzione dell'errore

Non dovrebbe essere possibile che un semplice errore provochi un danno diffuso.

Ecco che cosa dovrebbe essere fatto in fase di prevenzione:

- **Comprendere le cause dell'errore** per minimizzarne il ripresentarsi.
- Effettuare **controlli di sensibilità**, ovvero, chiedersi se le azioni superano il *test del buon senso*.
- Rendere possibile **annullare le azioni** (undo) o rendere più difficile fare ciò che non può essere annullato (per esempio con uso di lock).
- **Rendere più semplice la scoperta e la comprensione degli errori** e semplificare la risoluzione.
- Non trattare l'azione come errore, piuttosto **aiutare l'utente a compiere correttamente l'azione**.

I **novizi**, gli utenti base, coloro meno esperti del sistema cadono in **errori per lo più cognitivi** poiché non hanno una base di conoscenza adeguata e sufficientemente strutturata, viceversa, gli **utenti esperti** che usano il software o il sistema tutti i giorni e che lo conoscono bene fanno più errori di tipo **lapsus** poiché tendono a eseguire i compiti in maniera automatica, quasi istintiva, affidandosi al controllo subconscio, mentre un principiante è costretto a fare molta attenzione, cosicché incorre meno nei lapsus.

Gli **errori cognitivi** nascono dalla scelta di scopi e piani d'azione inadeguati, oppure, in sede di valutazione, dal confronto errato tra risultati e scopi. In altre parole dipendono da **informazioni ambigue o poco chiare sullo stato attuale del sistema e dalla mancanza di un buon modello concettuale**.

Si esamineranno adesso quali possono essere le cause di errore e come è possibile prevenirle.

Le **interruzioni** sono una delle più grandi cause di errore, **soprattutto i lapsus**. Quando un'attività viene interrotta da qualche evento, il costo in attenzione è molto maggiore della perdita di tempo causata dell'interruzione. Per riprendere il lavoro è necessario ricordare precisamente il precedente stato dell'attività: quale era l'obiettivo, a che punto del ciclo dell'azione si era rimasti, quale era lo stato del sistema.

La maggior parte dei sistemi rende difficile la ripresa di un'azione a seguito di un'interruzione. Tuttavia riducendo i passaggi dell'azione è possibile diminuire il costo per riprendere la concentrazione dopo esser stati interrotti.

Un altro fattore di errore sono i **feedback errati**: avvisi fastidiosi, non necessari che si presentano spesso nell'uso di un sistema. Spesso vengono silenziati, disattivati o ignorati, **facendo perdere di significato anche quelli utili per il raggiungimento dello scopo**.

Se si usano i feedback per segnalare errori, ma essi sono stati disattivati dall'utente, egli cadrà in errore non conoscendone nemmeno il motivo. **Avvisi e metodi di sicurezza vanno usati con cura e intelligenza**.

Un numero sempre maggiore di macchine e sistemi offrono informazioni attraverso l'uso di interfacce vocali, ma come tutti gli approcci anche questo ha dei pro e dei contro. Da una parte consente di fornire informazioni precise, specialmente quando l'attenzione visiva è diretta da qualche altra parte, ma se l'ambiente è rumoroso o se ci sono diversi avvisi vocali contemporaneamente, tali avvisi possono non essere compresi o risultare addirittura fastidiosi.

Person: Delete "my most important file."

System: Do you want to delete "my most important file"?

Person: Yes.

System: Are you certain?

Person: Yes!

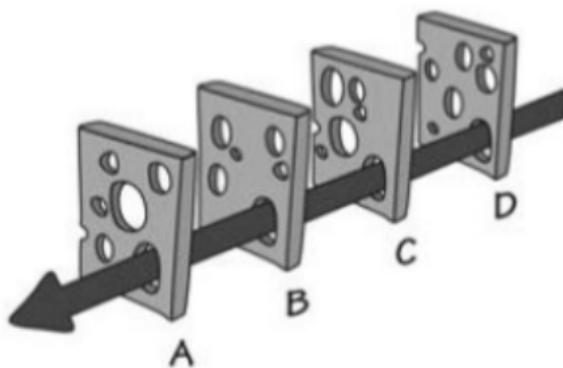
System "My most favorite file" has been deleted.

Person: Oh. Damn.

Per prevenire errori è possibile quindi utilizzare:

- **Constraints:** aggiungendo vincoli alle azioni. I sistemi elettronici hanno un'ampia selezione di metodi che possono essere usati per ridurre l'errore. Uno di questi può essere **segregare i controlli**, cosicché controlli confondibili tra loro vengano piazzati lontani l'uno dall'altro. Un altro è di **separare i moduli**, cosicché qualsiasi controllo non direttamente rilevante all'operazione corrente non sia visibile a schermo ma richieda uno sforzo extra per essere raggiunto.
- **Undo:** comando che annulla le operazioni effettuate dal precedente comando. I sistemi migliori hanno **più livelli di undoing** in modo tale da annullare intere sequenze di azioni.
- **Messaggi d'errore e di conferma:** molti sistemi cercano di prevenire l'errore richiedendo conferma prima di eseguire un comando, specialmente quando l'azione distruggerà qualcosa di importante. Ma queste richieste sono spesso mal temporeggiate, perché **dopo aver richiesto un'operazione le persone sono solitamente certe di volerla compiere**. Un controllo migliore sarebbe visualizzare sia l'azione da compiere che l'oggetto interessato, con l'opzione annulla o prosegui. **I messaggi di avviso sono sorprendentemente inefficaci contro gli errori.**
- **Controlli di Sensibilità:** i sistemi elettronici presentano il vantaggio di poter controllare che l'operazione richiesta sia **sensibile** o **ragionevole**. Ad esempio verificare che l'importo indicato sia giusto, magari esponendo un avviso in caso di numeri eccessivamente grandi.

In estrema sintesi e ricollegandosi all'esempio della groviera, per ridurre gli errori si hanno le seguenti possibilità:



- **Aumentare il numero di controlli** (le fette).
- **Migliorare il modello concettuale dell'utente.** (ridurre il numero di buchi, o rendere più piccoli i buchi esistenti, magari con un modello concettuale minimale e dei constraints)
- **Allertare l'operatore umano quando diversi buchi si allineano.**

Capitolo 10

Pretotyping



Un prodotto **pretotipo** serve a lottare contro la legge di fallimento di mercato. Ogni anno vengono progettati e prodotti quasi 25000 nuovi prodotti l'80% dei quali non vedrà mai la luce o non arriverà mai nelle case delle persone. Circa il 27% falliscono nel percorso di crescita dell'azienda, il 16% non raggiunge le aspettative degli utenti, trattasi quindi di fallimento di mercato, e per ben il 37% vengono cancellati durante la fase di lancio.

Del 20% rimanente, il 14% sono prodotti che raggiungo il mercato e ci rimangono ma non hanno successo. **Solo il 6% ha veramente successo.** La Legge del Fallimento di Mercato sostiene che la maggior parte dei nuovi prodotti fallirà nel mercato anche se la progettazione e lo sviluppo vengono eseguite in maniera corretta e competente. In legge, una persona è considerata innocente fino a prova contraria, mentre nella legge di mercato, **bisogna supporre ogni prodotto come fallito fino a prova contraria.**

10.1 Thoughtland, il mondo dei pensieri

Quando si progetta un nuovo prodotto, o si ha semplicemente un'idea ci si trova di fronte a **due grandi problemi**: il **lost in translation**, ovvero l'idea è un'astrazione soggettiva, qualcosa che si può immaginare o visualizzare in testa. Nel momento in cui si prova a comunicarla a qualcun altro, si incontra un problema di traduzione, specialmente quando l'idea è nuova ed è diversa da qualsiasi altra cosa abbia visto l'interlocutore.

Il modo in cui si immagina un nuovo prodotto ed i suoi usi può essere completamente diverso da come gli altri lo immaginano a loro volta. Si può ovviare utilizzando **termini noti**, infatti, è molto difficile veicolare il modello concettuale all'utente. L'altro problema è conosciuto come il **problema della predizione**: anche se la comprensione astratta dell'idea che l'interlocutore ha è molto vicina alla propria, egli in quanto essere umano è molto scarso per sua natura nel prevedere se essa potrebbe essere di suo gradimento o meno.

10.2 Thoughts without datas are just opinions

I pensieri senza dati sono solo opinioni.

Questa è una delle più importanti frasi da tenere presente quando si parla di nuovi oggetti o idee da immettere nel mercato, anche se è spesso sottovalutata.

I falsi positivi possono portare a credere che l'idea sia immune alla legge del fallimento di mercato, e quindi a investire troppo e presto in un prodotto che probabilmente fallirà.

I falsi negativi possono invece spaventare e portare a non concedere una chance all'idea, finendo per scartare prematuramente il prossimo Twitter, Google o Tesla.

Per poter minimizzare la possibilità di ottenere falsi positivi o negativi è necessario **uscire dal Thoughtland**, e quindi dal mondo delle idee astratte e delle opinioni, e **muoversi verso l'Actionland** dove si usano **artefatti** per collezionare dati e osservare le azioni degli utenti. Nel primo caso si usano le domande o questionari per poter ottenere informazioni sul prodotto che si andrà a sviluppare, col rischio di collezionare opinioni poco utili e magari fuorvianti, nel secondo caso mediante gli artefatti, si fanno svolgere azioni agli utenti al fine di raccogliere dati.

I prodotti del Thoughtland sono: **idee, domande e opinioni**. Quelli dell'Actionland sono **artefatti, azioni e dati**.

The Pretotyping Manifesto

innovators beat ideas

prototypes beat productypes

building beats talking

simplicity beats features

now beats later

commitment beats committees

data beats opinions

10.3 Pretotype vs Prototype

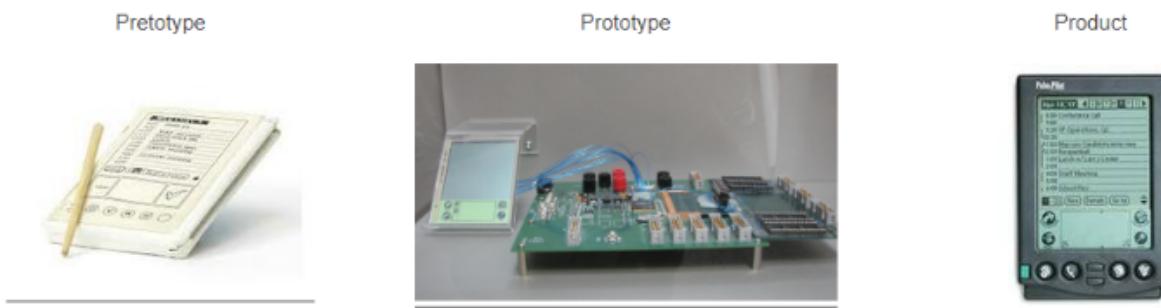
I prototipi aiutano a fallire in fretta, ma spesso non abbastanza in fretta o non abbastanza economicamente.

Più si investe in qualcosa e **più diventa difficile lasciarla morire** e ammettere che era la cosa sbagliata, anzi tendenzialmente, una volta ottenuto un buon prototipo che funziona, è facile portarlo avanti investendovi ancora denaro e tempo, pensando che l'aggiunta di funzionalità e features sia la risposta per renderlo vincente.

Molto spesso i prodotti lanciati sul mercato non sono altro che prototipi andati troppo oltre. **Tra il mondo delle idee astratte e i prototipi funzionanti**, si collocano i **pretotypes**.

Un pretotipo è un semplice mockup del prodotto che si vorrebbe sviluppare ed è utile per ottenere sia informazioni d'uso che di mercato e, inoltre, per poter prendere decisioni su cosa fare e su cosa non fare.

La **principale differenza** tra un pretotipo e un prototipo è l'**investimento**: un pretotipo costa molto meno sia in termini di tempo che in termini di denaro, e consente di fallire in fretta o nel caso, poiché lascia un ampio spazio di manovra, di apportare modifiche.



Il **pretotyping** ha l'obiettivo di aiutare a:

- **Identificare le funzionalità chiave** del nuovo prodotto.
- Decidere **quali** di queste possono o dovrebbero essere **inserite nel mockup**.
- Usare i mockup per **testare** sistematicamente e **collezionare feedback e dati d'uso**.
- Analizzare i dati raccolti per **determinare il prossimo passo da compiere**.

I sette pilastri del Pretotyping sono:

1. **Obbedire alla Legge del Fallimento di Mercato.**
2. **Assicurarsi di star costruendo il prodotto giusto.**
3. **Non perdersi in chiacchiere, idee o opinioni.**
4. **Fidarsi solo dei propri dati, TRUST YODA: Your Own DAta.**
5. **Fare pretotyping.**
6. **Parlare con i numeri e con i fatti.**
7. **Pensare globalmente e non localmente.**

10.4 Flusso del Pretotyping

I **cinque step** per produrre un buon pretotipo sono i seguenti:

1. **Isolare l'assunzione chiave:** quale è la assunzione o funzionalità chiave dell'idea che, se falsa, ne minerebbe la validità?
2. **Scegliere un tipo di prototype:** quale tipo di pretotyping permette di isolare e testare al meglio le funzionalità chiave?
3. **Fare ipotesi di mercato:** quante e quali tipi di persone utilizzeranno il pretotipo? Come lo utilizzeranno? Sarebbe possibile ipotizzare le percentuali di un determinato utilizzo?
4. **Testare il prototype:** mettere il pretotipo nel mondo reale e vedere quante persone sono interessate e quante ci interagiscono. Bisogna partire dal basso, un passo alla volta.
5. **Imparare, rifinire, hypozoom:** valutare i risultati, rifinire il pretotipo con i nuovi dati, e se l'ipotesi ha retto, decidere quali altre situazioni testare per ottenere una **visione completa o hypozooming**.

10.5 Tipi di Pretotyping

Andiamo ad analizzare alcuni tipi di pretotyping.

- Una **Fake Door** è un marketing entry point per un prodotto che ancora **non esiste** e può essere utilizzata per **pubblicizzare** un servizio non ancora pronto a misurare l'interesse degli utenti.

È un buon modo per capire se l'oggetto che si vuole sviluppare può avere successo o meno, spendendo pochissimo e quindi, in caso di fallimento, avere un basso impatto economico inteso sempre sia in termini di denaro che di tempo. Si può usare questo tipo di pretotyping **quando l'idea può essere descritta in poche e semplici parole**, senza possedere nulla di fisico o materiale.

What	A marketing entry point for an as-yet undeveloped idea.
Why	The solution doesn't exist yet and you want to capture an initial indication of interest at next to 0 cost.
When	Your idea can be concisely described and presented to potential customers where you would expect to find them, and you are confident you can manage the expectations of enthusiastic customers by following up within an appropriate time-frame.
How	Advertising a new product or feature, then tracking click-through and customer response rate to see who would be interested in an offering.
Where	Web tech enables a very robust method that includes: online ads + landing pages + simple response forms. Same approach also works with emails, offline posters and other media.

- Si parla di **Mechanical Turk** quando ci si riferisce a un oggetto che riesce a trasmettere l'esperienza del prodotto finale ad un utente, nel suo utilizzo, senza che esso sia stato sviluppato. Un meachanical turk usa solo man power. È molto utilizzato per sperimentare l'uso e la reale applicabilità di software e algoritmi molto costosi per essere implementati.

What	Use human power to simulate a technology that would take too much money or time to build
Why	To postpone costly development until market interest is validated.
When	When the final product requires the development of expensive and complex technology, and those actions and outputs could be simulated by humans.
How	Use a realistic interface to deliver target customers the essential experience of a proposed technology, simulating functionality of a complex back-end using human input.
Where	In the same real-life situation where the innovation will be used.

- Un **Impersonator** è un pretotipo che riesce a far vivere un'esperienza realistica all'utente in modo estremamente economico e con un lavoro minimo dietro. Consente cioè di far vivere l'esperienza esattamente come se il prodotto fosse finito e pronto per essere lanciato.

What	Use an existing product or service to pose as the new offer under test.
Why	Save on development costs while the market interest is not yet validated.
When	A test of the value of the solution depends on the customers' ability to interact with a full-scale design, and you need to create a plausible stand-in for the size, shape, color, features, etc. of the solution.
How	Apply a new skin to an existing product that can act as a good substitute to validate market interest.
Where	In the same real-life situation where the innovation will be used or accessed.

- Un **Pinocchio** è un prototipo **chiaramente falso**, serve per veicolare un messaggio così distante dalla realtà attuale che è faticoso e difficile da spiegare in altri linguaggi naturali. È usato molto spesso per testare l'interesse e il possibile uso di prodotti innovativi e non ancora lanciati da nessuno, nemmeno in maniera simile, sul mercato.

What An inanimate (or "dumb") artifact acts as a proxy for the real thing.

Why The solution doesn't exist yet and you want to validate a key design parameter early on.

When Your solution requires a significant switching or behavioral adaptation by customers to develop a new habit OR You expect demand to be sensitive to the appearance or form factor of your solution.

How Use a proxy to validate certain parameters of the product like form factor, features and usability.

Where In the same real-life situation where the innovation will be used.

- Con **One Night Stand** si indica una **tecnica di veicolazione** di un prototipo. Consiste di un market test. Viene usato insieme ad un'altra tecnica di prototyping per veicolare meglio il messaggio a una certa cerchia di persone.

What A complete service experience without the infrastructure required by a permanent solution.

Why Avoid investment in complex infrastructure and validate market interest and actual use.

When

- The solution is — or depends critically upon — an interactive service experience.
- You expect demand for the offer will be sensitive to the choice of channel, and you need to test a number of possible customer interception points.
- You want to validate a large homogeneous market before scaling up.

How Delivering target customers the essential experience within an extremely narrow geo scope and time frame.

Where In the same real-life situation where the innovation will be used but with limited time and geo scope.

- Un **Facade** è una sorta di Impersonator ma usato per dare un'immagine dell'azienda e non del prodotto stesso. Viene usato spesso per promuovere servizi.

What Borrow or rent expensive equipment, space, and assets to simulate a more stable or complex infrastructure underlying your offering.

Why Avoid investments in expensive equipment, space, and other assets while validating interest.

When

- The solution requires major upfront investment, in equipment, space, or assets.
- You expect demand will vary based on customer confidence in your infrastructure.

How Delivering target customers the essential experience, while communicating stability and complexity.

Where In the same real-life situation where the innovation will be used, but with all assets and space borrowed or rented cheaply.

10.6 Minimum Viable Product

Dopo varie fasi di pretotyping e aver accumulato sicurezze sufficienti circa il successo del prodotto, lo step successivo è produrre il **Minimum Viable Product**, cioè la **versione minimale del prodotto che contiene solo ed esclusivamente le features che si sono pretotipate attraverso la fase precedente**. Non si ha ancora per le mani un prodotto definitivo besì un qualcosa di **vendibile**, in modo da ottenere del ricavo e dell'utile, che se sufficiente, permetterebbe la produzione definitiva del prodotto.

MINIMUM VIABLE PRODUCT		Pretotype it book	An iPod Battery Pack	The Treadmill Bike
WHEN	The transition from prototyping to prototyping of the eventual product	Is there an audience for this content?	Is it handy to use one?	How is it like to run on a treadmill to propel a bike????
WHERE	Creating an artifact which delivers the core function(s) of the full solution			
WHY	you need to put the real product into customers' hands in order to permit a fair test			
WHAT	In the same real-life situation where the innovation will be used			
	You have learned all you can about market demand from simpler prototypes (Fake Door, Pinocchio, Mechanical Turk, One Night Stand, or Impersonator) and further insight requires a deeper customer interaction with a functioning artifact			

Figura 10.1: Minimum Viable Product.

Fruit of The Loom	Hackney Bike Racks	Large US Retailer	IDEO Surgical Dissector	Palm Pilot	Google Glasses
Pop-up store. Validate the ability to serve a different profile of customers with premium brand.	Validate interest and use by cyclists and actual influence on traffic.	Will customers trade in their white goods?	Optimising usability of a surgical tool	What if it fit in my shirt's pocket? Would I use it?	Should we superimpose a virtual image on all the visual field?

Figura 10.2: A sinistra esempi di one night stands e a destra esempi di pinocchio.

Tesla electric car and Lotus Elise	Upwell Studio and IKEA	Dog's Mineral Water
Validating market interest for a car that didn't exist yet.	Pretending the product is from IKEA and "selling" it in an IKEA store	Will customer buy mineral water for their pets?

Figura 10.3: Esempi di Impersonator.

Capitolo 11

UX for connected devices

Quando ci si trova di fronte al compito di progettare l'interfaccia di un **dispositivo connesso**, si tende ad adottare un approccio in linea alla *vecchia scuola* della UI e UX. Si tende quindi a dare maggior peso e a spendere maggior concentrazione su aspetti come l'estetica delle interfacce e la forma del prodotto fisico. **Per l'IoT ciò non basta.**

Un prodotto interconnesso può avere un'ottima UI ma una pessima UX, esso non è costituito solamente dalla sua apparenza o dalla sua interfaccia, ma è **l'insieme delle due cose**.

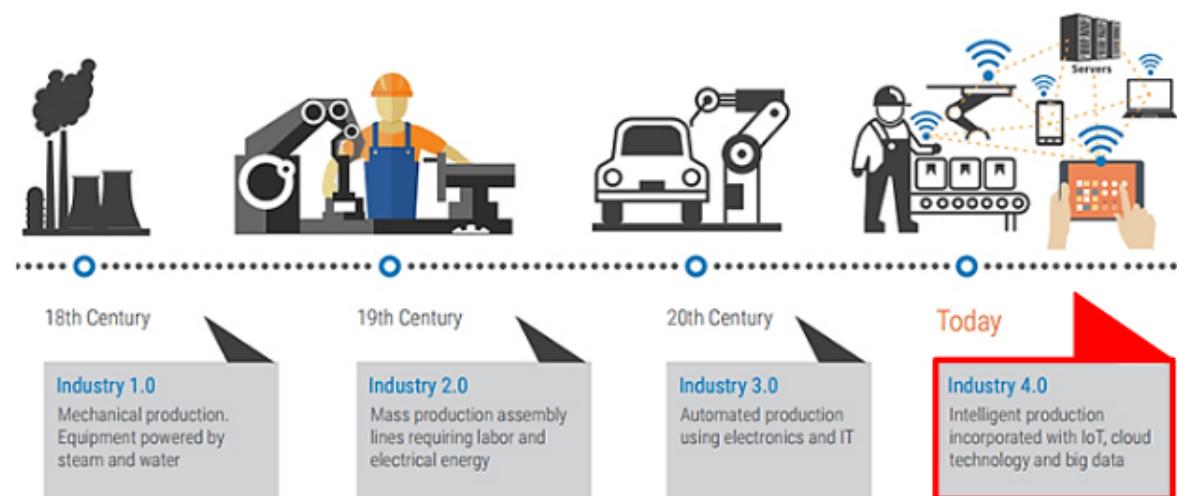
Internet è un **mezzo di comunicazione**, fatto dagli uomini per gli uomini, **l'Internet of Things o IoT è un sistema di dispositivi informatici correlati**, macchine meccaniche o digitali fornite di un **identificatore unico detto UID**.

Esse possiedono la capacità di trasferire i dati sulla rete senza richiedere l'interazione uomo-uomo o uomo-computer. La differenza tra i due mondi è sottile e spesso gli oggetti IoT sono creati per comunicare perfettamente tra di loro, ma hanno una pessima UI per l'uso umano, e di conseguenza sono portatori di una pessima UX.

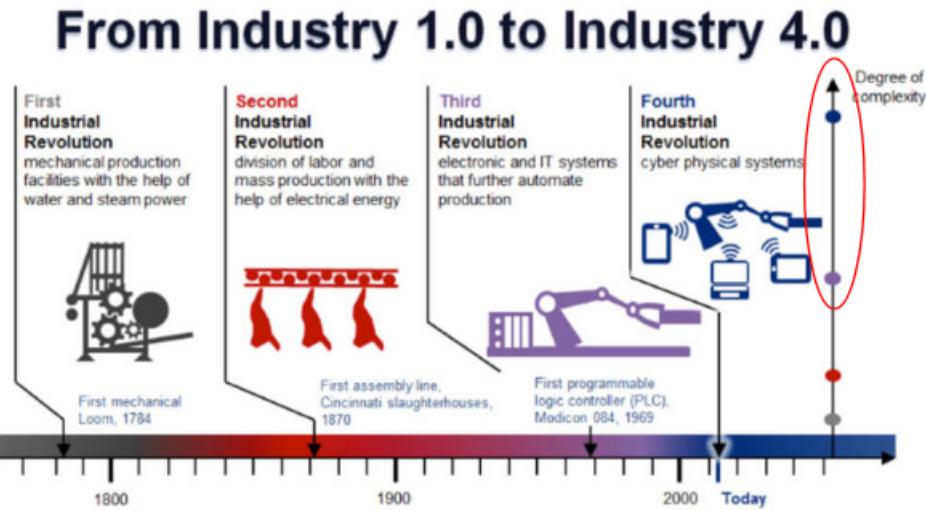
11.1 Industry 4.0

Il termine **Industria 4.0** indica l'ultima fase del processo con cui l'automazione industriale ad **integra** nuove **tecniche produttive** per **migliorare** le condizioni di lavoro, creare nuovi modelli di business e aumentare la produttività e la qualità produttiva degli impianti.

I sistemi computerizzati monitorano i processi fisici, creando copie virtuali del mondo su cui basare le scelte e le decisioni future per il business dell'azienda.



Il problema di ogni tappa della rivoluzione industriale risiede nell'aumento esponenziale della **complessità informatica**. I dati parlano chiaro: il salto informativo tra l'industria 3.0 all'industria 4.0 è pari al totale del salto effettuato da quando si usava l'asino come mezzo di locomozione fino alle attuali tecnologie.

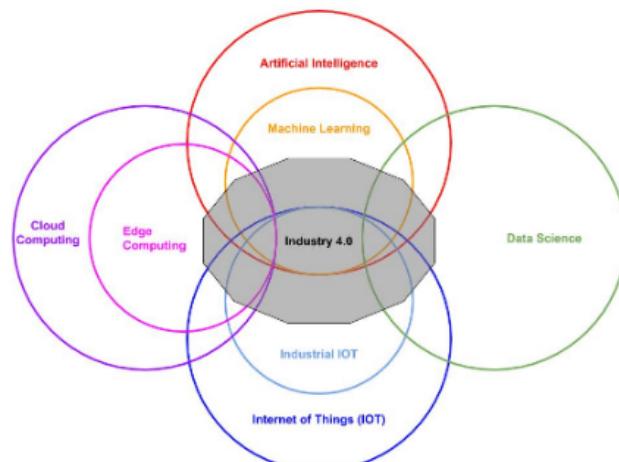


Quello che prima era un **processo produttivo**, con l'industria 4.0 diventa un **servizio** e la quantità di informazione necessaria cresce incredibilmente, di conseguenza progettare per la UX diventa estremamente complesso.

Ma come mai è importante questa evoluzione? Per queste ragioni:

- Ottimizzando la produzione i profitti aumentano.
- Sarà possibile creare **nuovi modelli di business**.
- Saranno accessibili **nuove tecnologie**.
- Sarà possibile trasformare la forza lavoro: da operai in tecnici, capaci di estrarre informazioni dalle macchine e modificare la produzione di conseguenza.

Il cuore di un sistema interconnesso, sia di tipo consumer che industriale, è il **Digital Twin**. Il Digital Twin è una **copia digitale di un bene, di una macchina o di un processo reale esistente**. La comunicazione tra l'oggetto fisico e il Digital Twin è **continua e bidirezionale**. Tale strumento può essere usato sia per logging o per controllo remoto, ma soprattutto è utile per simulazioni e manutenzione.



11.2 Products and Services in the 4.0 era

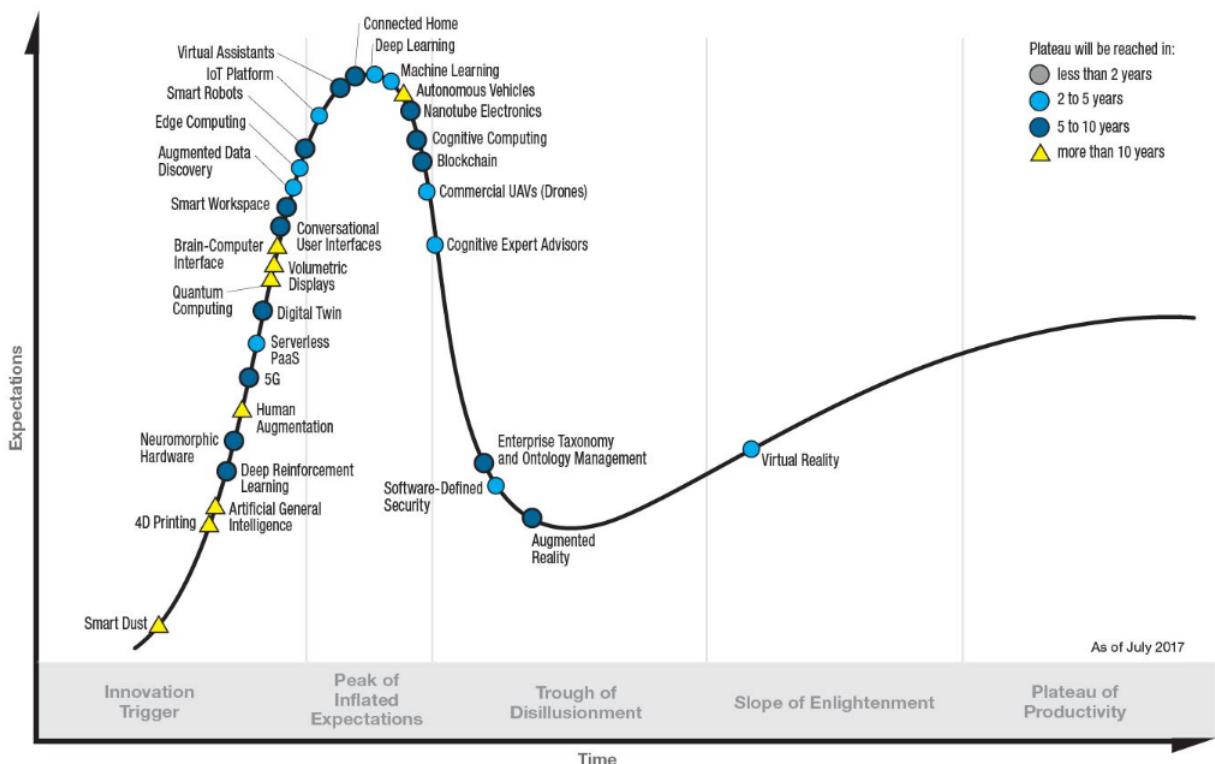


Figura 11.1: Gartner Hype Cycle for emerging technologies.

L'IoT è solo una tecnologia abilitante, quindi non bisogna progettare per l'IoT bensì bisogna progettare sistemi interconnessi. Ciò altera il paradigma di progettazione: prima erano i computer ad essere dei centri di aggregazione tecnologica, qualche anno fa, ad esempio, in una qualsiasi casa l'unico oggetto definibile smart era un computer, adesso si hanno più dispositivi connessi per svolgere svariate attività.

Sta sempre più emergendo il concetto di **intelligenza distribuita** accompagnata da un'**interfaccia personale**.

I dispositivi che si vogliono progettare dovrebbero avere funzionalità altamente **specifiche** ma veicolate attraverso interfacce **generiche**. Le interfacce specifiche sono da evitare.

Si prenda, per esempio, un robot da cucina che riesce a fare di tutto: frullare, centrifugare e cuocere, più una vasta gamma di funzionalità selezionabili, tramite pulsanti, uno per ogni funzione, attraverso la sua interfaccia che è posta sul robot stesso. **Ciò è da ripudiare** in favore di un robot senza un singolo tasto che capisce da sè cosa va fatto e come farlo.

La progettazione per l'IoT è intrinsecamente più **complessa** della progettazione di servizi Web. Design fisico, design della UX ed interconnettività di un unico sistema **non** possono essere gestiti separatamente.

I prodotti connessi pongono i progettisti dinanzi a sfide progettuali nuove, molte di queste derivano da:

- La **natura** specializzata dei servizi IoT.
- La capacità dei dispositivi IoT di fare da **ponte tra il mondo digitale e fisico**.
- Il fatto che molti prodotti IoT siano **sistemi distribuiti composti da più dispositivi**.
- **Le stranezze del networking**.

11.3 Real world context

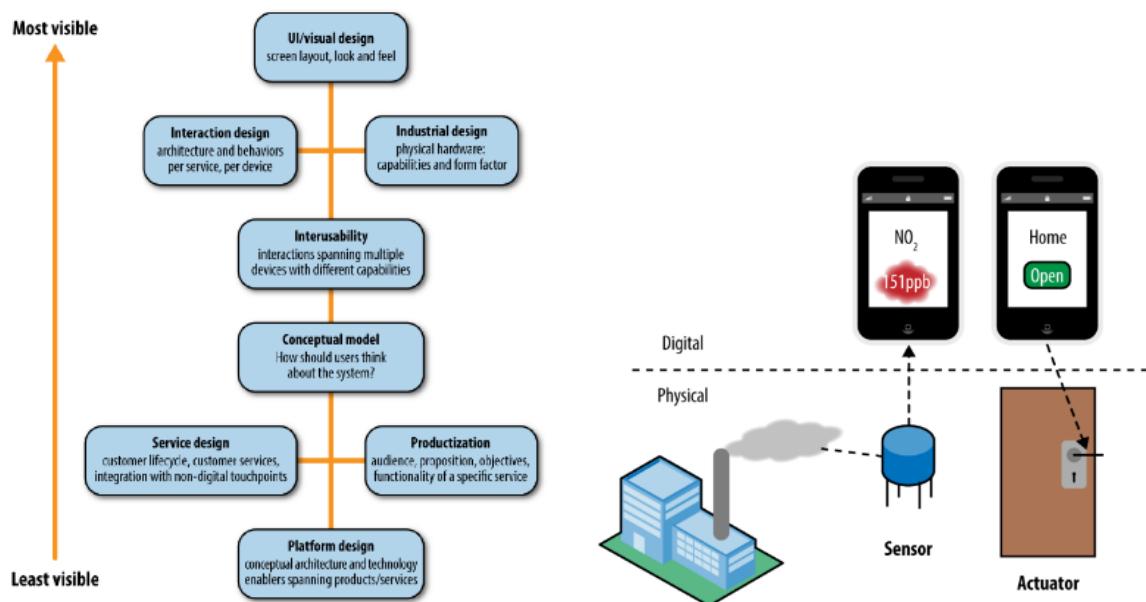
Le interfacce, nel mondo dell'IoT, sono sensori e attuatori. I **sensori** convertono una variabile **fisica** in un segnale **elettrico**, mentre gli attuatori convertono un segnale **elettrico** in una variabile **fisica**.

Gli attuatori possono essere controllati a distanza o automatizzati, ma a differenza dei comandi digitali, le azioni del mondo reale **non possono essere annullate**. Questo contesto fisico di utilizzo crea ulteriori sfide. Nell'IoT non si può dare per scontato lo stato d'animo dell'utente mentre interagisce con l'oggetto.

Gli oggetti ubiquitari, andando a posizionarsi nei vari angoli della casa o dell'ufficio, si trovano ad interagire con persone che potrebbero momentaneamente **non essere predisposte all'interazione con un oggetto digitale**, ciò rende l'errore molto più frequente.

Si devono creare quindi oggetti molto semplici sia da capire che da utilizzare.

Un altro aspetto importante è l'**interusabilità**, cioè la proprietà del sistema di essere usato attraverso **tutti** i dispositivi o le interfacce che lo compongono (*e.g. Google Assistant può essere usato tramite app, sito web o Google Home*).



Perché puntare così tanto su questa caratteristica?

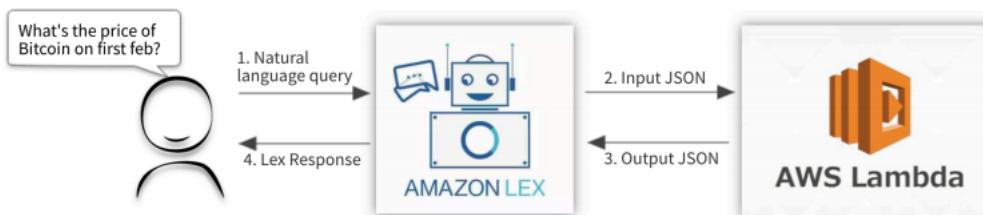
Bisogna infatti consentire all'utente l'accesso e l'interazione col sistema dove e come egli la desidera, il più compatibilmente possibile con le circostanze in cui egli si trova. L'**esperienza** dell'utente **non deve però differire** tra le varie interfacce. Inoltre, gli utenti hanno bisogno di una certa comprensione di come funziona il sistema. Anche i prodotti connessi piuttosto semplici sono concettualmente molto più complessi di quelli non connessi. Quindi è necessario un oggetto che veicola un modello **concettuale chiaro** in modo tale che l'utente si crei un modello mentale solido.

È necessario quindi creare un **modello unificato di interazione**, dove è chiaro che l'interfaccia e l'intelligenza sono distribuite, in modo tale che l'utente abbia un unico modello concettuale utilizzabile per l'intero sistema.

Altra problematica è che molti progettisti progettano supponendo che i dispositivi siano sempre connessi, ma questo **non è sempre vero!** Anzi conviene rovesciare il paradigma, **progettando per l'assenza di internet e assumendo che sia disponibile solo sporadicamente**. Con questo modo di progettare si considerano anche i ritardi e i problemi dovuti alle reti fisiche e alla trasmissione su di esse.

Capitolo 12

Natural Language



Un **Conversational System** è un'interfaccia uomo macchina in grado di comprendere **il linguaggio umano** e condurre una conversazione scritta o verbale con l'utente. È usata per migliorare l'esperienza dell'utente guidandone l'interazione durante l'uso del prodotto.

La conversazione deve essere:

- **Naturale**: l'utente deve poter usare un linguaggio **spontaneo**, non meccanico o verboso.
- **Accessibile**: l'utente deve capire facilmente e senza nessuna fatica come interagire con l'interfaccia.
- **Efficiente**.

Il primo problema che un programmatore deve affrontare quando vuole implementare un'interfaccia del genere è il seguente: **il linguaggio naturale è ambiguo**, infatti si possono usare molti costrutti per esprimere il medesimo concetto o impartire il medesimo comando.

Un servizio efficiente per poter scavalcare questo problema è **Amazon Lex**.

Amazon Lex si basa su due concetti fondamentali: **speech recognition** e **natural language understanding**. Benché esistano molti modi per esprimere il medesimo concetto, un servizio di riconoscimento vocale **deve essere in grado di capire** ciò che vogliamo fare e quale sia il nostro obiettivo.

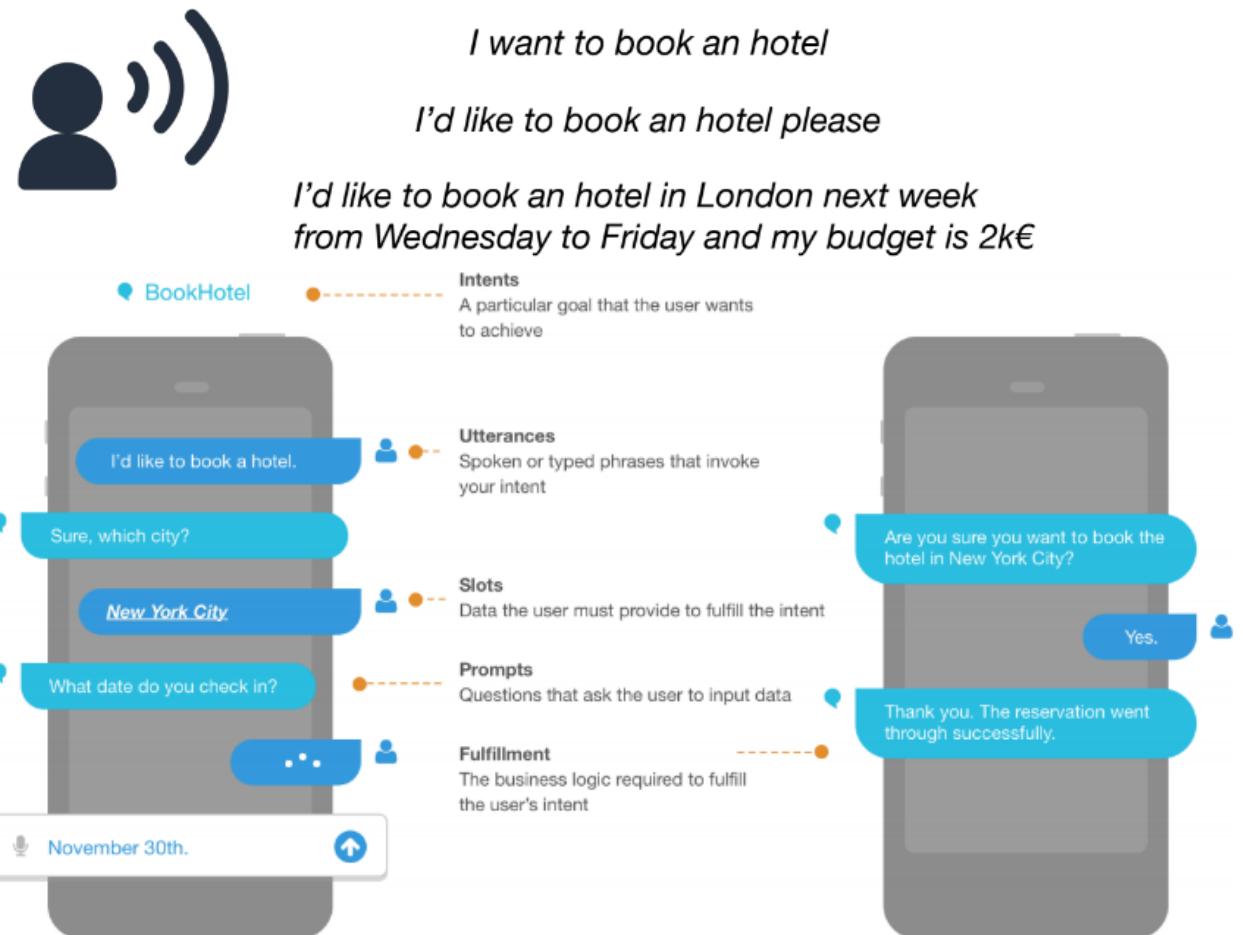
Si indicherà con il termine **intento** l'obiettivo dell'utente. Per poter soddisfare l'intento, si deve fare in modo che il sistema riconosca delle **informazioni chiave** per comprendere il comando. Queste informazioni chiave prendono il nome di **slots**.

Gli **slots** sono quindi i dati necessari al conversational system per soddisfare la richiesta dell'utente.

Inoltre per aiutare il sistema a interagire con gli utenti e a riconoscere il comando si definiscono anche delle **utterances**, ovvero delle frasi di esempio con diversa struttura ma uguale significato. Un insieme di **utterances** fa riferimento ad un **intento**.

Infine si procede a definire la sequenza d'interazione tra l'utente e il conversational system fornendo a quest ultimo dei **prompts**, ovvero la risposta che l'interfaccia darà all'utente dopo ogni comando ricevuto.

AWS Lambda è una piattaforma di calcolo **event-driven** e **serverless** che Amazon fornisce come parte dei suoi **servizi web**: mette in esecuzione del codice in risposta ad eventi e ne gestisce le risorse computazionali.



Lex Bot Structure - Utterances

- I'd like to book an hotel
- I want to make my hotel reservation
- Can you help me book an hotel?
- I want to book an hotel in London

Lex Bot Structure - Slots

I'd like to book an hotel in {London} {next week}
from {Wednesday} to {Friday} and my budget is {2k€}

Slot	Type	Values
Destination	City	New York, London, Barcelona
Check In	Date	Valid Dates
Check Out	Date	Valid Dates