Usabilità del software

L'essenza del problema...

L'altalena...

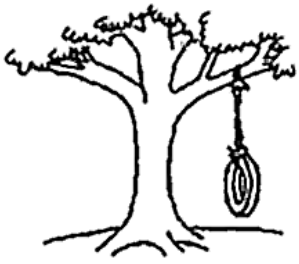
Come è stata richiesta dal committente

Come è stata descritta nelle specifiche

Come è stata progettata dall'analista senior

Come è stata realizzata dai programmatori

Come è stata installata presso l'utente



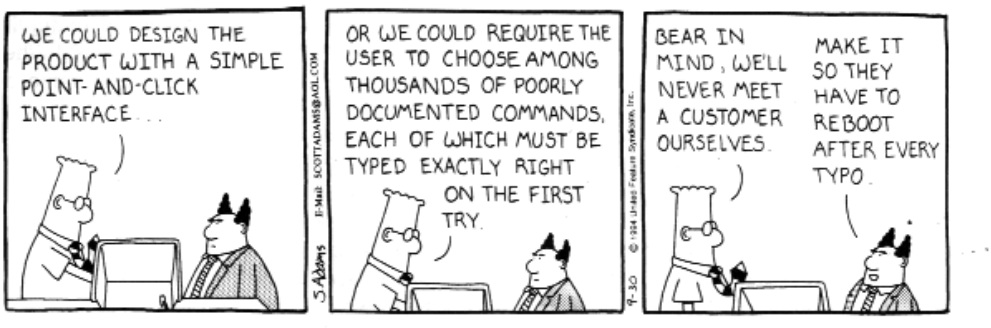
Come la voleva realmente l'utente

Definizioni di usabilità

- L'usabilità è la facilità con cui un utente impara ad operare con un sistema o un componente, a fornirgli gli input e ad interpretarne gli output [IEEE 90]
- **L'usabilità di un'interfaccia è la misura dell'efficacia, efficienza e soddisfazione con cui determinati utenti possono compiere determinati compiti in un determinato contesto utilizzando tale interfaccia** [ISO 13407]

Cos'è l'usabilità?

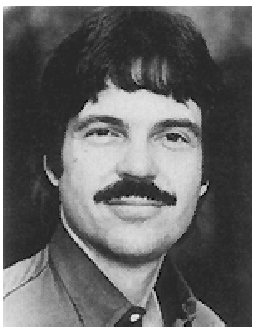
- Bisogna mettere al centro di tutto l'utente (**user centered design**)
- Gli utenti usano le applicazioni per essere produttivi
- Gli utenti sono persone impegnate che cercano di eseguire dei compiti
- **Sono gli utenti che decidono quando un prodotto è facile da usare**



Usabilità nel processo

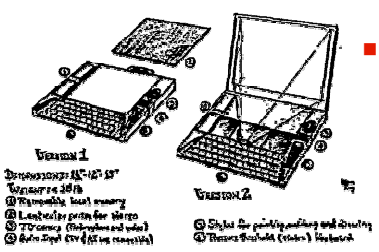
- L'usabilità deve essere presa in considerazione nelle prime fasi del processo di progettazione e sviluppo
- Bisogna coinvolgere gli utenti nel processo di progettazione fino alle fasi di test
- Bisogna far sì che l'usabilità e le esigenze degli utenti guidino il processo di progettazione
- Eseguire test frequenti durante il processo di progettazione

Santino n. 1: Alan Kay



- **Chi è?**
  - Direttore negli anni 70-80 del mitico Xerox PARC a Palo Alto
- **Che cos'ha fatto?**
  - Ha inventato la **programmazione ad oggetti**, le **interfacce grafiche a finestre**, i **computer portatili**, la **rete ethernet**...
  - E' stato uno dei primi a porre il problema della progettazione delle interfacce utente
- **Che cosa ha detto?**
  - "Il miglior modo di predire il futuro è quello di inventarlo"
  - "Per un utente l'interfaccia è l'applicazione"

Early renderings of Dynabook.



A chi interessa l'usabilità

- Un'organizzazione che produce prodotti usabili beneficia di:
  - Maggior soddisfazione degli utenti
  - Minori costi di supporto
  - Minori costi di formazione
  - Minori necessità di aggiornamenti e di release di manutenzione
  - Documentazione più semplice da realizzare
  - Utenti più produttivi con meno stress.

Usabilità e progettazione

- L'usabilità è un problema che riguarda la progettazione del software e quindi rientra in qualche modo nell'ambito dell'ingegneria del software
- Rappresenta uno degli aspetti principali di questa disciplina, assieme alla progettazione della base dati, dell'architettura applicativa ecc.
- E' in realtà l'aspetto più importante perché è quello con l'impatto più elevato nei confronti del cliente: **buona parte del giudizio del cliente è determinato dalla qualità dell'interfaccia utente**

Progettare l'usabilità

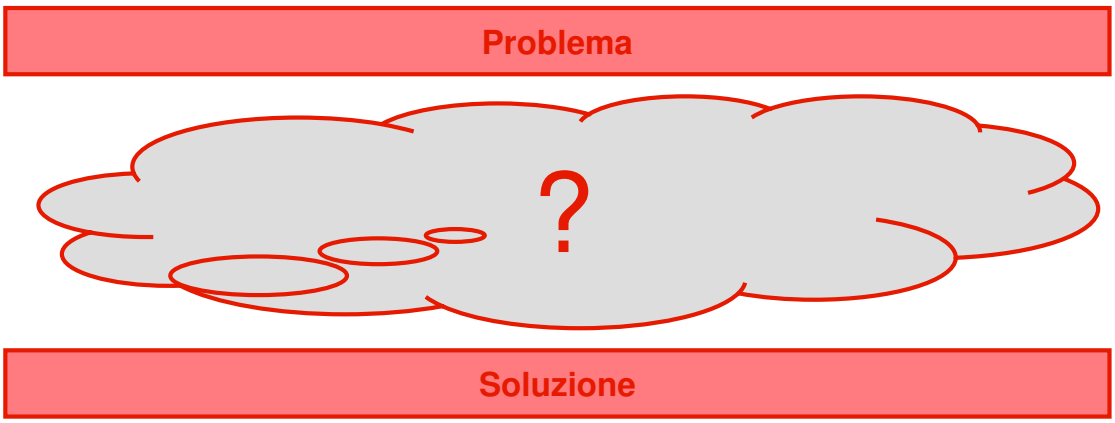
- Per realizzare un sistema usabile il progettista deve riuscire a fare correttamente due cose:
  - Creare un buon modello concettuale del sistema (corretto, coerente, semplice)
  - Riuscire a trasmetterlo efficacemente all'utilizzatore attraverso l'interfaccia utente
- Il **modello concettuale** è un'immagine mentale che le persone si costruiscono riguardo al funzionamento di un oggetto o di un sistema

I tre modelli

- Esistono di solito tre modelli:
  - Quello nella testa del progettista
  - Quello esposto dal sistema (immagine del sistema)
  - Quello nella testa dell'utente
- In una situazione ideale i tre modelli dovrebbero coincidere
- **Il problema dell'usabilità si pone quando il modello del progettista non coincide con il modello dell'utente**
- Abbiamo due possibili cause di non usabilità:
  - Il modello concettuale del progettista non è chiaro e coerente
  - Il sistema non rappresenta in modo corretto ed efficace il modello concettuale del progettista

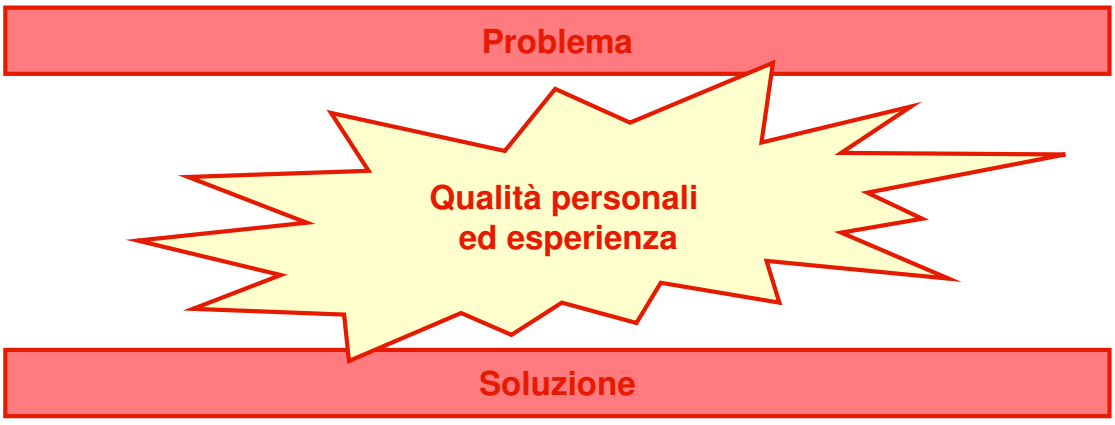
Come si crea un (bravo) progettista?

- Come in tutti i problemi di progettazione – in qualunque campo – l'abilità e l'esperienza del progettista rappresentano un fattore fondamentale
- Il problema a questo punto è: **come si forma un bravo progettista di interfacce utente?**
- Come si può colmare il gap tra problema e soluzione?



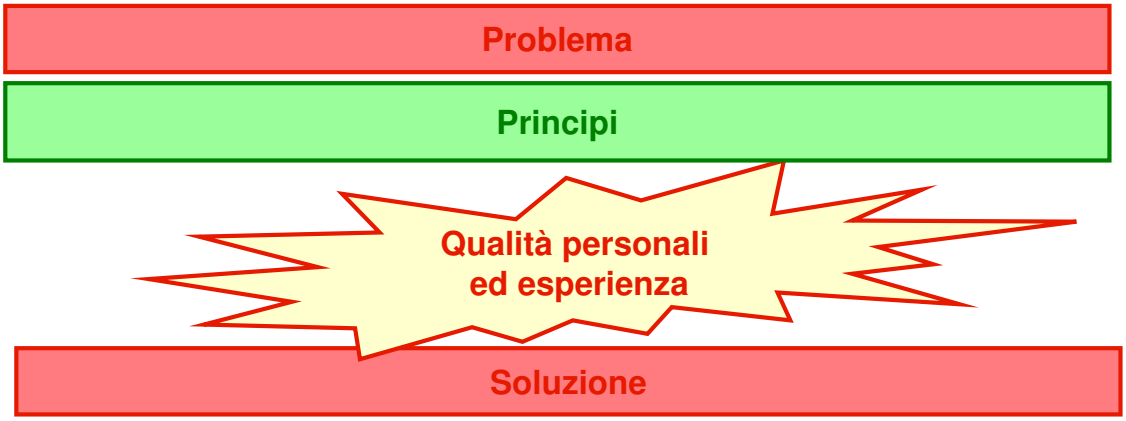
Approccio n. 1: Selezione naturale

- Si lancia il progettista nell'arena: quelli che sopravvivono agli utenti (notoriamente feroci) diventano (forse) dei buoni progettisti di interfacce utente
- Probabilmente non è il metodo più efficace…



Approccio n. 2: I principi

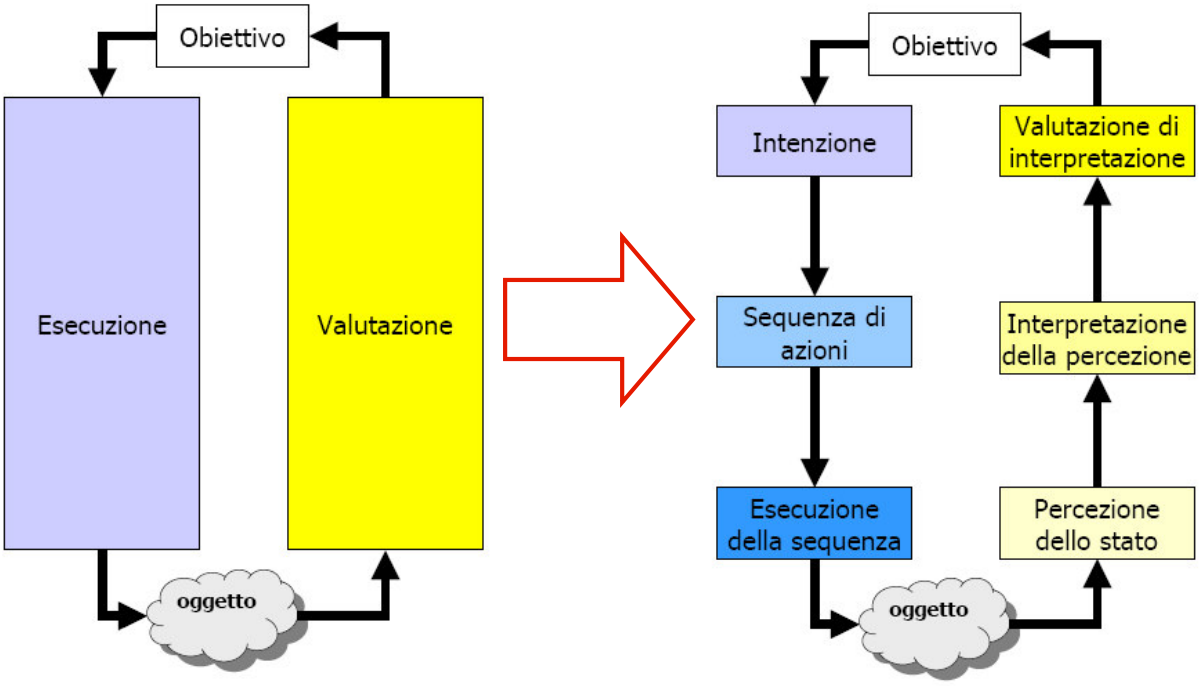
- Si cerca di ridurre il gap dall'alto: si definiscono principi e linee guida che guidano il progettista nella ricerca della soluzione
- Il progettista deve assimilare i principi e poi utilizzare la propria abilità ed esperienza per applicarli correttamente



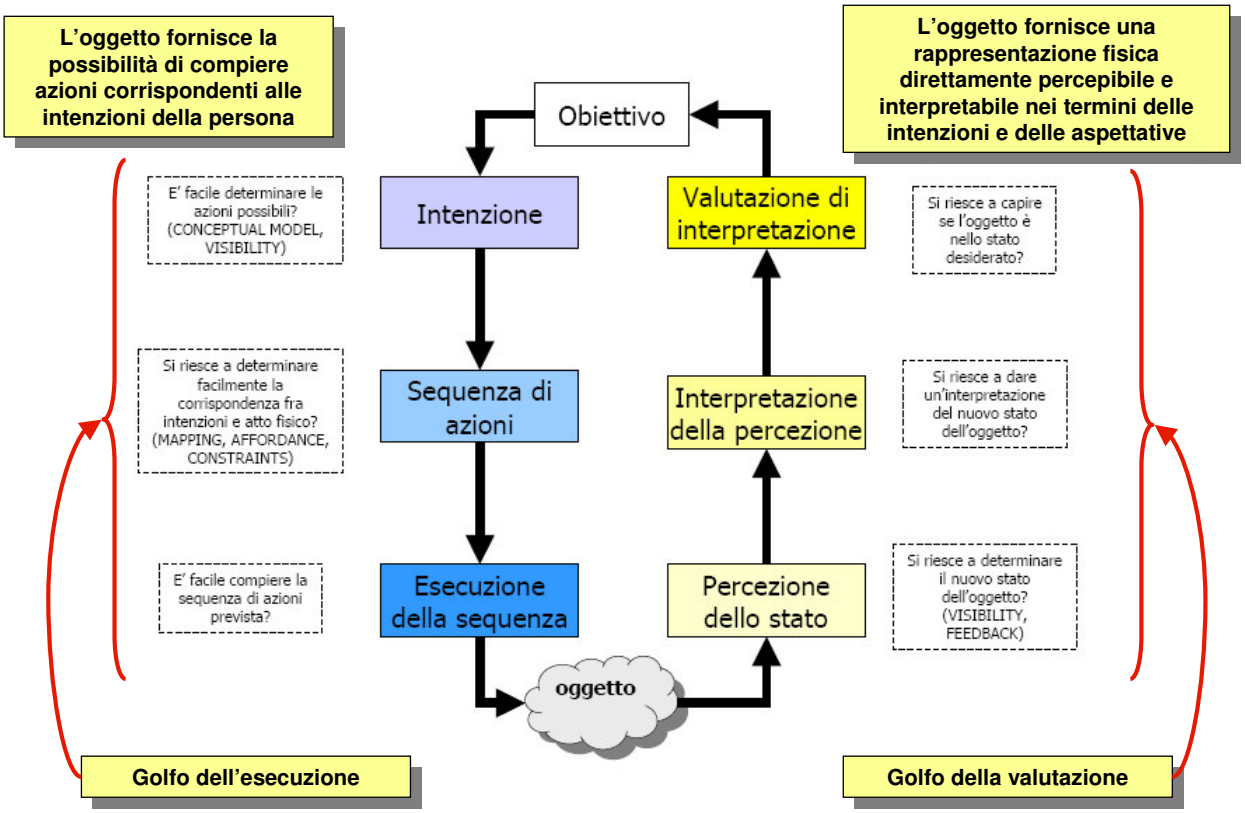
I principi di usabilità

- Fonti dei principi di usabilità:
  - **Studi di psicologia cognitiva**
  - **Progettazione grafica**
  - **Conoscenza ed esperienza dei progettisti**
- Possono essere di diversi tipi e livelli di formalità:
  - **Principi di alto livello:** raccomandazioni basate sulla conoscenza del comportamento umano
  - **Standard di progetto:** norme definite formalmente da appositi organismi (de jure) o imposte dal mercato (de facto).
  - **Linee guida:** raccomandazioni che derivano da convenzioni comunemente adottate fra gli "addetti ai lavori"

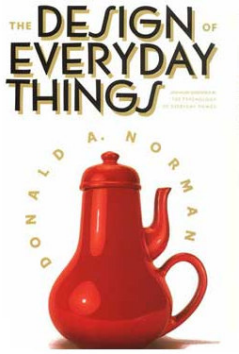
Psicologia cognitiva: i sette stadi dell'azione



Azioni e usabilità



Santino n. 2: Donald Norman



- **Chi è?**
  - Informatico e psicologo cognitivo.
  - Uno dei padri dell'usabilità, non solo nel campo del software
- **Cosa ha fatto?**
  - Autore di molti libri tra cui il libro cult: **The Psychology of Everyday Things** (POET) . (In italiano "La caffettiera del masochista")
- **Che cosa ha detto?**
  - "Se siete progettisti contribuite alla battaglia per l'usabilità degli oggetti quotidiani"
  - "Avete dimenticato l'aspetto più importante della progettazione: capire i vostri utenti"

I sette principi di Norman

- Sette principi per trasformare compiti difficili in compiti facili:
  1. Usare sia la conoscenza nel mondo che quella nella testa.
  2. Semplificare la struttura dei compiti (task)
  3. Rendere visibili le cose
  4. Impostare bene i mappings (correlazioni)
  5. Sfruttare i vincoli, sia naturali che artificiali.
  6. Lasciare un margine di errore.
  7. Quando tutto il resto non serve aderire agli standard.

1. Conoscenza nel mondo e conoscenza nella testa

- Le persone si trovano più a loro agio quando la conoscenza richiesta per eseguire un compito è accessibile nel mondo esterno, sia espressa esplicitamente che sotto forma di vincoli naturali
- La conoscenza esterna è però utile solo se c'è un rapporto facilmente interpretabile fra questa conoscenza e le azioni e i risultati possibili
- D'altro lato se un utente riesce a interiorizzare la conoscenza esterna l'esecuzione di un compito è di solito più rapida ed efficiente
- **Il design deve aiutare l'utente inesperto (che cerca la conoscenza nel mondo) senza ostacolare o rallentare quello esperto (che ha la conoscenza in testa).**

2. Semplificare la struttura dei compiti

- La tecnologia dovrebbe servire soprattutto a semplificare: bisogna rendere più semplici possibili i compiti da eseguire.
- Quattro approcci possibili:
  - **Fornire strumenti di supporto** (un calendario, una calcolatrice, un blocco appunti)
  - **Rendere visibile l'invisibile** (elaborazioni grafiche per "vedere" i numeri e evidenziare a colpo d'occhi le informazioni importanti)
  - **Automatizzare** (pilota automatico, cambio automatico). Attenzione però a non eliminare i comandi manuali!
  - **Cambiare la natura del compito** (stringhe e scarpe con il velcro). Pensiero laterale

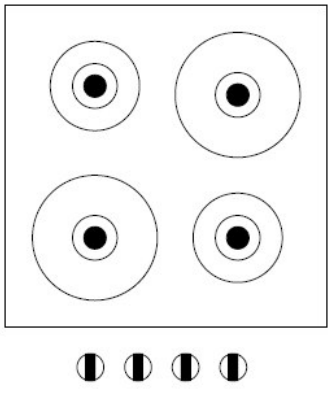
3. Rendere visibili le cose

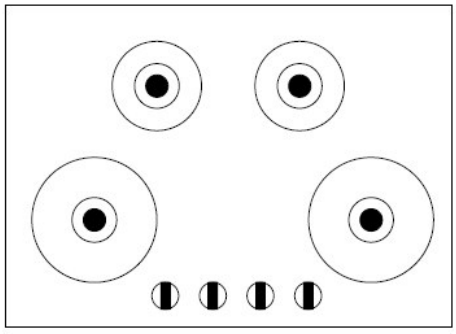
- **Golfo dell'esecuzione**: distanza fra gli obiettivi dell'utente e il modo di ottenerli mediante il sistema
- **Golfo della valutazione**: quantità di sforzo necessario per interpretare lo stato fisico del sistema
- **Un design usabile punta a ridurre i due golfi**
- Rendere visibili le cose aiuta a cogliere questo obiettivo
  - Rendere visibili le cose sul versante esecutivo, aiutando l'utente a capire come cosa può fare e come può farlo, permette di ridurre il golfo dell'esecuzione
  - Rendere visibili e facilmente interpretabili lo stato del sistema e il risultato di un'azione aiuta a ridurre il golfo della valutazione

4. Impostare bene i mappings (correlazioni)

- Un concetto centrale è la **compatibilità di risposta:**

  Il rapporto spaziale fra la posizione dei comandi e gli oggetti su cui agiscono deve essere il più diretto possibile
- La stessa cosa deve valere per il feedback che permette di valutare gli effetti di un'azione





5. Sfruttare i vincoli, sia naturali che artificiali

- Bisogna usare i vincoli per fare in modo che l'utente abbia l'impressione che ci sia una sola cosa possibile da fare (quella giusta si spera)
- I vincoli possono essere:
  - **Fisici** (importanti per chi progetta oggetti fisici)
  - **Logici** (utilizzano il ragionamento per determinare le alternative possibili: se non si vede tutta la pagina si sa che si può scrollare)
  - **Culturali** (convenzioni). Per es. manina sui link. Sono fondamentali nei progetti software.
- In un rubinetto:
  - L'impossibilità di tirare o spingere e la possibilità di ruotare sono vincoli fisici
  - L'uso del blu e del rosso per indicare l'acqua fredda o calda è un vincolo culturale

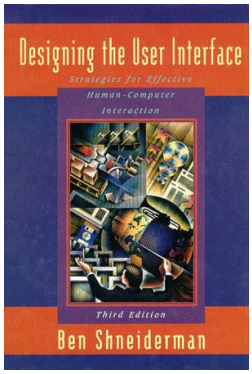
6. Lasciare un margine di errore

- Progettare rispettando la legge di Murphy:
  - **Ogni errore che può essere commesso prima o poi sarà commesso**
- Bisogna interpretare ogni azione dell'utente come un tentativo di andare nella giusta direzione e quindi incoraggiarlo, non punirlo:
  - Consentire di rimediare agli errori (undo)
  - Indicare le possibili alternative corrette
  - Rendere difficili le operazioni irreversibili
  - Progettare sistemi esplorabili in cui andare a ficcare il naso nelle cose non porta ad azioni dannose
  - Sfruttare le funzioni obbliganti: la mancata esecuzione di un passaggio impedisce di eseguire il successivo (p.es cambio automatico e chiave)

7. In mancanza di meglio aderire agli standard

- Se non esiste modo di eliminare del tutto l'arbitrarietà la cosa migliore da fare è aderire agli standard
- Esempio di funzionamenti arbitrari che risultano accettabili grazie agli standard:
  - Direzione di rotazione degli orologi (non c'è alcun motivo pratico per farli girare da sinistra a destra)
  - Colori dei semafori (ricordarsene sempre quando si usano segnali di allarme o corretto funzionamento)
- Non sempre funziona purtroppo: si veda l'esempio dei bottoni Ok e Annulla in Windows e su Mac

Santino n. 3: Ben Shneiderman



- **Chi è?**
  - Informatico
  - Fondatore dello **Human-Computer Interaction Laboratory** presso l'Università del Maryland
- **Cosa ha fatto?**
  - Autore di molti libri tra cui **Designing the User Interface: Strategies for Effective Human-Computer Interaction**
- **Che cosa ha detto?**
  - "Un'immagine vale mille parole, un interfaccia vale mille immagini"
  - "Leonardo Da Vinci combinava arte e scienza, estetica e ingegneria, il tipo di unità che serve nuovamente oggi."

Le otto regole d'oro di Shneiderman

- Richiamano in parte i principi di Norman ma con un taglio un po' più pratico :
  1. Preservare la coerenza
  2. Consentire agli utenti abituali di usare comandi rapidi (shortcuts)
  3. Offrire un feedback informativo
  4. Progettare dialoghi provvisti di chiusura
  5. Offrire una **prevenzione** e una gestione semplice degli errori
  6. Permettere un'inversione agevole delle azioni
  7. Favorire un **locus of control** interno: dare agli utenti la sensazione di avere le cose sotto controllo
  8. Ridurre il carico della memoria a breve termine

4. Progettare dialoghi provvisti di chiusura

- I dialoghi dovrebbero avere :
  - Un inizio "File->Apri"
  - Una parte intermedia: completamento del dialogo
  - Una fine pressione del tasto "Open"
- Il feedback informativo al completamento di un gruppo di azioni dà all'operatore:
  - La soddisfazione del compimento
  - Un senso di sollievo
  - Il segnale che può rimuovere dalla mente le opzioni e i piani a breve termine (scarico della memoria a breve termine)
  - Un'indicazione che può prepararsi per il prossimo gruppo di azioni.

5. Prevenzione e gestione facile degli errori

- **Prevenzione degli errori:**
  - Progettare il sistema in modo che gli utenti non possano compiere errori gravi
  - Usare selezioni in un menu anziché riempimento di campi.
  - Quando si usano campi da compilare non consentire, ad esempio, di inserire caratteri alfabetici in campi numerici
  - Non lasciare attivi comandi relativi ad azioni che non possono essere compiute
- **Gestione degli errori:**
  - Cos' è successo?
  - Perché è successo?
  - Quanto è grave?
  - Come si può rimediare?

7.Locus of control interno

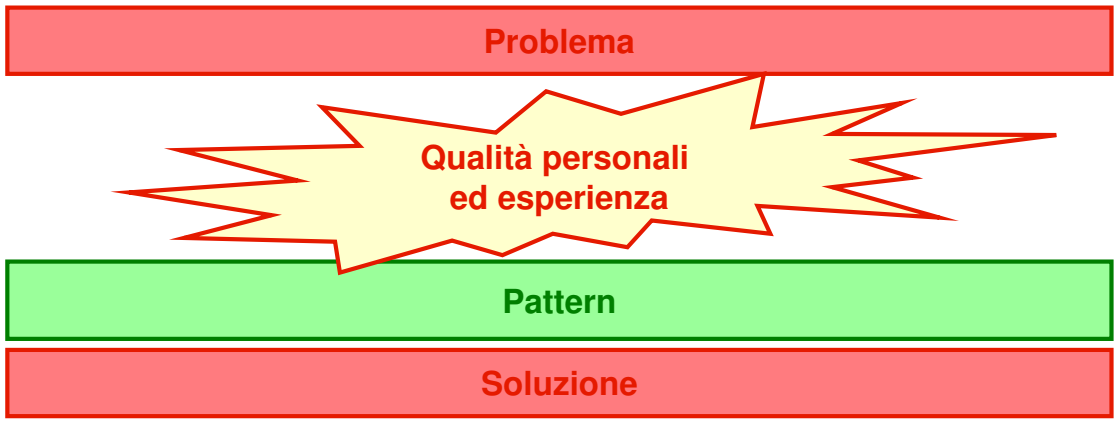
- Il **locus of control** è una variabile psicologica che indica il grado di percezione rispetto al controllo del proprio destino e gli eventi.
  - Un locus of control "**esterno**" porta il soggetto ad attribuire prevalentemente al destino o agli "altri" il controllo di quanto accade.
  - Un locus of control "**interno**" vede invece il soggetto molto più indirizzato a considerare il destino come un effetto delle proprie azioni.
- Un software dovrebbe dare all'utente la sensazione che quello che avviene dipende dalle proprie azioni e non da qualche entità misteriosa che agisce dietro le quinte.

8.Ridurre il carico della memoria a breve termine

- Gli esseri umani possono tenere in mente 7 (+/- 2) concetti (**capacità di canale**)
- Nuovi stimoli "spingono fuori" quelli più vecchi
- Quindi, per esempio:
  - Mantenere semplici le schermate
  - Fornire un accesso in linea a regole sintattiche, abbreviazioni, codici…
  - Non creare situazioni in cui gli utenti devono segnarsi su un foglietto un codice per reinserirlo successivamente: se è già stato inserito il computer dovrebbe conoscerlo

Approccio n. 3: I pattern

- Per un progettista inesperto non è facile interpretare e attuare i principi
- Si cerca di ridurre il gap dal basso: si definiscono dei pattern, soluzioni progettuali riusabili che garantiscono a priori una buona usabilità



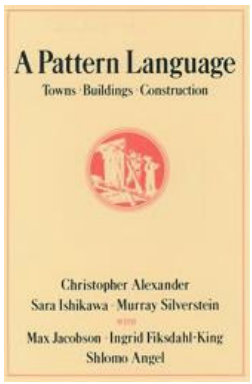
Cosa sono i pattern?

- L'esperienza è un fattore fondamentale per una buona progettazione: questo avviene perché un progettista esperto non parte mai da zero, riutilizza soluzioni che si sono dimostrate valide in passato.
- Esiste quindi, oltre al riuso nella scrittura del software, anche un **riuso nella progettazione**.
- **Un pattern è la soluzione ad un problema ricorrente**: dal momento in cui il problema viene evidenziato si ha a disposizione tutta una serie di criteri automatici per risolverlo
- La "comunità dei pattern" si propone come obiettivo la catalogazione di questi schemi ricorrenti in modo da costituire un dizionario a disposizione dei progettisti.
- **Le soluzioni proposte dai pattern rispettano i principi generali di buona progettazione**

L'origine

- L'idea è stata presa in prestito dal campo dell'architettura e in particolare dal lavoro di Christopher Alexander: **A pattern language: towns, buildings, construction** (1977)
- Alexander riassume così la sua visione:
- **"Ogni pattern descrive un problema che si presenta ricorrentemente nel nostro ambiente, e quindi descrive il nucleo della soluzione in modo tale da poter utilizzare questa soluzione milioni di volte senza dover rifare due volte le stesse cose"**

Santino n. 4: Christopher Alexander



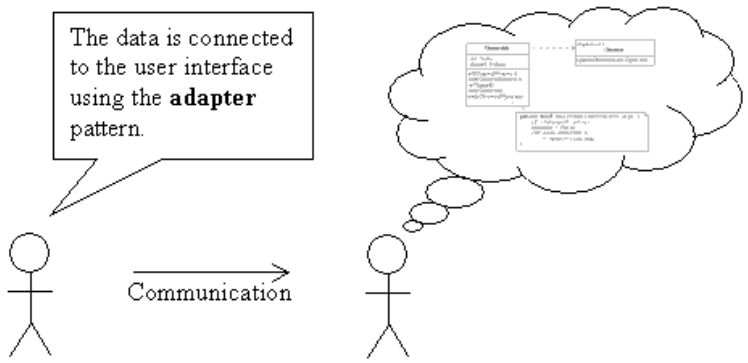
- **Chi è?**
  - Architetto, nato a Vienna, cresciuto in Inghilterra
  - Padre del **movimento dei pattern** nel campo dell'informatica
- **Cosa ha fatto?**
  - Autore di molti libri tra cui **A pattern language: towns, buildings, construction (1977)**
- **Che cosa ha detto?**
  - C'è solo un modo senza tempo per costruire. E' vecchio di migliaia di anni ma è sempre lo stesso. … Non è possibile costruire luoghi dove ci si sente se stessi, dove ci si sente vivi se non seguendolo…

The Gang of Four e HCI Patterns

- L'idea è stata ripresa e adattata alla progettazione del software da Gamma, Helm, Johnson e Vlissides
- E' stata esposta nel libro **"Design Patterns: Elements of Reusable Object-Oriented Software"** (1994)
- I 4 autori sono noti anche come **Gang of Four** (GoF)
- Il libro descrive 23 pattern che vengono chiamati **GoF Patterns**
- In seguito ne sono stati individuati e descritti molti altri
- L'idea è stata ripresa anche nell'ambito dell'usabilità con gli **HCI (Human Computer Interaction) Patterns**
- In fondo anche il lavoro di Alexander poneva un forte accento sull'usabilità

Il vocabolario dei pattern

- Oltre a rendere disponibili tecniche di progettazione sperimentate, il movimento dei pattern ha il merito di aver creato un **"vocabolario"** di progettazione molto utile per le comunicazioni all'interno di un team
- Quando un progettista dice **"questo è un adapter"** trasmette ai suoi interlocutori in forma concisa e priva di ambiguità una grossa quantità di informazioni utili



Formato descrittivo dei pattern

- I pattern sono organizzati come schede descrittive e ogni autore tende ad utilizzare un formato uniforme
- Il formato utilizzato originariamente da Alexander viene chiamato **Alexandrian Form**
- Il formato utilizzato nel testo di Gamma, Helm, Johnson e Vlissides prende il nome di **GoF Form**

Schema generale

- In generale i vari formati prevede 4 elementi base:
  - **Nome:** fornisce un'identificazione precisa e concisa
  - **Problema:** descrive quando applicare il pattern
  - **Soluzione:** descrive gli elementi di progetto da utilizzare e le relazioni fra di esse
  - **Conseguenze:** descrivono i vantaggi e gli eventuali svantaggi dell'applicazione del pattern al problema. Permettono di valutare le alternative progettuali.
- A questo schema di base possono corrispondere schemi più dettagliati (per esempio quello GoF)

Alexandrian Form

- **Nome** del pattern
- Valutazione della sua **validità (ranking)**
- Una figura come **esempio** di applicazione
- Il **contesto** in cui deve essere usato
- Il **problema**
  - Una breve esposizione
  - Una descrizione dettagliata
  - Le **forze** in competizione
- Le **soluzioni** che costituiscono i cuore del pattern
- Un **diagramma** che illustra la soluzione
- **Riferimenti** ad altri pattern

Alexandrian Form: esempio

Nome

ranking

contesto

243 SITTING WALL**

figura

. . if all is well, the outdoor areas are largely made up of positive spaces—POSITIVE OUTDOOR SPACES (106); in some fashion you have marked boundaries between gardens and streets, between terraces and gardens, between outdoor rooms and terraces, between play areas and gardens—GREEN STREETS (51), PEDESTRIAN STREET (100), HALF-HIDDEN GARDEN (111), HIERARCHY OF OPEN SPACE (114), PATH SHAPE (121), ACTIVITY POCKETS (124), PRIVATE TERRACE ON THE STREET (140), OUTDOOR ROOM (163), OPENING TO THE STREET (165), GALLERY SURROUND (166), GARDEN GROWING WILD (172). With this pattern, you can help these natural boundaries take on their proper character, by building walls, just low enough to sit on, and high enough to mark the boundaries.

If you have also marked the places where it makes sense to build seats—SEAT SPOTS (241), FRONT DOOR BENCH (242)—you can kill two birds with one stone by using the walls as seats which help enclose the outdoor space wherever its positive character is weakest.

❖ ❖ ❖

problema

In many places walls and fences between outdoor spaces are too high; but no boundary at all does injustice to the subtlety of the divisions between the spaces.

forze

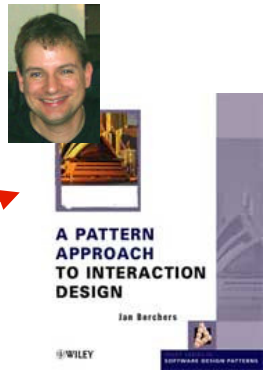
Consider, for example, a garden on a quiet street. At least somewhere along the edge between the two there is a need for a seam, a place which unites the two, but does so without breaking down the fact that they are separate places. If there is a high wall or a hedge, then the people in the garden have no way of being connected to the street; the people in the street have no way of being connected to the garden. But if there is no barrier at all—then the division between the two is hard to maintain. Stray dogs can wander in and out at will; it is even uncomfortable to sit in the garden, because it is essentially like sitting in the street.

GoF Form

- **Nome e classificazione:** importante per il vocabolario
- **Scopo:** cosa fa il pattern e suo fondamento logico
- **Nomi alternativi:** molti pattern sono conosciuti con più nomi
- **Motivazione:** scenario che illustra un problema di progettazione e la soluzione offerta
- **Applicabilità:** quando può essere applicato il pattern
- **Struttura (o modello):** rappresentazione UML del pattern
- **Partecipanti:** le classi/oggetti con le proprie responsabilità
- **Collaborazioni:** come collaborano i partecipanti
- **Conseguenze**: pro e contro dell'applicazione del pattern
- **Implementazione:** come si può implementare il pattern
- **Codice d'esempio:** frammenti di codice
- **Pattern correlati:** relazioni con altri pattern
- **Utilizzi noti:** esempi di utilizzo reale del pattern in sistemi esistenti

HCI Patterns

- Anche nell'ambito dei pattern che si occupano dell'interazione fra uomo e computer (HCI) e quindi dell'usabilità ci sono vari formati (legati ai diversi autori):
  - **Borcher**'s form (utilizzato nel libro "A pattern approach to interaction design")
  - Tidwell's **Common Ground** form
  - **Van Weelie** form
  - …
- Spesso si tratta di leggere variazioni sul tema



Un esempio in formato Common Ground

- **Nome:** Go back to a safe place
- **Esempi**:
  - Il bottone "Home" in un Web browser
  - Tornare all'inizio di un capitolo in un libro
  - La funzione "undo" in alcune applicazioni
- **Contesto:** L'applicazione permette all'utente di muoversi attraverso spazi o stati diversi e possiede uo o più "checkpoint" in questo insieme di spazi.
- **Problema:** come si può rendere la navigazione facile, conveniente e psicologicamente sicura per l'utente?

Go back to a safe place…

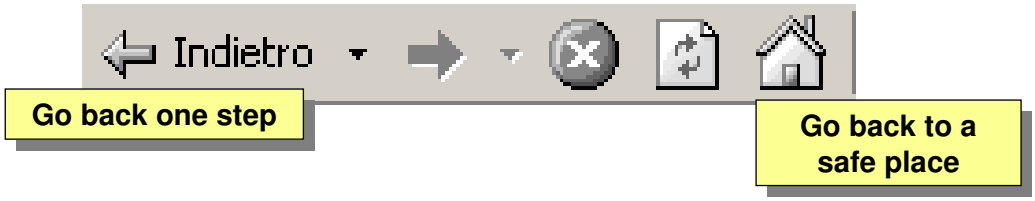
- **Forze:**
  - Un utente che esplora una situazione complessa può perdersi
  - Può dimenticarsi dove si trovava se si interrompe e riprende dopo un po' di tempo
  - Se si porta in un luogo dove non vorrebbe essere vorrà uscirne in un modo sicuro e predicibile
  - Esplorerà più volentieri l'ambiente se ha la sicurezza di poter uscire facilmente da situazioni indesiderate.
  - Ritornare indietro passo passo da un percorso di navigazione lungo può essere molto noioso.

Go back to a safe place…

- **Soluzione:** fornire un modo di tornare indietro ad un "chekpoint" scelto dall'utente (l'home page, uno stato salvato, l'inizio di una sezione…)
- **Diagramma:**

- **Contesto risultante:** si usa spesso assieme al pattern **Go Back One Step**. Anche **Interaction History** è utile, al punto forse da rendere il pattern in oggetto non necessario.

Pattern in azione: la toolbar del browser



Un esempio in formato Van Weelie: Web Application

- **Nome:** Web-based Application
- **Problema:** gli utenti hanno la necessità di eseguire compiti complessi in un sito web
- **Esempio:** Outlook Web Access
- **Quando usarlo:** il sito serve per fare cose piuttosto che per trovare informazioni. Gli utenti possono eseguire task complessi come leggere o scrivere e-mail, registrare ordini, gestire un conto corrente. Nella maggior parte dei casi prevede l'esistenza di "oggetti" che appartengono agli utenti e che devono essere creati, modificati, cancellati. Potrebbe essere una normale applicazione con l'unica differenza che gira in un browser.

Web application - 2

- **Soluzione:** strutturare l'applicazione intorno ad una serie di "viste" e consentire agli utente di lavorare all'interno di queste viste.
- Le web-applications sono basate su **View** per mostrare gli oggetti e su **Form** per modificarli. La vista costituisce un **Safe place** dove gli utenti ritornano dopo aver fatto qualcosa usando un **Form** o un **Wizard**. Le viste sono di solito liste o tabelle che consentono di controllare la rappresentazione dell'informazione (per esempio con **Table sorter** o **Table filter)**. Usare **Alternating Row Colors** per renderle più leggibili.
- E' presente un meccanismo di navigazione per passare da una vista all'altra (menu o tabs).
- **I nomi delle viste sono definiti in base agli oggetti che contengono e non in base alle azioni. Le azioni si trovano all'interno della vista.**

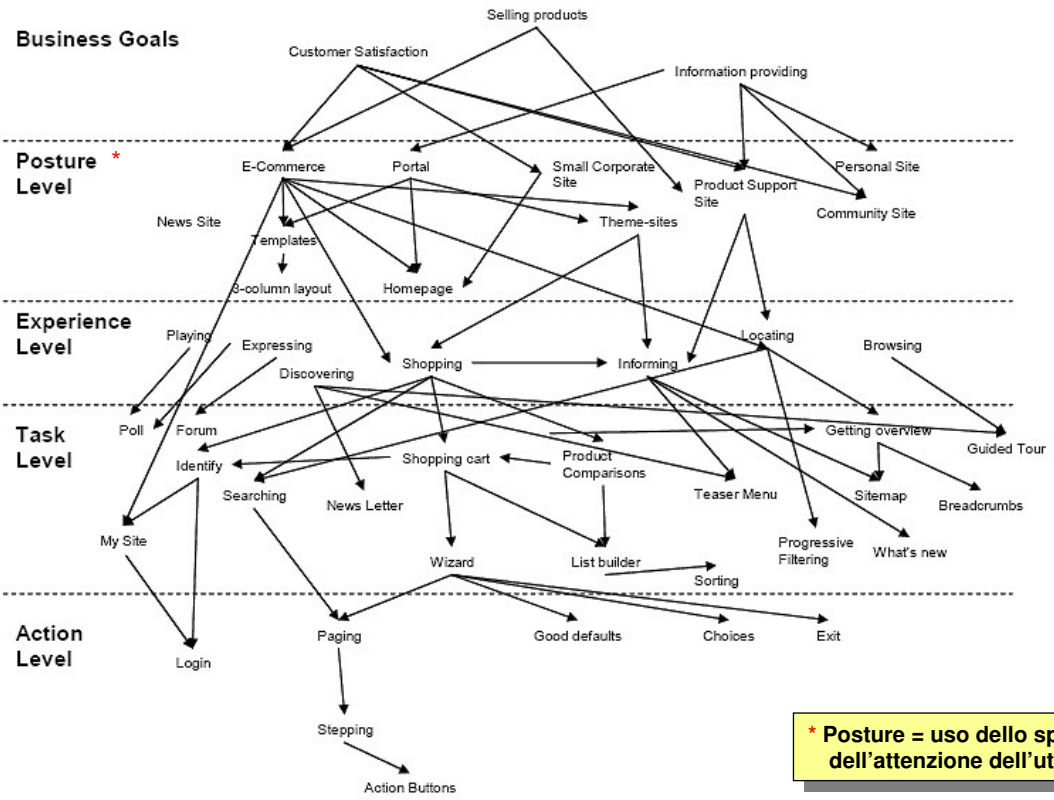
Web application - 3

- **Motivazioni:** le viste contengono oggetti di interesse e la vista deve quindi essere denominata in base agli oggetti piuttosto che alle azioni. Strutturare le web application principalmente sulle viste rende facile per gli utenti comprendere ciò che possono fare e come possono interagire
- **Altri esempi:** …

Pattern e livelli

- Come si può notare non tutti i pattern sono allo stesso livello
- Alcuni offrono soluzioni ad un aspetto ben preciso dell'interazione tra persone e computer (**Go back to a safe place**)
- Altri (**Web application**) sono di livello molto più elevato e arrivano fino a descrivere la struttura di una particolare tipologia di applicazione nel suo complesso
- Tra questi due estremi ci sono diversi livelli intermedi

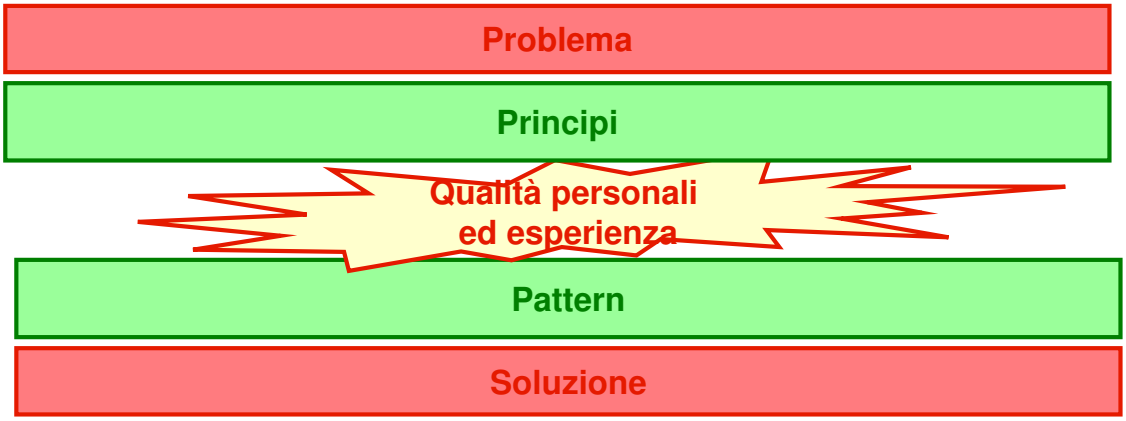
Stratificazione dei pattern (Van Weelie)



* Posture = uso dello spazio e dell'attenzione dell'utente

Top down o bottom up?

- Sono meglio i principi o i pattern?
- In realtà nessuno dei due approcci è esente da difetti
- Non sono mutuamente esclusivi ma complementari
- La cosa migliore è combinarli, in fondo l'obiettivo è quello di colmare il gap fra soluzione e problema:



Altri strumenti…

- Esistono altri strumenti a disposizione del progettista?
- Si almeno due:
  - Una buona conoscenza degli elementi di base che costituiscono l'interfaccia utente
  - La tecnologia dei componenti software
- Fanno entrambi parte di un approccio bottom-up e si situano al di sotto dei pattern

Elementi di base

- In ogni mestiere è fondamentale una conoscenza approfondita degli strumenti del mestiere
- Per chi si occupa di usabilità gli strumenti del mestiere sono, per esempio:
  - La conoscenza dei rudimenti della tipografia
    Fondamentale per la leggibilità, che rappresenta il livello zero dell'usabilità
  - Una buona comprensione del funzionamento e delle condizioni d'uso dei controlli grafici disponibili in un ambiente (widget).
    Fondamentale per rispettare gli standard e l'uniformità del "look and feel"

Esempio di elementi di tipografia: i font

- Un **font** è una serie completa di caratteri (lettere, cifre, segni si interpunzione) con lo stesso stile.
- Possono essere classificati sulla base di diversi criteri (presenza o assenza di grazie, spaziatura fissa o proporzionale):

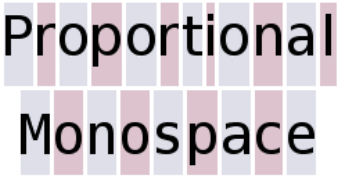
Font con grazie (o *serif*) → ABCIMN abcimn 1234 Times Roman

Font senza grazie (o *sans serif*) → ABCIMN abcimn 1234 Arial

Font monospaziato → ABCIMN abcimn 1234 Courier

Font calligrafico (o *script*) → ABCIMN abcimn 1234 Edwardian

Font decorativo (o fantasy) → ABCIMN abcimn 1234 Sand



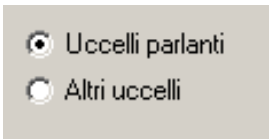
Con grazie o senza grazie?

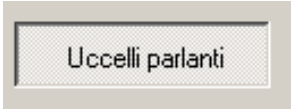
- Le grazie sono piccole decorazioni che sporgono dal corpo della lettera (in rosso nella figura a lato)
- Nei testi stampati ad alta risoluzione i caratteri con grazie risultano molto più leggibili (le grazie "guidano" l'occhio)
- Quando però si lavora a bassa risoluzione (i video dei computer sono tutti a bassa risoluzione) i caratteri senza grazie risultano molto più leggibili
- Esistono alcune font senza grazie, per esempio il **Verdana**, che sono stati pensati per essere molto leggibili anche a video e con caratteri di piccola dimensione

Conoscenza dei controlli grafici: un esempio

- **Problema:** scelta tra due opzioni

?

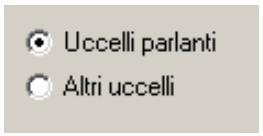




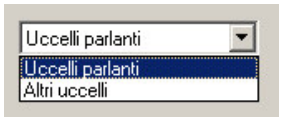
Scelta fra due opzioni…

- **Soluzione 1: checkbox**
  - Pro: semplice, poco consumo di spazio
  - Contro: esprime uno solo delle due scelte, l'altra rimane implicita

- **Soluzione 2: due radio button**
  - Pro: entrambe le scelte sono visibili ed esplicitate
  - Contro: maggior consumo di spazio



- **Soluzione 3: Combo con due righe**
  - Pro: entrambe le scelte sono esplicitate, consumo di spazio limitato e predicibile, facilmente espandibile a più di due scelte
  - Contro: una sola scelta visibile, richiede una certa destrezza nell'uso



Il ruolo dei componenti software

- La tecnologia dei componenti software è diventata negli ultimi anni un elemento essenziale dell'ingegneria del software
- I componenti rappresentano elementi di riuso di granularità intermedia fra oggetti e pattern
- In qualche modo stanno ai pattern come i pattern ai principi:
  - Un pattern ben formulato rappresenta una soluzione già pronta che rispetta uno o più principi
  - Un componente ben progettato è una soluzione gia pronta che implementa correttamente uno o più pattern
- In un processo complesso la scelta di una palette adeguata di componenti garantisce una base standardizzata per la realizzazione di applicazioni usabili

Ricapitolando: progettare l'usabilità