

## Capitolo 2

### Principi di Usabilità e Metodi di Valutazione

- 2.1 Introduzione
- 2.2 Proprietà delle interfacce 'usabili'
- 2.3 Come progettare interfacce usabili
- 2.4 Come valutare l'usabilità
- 2.5 Usabilità di Nuove Tecnologie

#### 2.1. Introduzione

In questo Capitolo, vedremo quali sono i criteri da seguire per progettare, realizzare e valutare una interfaccia 'ben fatta': e cioè che renda semplice e gradevole l'utilizzo dell'applicazione a cui si collega, da parte delle diverse categorie di utenti che la useranno. Parleremo di *usabilità* delle interfacce, partendo dalla definizione che ne ha dato l'International Standard Office (ISO: Scheda 2.1).

#### Scheda 2.1: La Definizione di Usabilità dell'ISO

"Usability is the effectiveness, efficiency and satisfaction with which specified users can achieve specified goals in a particular environment;

- *Effectiveness* is the accuracy and completeness with which users achieve specific goals;
- *Efficiency* is the accuracy and completeness of goals achieved in relation to resources employed;
- *Satisfaction* is the comfort and acceptability of using the system."

I criteri di usabilità dell'ISO stabiliscono che, per dirsi 'usabile', un'interfaccia deve innanzitutto permettere agli utenti di raggiungere i loro obiettivi in modo accurato e completo, e cioè di eseguire tutti i task per i quali l'applicazione è stata sviluppata e a cui hanno diritto di accesso. Inoltre, l'esecuzione di questi task deve essere effettuata con un minimo dispendio di risorse (tempo di apprendimento e di esecuzione). Infine, l'utente deve poter svolgere il suo lavoro nel modo più possibile confortevole e piacevole.

Rendere operativi i criteri enunciati significa definire standard precisi rispetto ai seguenti parametri:

*per l'efficienza:*

- Il *tempo di apprendimento*, che deve essere breve soprattutto per utenti non frequenti;
- Il *tempo di esecuzione*, che deve essere breve soprattutto per i task e gli utenti più frequenti;

- gli *errori* effettuati nella fase di apprendimento o di esecuzione, che devono essere minimizzati;

*per l'efficacia:*

qui, le variabili da misurare dipendono dagli obiettivi del sistema: apprendimento di concetti, ricordo di nozioni o persuasione a compiere azioni ritenute 'corrette' quando l'obiettivo è la formazione, l'informazione o il supporto alle decisioni;

*per la soddisfazione:*

- il sistema deve produrre un alleggerimento del carico di lavoro sia attraverso la riduzione dell'affaticamento che attraverso la gradevolezza dell'interazione.

Questi obiettivi definiscono criteri di valutazione dell'usabilità di una interfaccia realizzata o in fase di realizzazione: sono cioè applicabili per valutare, a posteriori, se il progetto era corretto, o per mettere a confronto soluzioni progettuali alternative mediante l'uso di metodi formali (che descriveremo nel Capitolo 4) o di studi del tipo 'Wizard of Oz'. Non aiutano invece a capire *quali regole seguire* nella fase di progettazione.

Secondo un criterio tipico delle scienze sperimentali, distingueremo fra *variabili indipendenti*, che influenzano l'andamento di un fenomeno, e *variabili dipendenti*, che descrivono il fenomeno stesso. In materia di usabilità, tempo e tasso di errore nell'apprendimento e nell'esecuzione di ogni task, livello di ricordo o di persuasione e gradevolezza dell'interfaccia sono esempi di variabili dipendenti. Discuteremo invece le possibili variabili indipendenti nella prossima Sezione.

#### Esempio sul Caso di Studio 2.1

Ricordiamo gli obiettivi del SIMB, descritti nel Capitolo 1. Misure di efficacia specifiche di questo sistema informativo sono parametri che misurano la qualità dell'assistenza (tasso e tempi medi di guarigione per classi di malattia; grado di soddisfazione dei pazienti) ma anche parametri che misurano il grado di consapevolezza del medico sulle caratteristiche dei suoi pazienti, sull'efficacia delle terapie adottate, ecc.

#### 2.2 Proprietà delle Interfacce 'usabili'

Come per altri settori dell'ingegneria del software, non esistono metodi consolidati che definiscano come progettare interfacce 'ben fatte'. L'esperienza acquisita in anni di lavoro ha prodotto una serie di *linee guida* da seguire, oltre all'indicazione di una *lista di proprietà* che sono verificabili, almeno in parte, mediante *metodi formali preempirici*. Chiameremo 'prempirici' i metodi che consentono di fare valutazioni prima di sviluppare l'interfaccia e 'empirici' i metodi sperimentali di valutazione di prototipi, che coinvolgono campioni dell'utenza prevista. Esamineremo in questa Sezione la lista di proprietà, mentre nella prossima descriveremo brevemente i principali metodi sperimentali che sono applicabili per valutare l'usabilità di prodotti realizzati. Rimanderemo invece al Capitolo 4 la descrizione dei metodi formali.

Le principali proprietà di una interfaccia usabile sono le seguenti:

#### a. *naturalzza*

##### **Definizione 2.1.**

"l'interazione con una applicazione è naturale quando non richiede all'utente di alterare in modo significativo il suo modo comune di eseguire il task corrispondente".

L'ipotesi è che ogni applicazione si sovrapponga ad una consuetudine di lavoro più o meno consolidata e che, perché il suo uso sia semplice, debba comportare il numero minimo di cambiamenti rispetto a questa consuetudine. E' chiaro che questa esigenza si scontra spesso con la necessità di rendere più razionale, con l'automazione, il lavoro svolto in precedenza. Proprio per questo, questa regola (come le successive) non va interpretata in modo rigido. Quando l'automazione introduce una nuova funzione, la naturalzza va considerata in relazione a funzioni 'esterne' simili, che l'utente svolge in altri contesti. Un esempio tipico di naturalzza è dato dal rispetto della struttura e dell'ordine di presentazione delle informazioni nella modulistica utilizzata prima dell'automazione. Un altro esempio, più di alto livello, è dato dal rispetto dell'ordine di esecuzione dei task.

##### **Esempio sul Caso di Studio 2.2**

*Primo esempio:* la visita si svolge secondo un ordine preciso: identificazione del paziente, raccolta dei sintomi e della storia clinica, esame obiettivo, analisi di eventuali risultati di test svolti, diagnosi, prescrizione di altri test e/o terapie, suggerimenti finali. Se la visita ha unicamente lo scopo di rilasciare una prescrizione per una patologia cronica, alcuni dei passi elencati possono essere saltati. L'interfaccia dovrà rispettare quest'ordine nella visualizzazione e nell'aggiornamento dei dati e consentire al medico di saltare passi non necessari o invertire l'ordine in casi particolari (urgenza, terapie ripetute ecc).

*Secondo esempio:* L'ordine di presentazione delle informazioni anagrafiche nella cartella clinica (nome, cognome, data di nascita, sesso ecc) o di inserimento dei dati relativi alla prescrizione di farmaci dovranno seguire, ove possibile, la modulistica utilizzata prima dell'introduzione del sistema informativo automatizzato, introducendo cambiamenti soltanto se necessario e comunque non quando esistano vincoli di legge per questa modulistica.

##### **Esempio 2.1**

L'organizzazione delle informazioni nella finestra mostrata in figura 2.1 non è naturale, perché non rispetta l'ordine con cui normalmente vengono definite le condizioni di stampa, mescolando la scelta della stampante con le indicazioni sul documento da stampare.

##### **Esercizio 2.1**

Riprendi il caso dell'interfaccia del tuo telefono cellulare e fai qualche esempio di non naturalzza nell'ordine di esecuzione dei task.

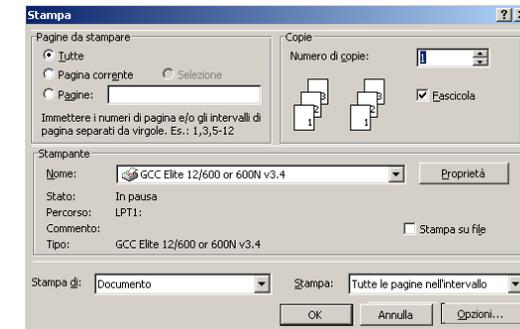


Figura 2.1: Un esempio di non naturalzza

#### b. *completezza*

##### **Definizione 2.2.**

"l'interazione è completa quando permette di effettuare tutti i task previsti nella fase di progettazione".

Ogni categoria di utenti deve poter accedere ai task ai quali è abilitato, in ogni fase dell'interazione in cui l'accesso al task è necessario. L'interfaccia non deve contenere situazioni in cui non è possibile 'uscire' dall'esecuzione di un task (anche iniziato) senza completarlo, a meno che questo non sia ritenuto necessario.

##### **Esempio sul Caso di Studio 2.3**

Consideriamo il caso di uno studio in cui la Segretaria abbia il compito di inserire, alla fine della giornata, i dati relativi a tutte le visite eseguite, alcune su pazienti già presenti nell'archivio ed altre su nuovi pazienti. Rappresentiamo il flusso di esecuzione dei task nel diagramma in figura 2.2. Questo diagramma non include la possibilità di iterare il task principale di inserimento dati, né di gestire il caso di 'pazienti ignoti' né di sospendere il task in una fase intermedia: quindi, rappresenta un caso di incompletezza. Vedremo meglio questo esempio di incompletezza in seguito, con una Rete di Petri.



**Figura 2.2.** Un esempio di incompletezza nell'esecuzione di un task

#### Esempio 2.2:

Il comando di 'Esci' deve essere disponibile in tutte le fasi dell'interazione (con o senza 'salva', a seconda dell'applicazione). Deve essere fatta una verifica dell'inserimento di tutti gli elementi di un form ritenuti necessari, con segnalazione di eventuali errori.

### c. consistenza

#### Definizione 2.3.

"l'interazione è consistente quando l'esperienza acquisita dall'utente con l'uso di una parte del sistema non viene contraddetta da cambiamenti ingiustificati introdotti in altre parti dello stesso sistema".

Questa definizione si basa sul principio che ad un utente non venga richiesto di imparare come si usa una applicazione studiando un manuale, ma che possa farlo quasi esclusivamente attraverso l'esperienza (al massimo, con l'ausilio di messaggi di help). In questa ipotesi, è essenziale che la conoscenza acquisita nell'uso di una parte del sistema sia direttamente riutilizzabile per 'indovinare le modalità d'uso' di parti ancora sconosciute. Perché questo avvenga, deve esistere una *somiglianza fra le modalità di esecuzione di task simili*: se questo accade, l'utente tenderà ad applicare le regole apprese usando alcuni task del sistema per affrontare l'esecuzione di task nuovi.

Nella consistenza, entrano in gioco almeno due fattori. Da un lato, il concetto di 'sistema'. Si può intendere, con questo termine, una singola applicazione (ad esempio, MS-Word) oppure l'insieme delle applicazioni disponibili all'interno di un ambiente (ad esempio, Office, o Windows, o addirittura l'insieme di tutti i sistemi ad interazione grafica). Nel primo caso, si parlerà di *consistenza interna* all'applicazione. Nel secondo, si parlerà di *consistenza esterna*.

L'altro fattore che caratterizza il concetto di consistenza riguarda la forma di esperienza acquisita: questa può essere *visiva* (ed è legata alla collocazione degli oggetti grafici, al linguaggio usato nei messaggi di errore, alla forma e al colore delle finestre) oppure *funzionale* (sequenza di comandi elementari necessari per eseguire un determinato task).

Un'interfaccia consistente deve possedere tutte le proprietà descritte: deve essere internamente consistente ma anche consistente con le altre applicazioni dell'ambiente in cui è inserita. Deve essere, inoltre,

visivamente e funzionalmente consistente. Per capire a fondo cosa significhi consistenza, vale la pena di riflettere su esempi (positivi e negativi) tratti da software di uso comune.

#### Esempio sul Caso di Studio 2.4

Facciamo esempi (negativi) di '*non consistenza visiva*':

- collocare il bottone di 'Exit' in posizioni diverse, in diverse fasi;
- ordinare gli elementi di un menu in modo diverso, in diverse fasi;
- assegnare nomi in inglese o in italiano, agli elementi di un menu, in diverse fasi.
- rappresentare la funzione 'trova la cartella di un paziente' con icone diverse, in fasi diverse dell'interazione;
- rappresentare la funzione 'trova la cartella di un paziente' e 'visualizza la cartella di un paziente' con inconsistenza nella rappresentazione della funzione o dell'oggetto.

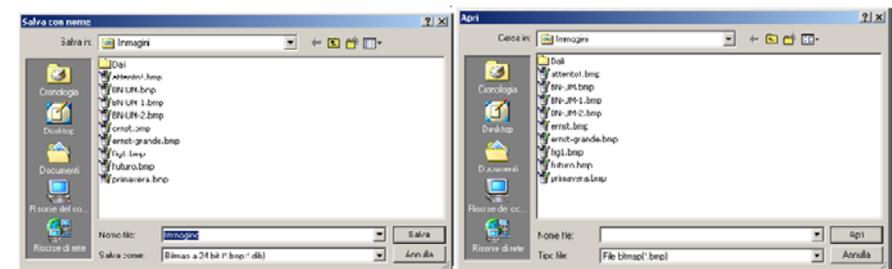
e di '*non consistenza funzionale*':

- eseguire la funzione 'uscire dal task salvando i dati introdotti', mediante un solo comando (esempio, bottone 'Quit') o mediante due comandi successivi, (esempio, bottone 'Save' + bottone 'Quit'), in fasi diverse;
- imporre un ordine prefissato / libero nell'inserimento dei dati nello stesso form, in fasi diverse.

#### Esempio 2.3:

Un esempio di *consistenza visiva interna* :

prova ad esaminare le funzioni di 'Gestione del file-immagine' in Paint. Noterai che le popup window che si aprono in corrispondenza delle funzioni di 'apri', 'salva con nome', ecc (figura 2.2) hanno tutte lo stesso aspetto.



**Figura 2.3.** un esempio di consistenza visiva interna in Paint

**Esempio 2.4:**

Ragioniamo sulla *consistenza visiva esterna*: Prova a confrontare Word, WordPad e BloccoNote per Windows 2000 (figura 2.3). Osserva la toolbar principale di bottoni (File, Modifica, ecc) in WordPad: noterai che è un sottoinsieme di quella di Word e che gli elementi sono posti nello stesso ordine. Lo stesso accade per le sottostanti toolbar di icone, con qualche eccezione: lo stile 'corsivo' è indicato in modo diverso, WordPad include una icona di 'trova' e un'icona di inserimento di 'data e ora' che Word ha soltanto fra gli elementi del menu a tendina di 'Inserisci'.

Confronta ora i menu a tendina aperti da 'File' in Word e Blocco Note: noterai che gli elementi comuni alle due applicazioni sono presentati nello stesso ordine e chiamati con gli stessi nomi. Nei due menu a tendina aperti da 'Modifica', invece, la stessa funzione si chiama 'Cancella' in un caso ed 'Elimina' nell'altro.

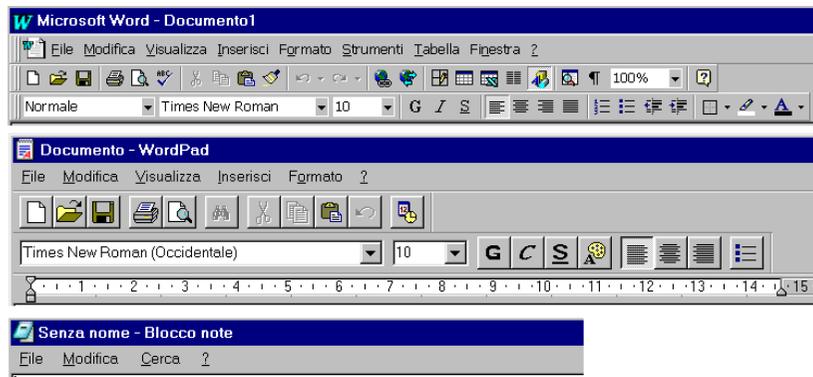


Figura 2.4. consistenza/inconsistenza visiva fra i Text Editor di Windows

**Esempio 2.5:**

Ragioniamo sulla *consistenza funzionale esterna*: la funzione di 'Trova' viene eseguita con 'CTRL + MAIUSC + T' in Word e con 'CTRL + T' in Blocco Note. La funzione di 'Vai a' viene eseguita in Word con il tasto-funzione F5, in Blocco Note con la combinazione 'CTRL+G', in Word con il tasto-funzione F5, in Blocco Note con la combinazione 'CTRL+G'.

**Esercizio 2.2**

Prova ad individuare esempi di inconsistenza funzionale interna in Paint ed esterna fra diverse applicazioni di Window. Ad esempio: Paint e Word, oppure Paint e Imaging.

**d. non complessità:****Definizione 2.5.**

"una interazione è non complessa quando permette di effettuare le funzioni previste con un 'ragionevole' grado di efficienza".

La complessità è un altro fattore che influenza il carico cognitivo di chi usa un'applicazione, influenzando quindi l'efficienza del lavoro svolto. Ma in che modo questo avviene e cosa significa grado 'ragionevole' di efficienza, in questo caso? A parità di caratteristiche ergonomiche della tecnologia usata, il carico cognitivo determinato dall'uso di una applicazione è funzione, allo stesso tempo, della complessità visiva e funzionale dell'interfaccia. La *complessità visiva* dipende dal numero di oggetti simultaneamente presenti in un display, mentre la *complessità funzionale* dipende dal numero di azioni elementari che l'utente deve eseguire per completare un task. Una bassa complessità visiva consente all'utente di individuare con facilità gli oggetti di cui ha bisogno per eseguire il suo task. Una bassa complessità funzionale gli permette di completare il task con poche azioni. Nei messaggi in linguaggio naturale, la complessità è legata alla lunghezza del messaggio e alla sua chiarezza: troveremo un esempio di messaggio in linguaggio naturale troppo lungo nel Capitolo 5.

**Esempio sul Caso di Studio 2.5**

Confrontiamo i due modi seguenti di compiere il task di 'Ricerca della cartella di un paziente':

- *click sull'icona "Ricerca di una Cartella", nella finestra principale:*  
apertura di una finestra 'dati di identificazione', che contiene cognome, nome, data di nascita, n.cartella;
- *inserimento di 'cognome, nome e data di nascita (oppure di 'numero cartella'); conferma;*  
visualizzazione della prima pagina della cartella, che contiene (oltre ai dati di identificazione suddetti) sesso, stato civile, lavoro svolto, e lista sintetica dei principali problemi di salute del paziente.

*Oppure*

- *click sul bottone "Gestione dell'Archivio", nella finestra principale;*  
apertura di un menu a tendina, con gli item: 'Ricerca', 'Creazione', ecc;
- *click sull'item "Ricerca";*  
apertura di una finestra di pop-up con la domanda: "Conosci il numero della cartella?"
- *Risposta si/no*  
apertura di una finestra con i soli dati d'identificazione scelti.

Notiamo il trade-off fra complessità visiva e funzionale!

Nel definire gli standard per i tempi di esecuzione dei vari task in questo sistema informativo, terremo conto della loro frequenza. Le funzioni più frequenti (come la ricerca e la visualizzazione di una cartella) dovranno avere dei tempi di esecuzione il più possibile brevi, per ridurre possibili effetti negativi sul rapporto medico-paziente dovuti al medico che 'osserva' il sistema piuttosto che il paziente. Gli standard dovranno essere fissati dopo aver osservato attentamente la durata delle visite e delle diverse fasi, ponendosi l'obiettivo di ridurre i tempi d'interazione con il documento per lasciare spazio maggiore all'interazione con il paziente.

**Esempio 2.6**

La complessità visiva di Blocco Note è minima. L'interfaccia principale contiene soltanto tre oggetti grafici, oltre alla funzione di display. La complessità visiva di WordPad e di Word sono via via crescenti, anche a causa della crescente complessità delle due applicazioni. Tuttavia, a causa di questa complessità, anche utenti frequenti di Word non conoscono la funzione associata a diversi oggetti grafici presenti nell'interfaccia.

**Esempio 2.7**

Un esempio di inutile complessità è in Paint, che non adatta automaticamente la dimensione della finestra 'attiva' all'immagine da incollare.

**Esercizio 2.3**

Considera una delle applicazioni che usi comunemente e prova a verificare se ci sono delle funzioni, incluse in questa applicazione, che non conosci. Chiediti se avevi veramente bisogno di utilizzarle oppure se non le hai utilizzate semplicemente perché non erano rappresentate in modo adeguato nell'interfaccia.

Per ridurre la complessità visiva di una interfaccia, si possono 'nascondere' gli oggetti corrispondenti alle funzioni meno frequenti, rendendoli visibili soltanto su richiesta dell'utente. Oppure, si possono distribuire gli oggetti fra diverse finestre. In questo modo, la complessità visiva si riduce, ma la complessità funzionale aumenta: si parla quindi di 'trade-off' (o 'baratto') fra complessità visiva e funzionale.

Consideriamo l'interfaccia grafica di una applicazione T che comprenda n task, rappresentati dagli oggetti grafici  $O_1, O_2, \dots, O_i, \dots, O_n$ . Possiamo progettare l'interfaccia secondo due modalità:

- mostrare nell'interfaccia principale tutti gli oggetti  $O_1, O_2, \dots, O_n$  che rappresentano le n funzioni disponibili (ad esempio in una o più toolbar), come in figura 2.5.a, oppure
- organizzare le n funzioni in m classi  $C_1, C_2, \dots, C_j, \dots, C_m$  e mostrare nell'interfaccia principale soltanto queste classi rendendo le singole funzioni accessibili in display successivi (menu a tendina, finestre, ecc), come in figura 2.5.b

Nel primo caso, adotteremo la soluzione con complessità visiva massima (almeno n oggetti mostrati in ogni fase dell'interazione). Nel secondo, la complessità visiva si ridurrà ad una media di  $(n/m + m)$ .

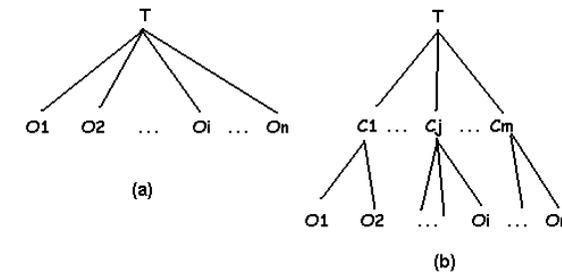


Figura 2.5: Rappresentazione dei task in una interfaccia

Tuttavia, se nella seconda soluzione la complessità visiva si riduce, la complessità funzionale aumenta. Infatti, per 'raggiungere' l'oggetto  $O_i$  l'utente dovrà eseguire i comandi necessari per percorrere in discesa la gerarchia delle funzioni. E, una volta terminato il task associato ad  $O_i$ , dovrà eseguire i comandi necessari per ritornare alla 'radice' della gerarchia. La scelta del giusto punto di equilibrio fra le due fonti di complessità deve tener conto dell'associazione fra task. Se la classe di task  $C_k$  viene eseguita, in genere, in momenti separati dall'esecuzione dei task appartenenti alla classe  $C_j$ , sarà sufficiente mostrare, nel contesto dell'interazione relativa a  $C_j$ , soltanto l'oggetto che corrisponde a  $C_k$ . Ancora una volta, quindi, l'analisi della gerarchia dei task è elemento essenziale per le fasi successive del progetto.

**Esempio sul Caso di Studio 2.6**

Riprendiamo la gerarchia dei task di SIMB descritta nel Capitolo 1. Si può dire che in questo caso, vista la complessità dell'albero, non sia logico adottare una soluzione di tipo (a). Sarà preferibile scegliere, invece, una strategia in cui si tenga conto delle diverse frequenze di esecuzione dei task e quindi vengano mostrati simultaneamente, oltre agli oggetti relativi ai subtask di primo livello ('Aggiornare il modello di utente', 'Aiutare il medico nelle sue decisioni' ecc), anche i subtask di 'Gestire l'archivio' ('Trovare una cartella', 'Visualizzarne ed aggiornarne il contenuto', ecc).

**Esempio 2.8:**

La funzione 'visualizza' di Paint permette all'utente di adattare la complessità visiva dell'interfaccia alle sue esigenze, mostrando o nascondendo parte delle funzioni eseguibili.

**Esercizio 2.4**

Prova a riorganizzare in un solo livello le funzioni di Blocco Note e confronta le complessità visiva e funzionale delle due soluzioni.

e. **non ridondanza:****Definizione 2.5.**

"una interazione è non ridondante quando richiede all'utente di fornire il minimo indispensabile d'informazioni necessarie per attivare ciascuna delle funzioni disponibili e presenta soltanto le informazioni realmente necessarie".

E' possibile fare innumerevoli esempi di violazione di questa proprietà: richiesta di data di nascita ed età quando la data attuale è nota, richiesta di conferma per operazioni 'non rischiose' (come il salvataggio di dati nuovi) o di dati che possono essere definiti 'per default' (come il tipo di stampante da utilizzare). Evidentemente, il rispetto di questa proprietà determina una riduzione della complessità funzionale e visiva.

**Esempio sul Caso di Studio 2.7**

Alcuni esempi di ridondanza da evitare:

- *Inserimento* dei dati di identificazione: data di Nascita ed età'
- *Richiesta di conferma* per operazioni 'non rischiose' (esempio, "Vuoi davvero salvare questi dati?")
- *Visualizzazione* di informazioni non necessarie: dati dettagliati relativi ai 'problemi' anziché una semplice lista nella finestra principale.

**Esercizio 2.5**

Fai qualche esempio di ridondanza, in un software che conosci.

f. **assistenza:****Definizione 2.6.**

"una interazione deve assistere l'utente fornendo diversi tipi di aiuto durante l'esecuzione:

elenco delle funzioni attivabili

messaggi di errore

richiesta di conferma delle richieste 'rischiose'

informazione sullo stato del sistema (soprattutto per le operazioni lunghe)

possibilità di annullare uno o più comandi errati (funzione di 'undo')."

Esempi classici di assistenza sono le label, mostrate su passaggio del mouse, che descrivono le funzioni attivabili dalle icone, oppure i messaggi sullo stato del sistema mostrati in figura 2.6. La figura 2.7 mostra invece due esempi di messaggi di errore.

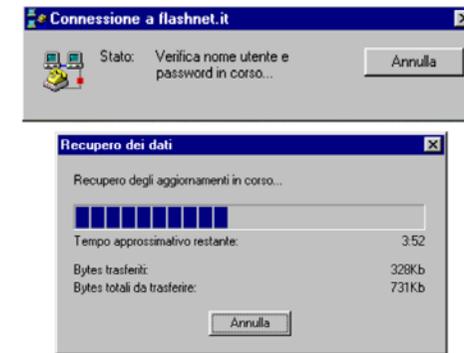


Figura 2.6: Messaggi d'informazione sullo stato del sistema

Notare che, nei messaggi d'informazione sullo stato del sistema, è sempre inserita la possibilità di interrompere l'esecuzione della funzione in corso (bottone di 'Annulla'): questo consente di disattivare procedure non essenziali che richiedano tempi di esecuzione ritenuti eccessivi dall'utente. Nei messaggi di pericolo o di errore, invece, è sempre inserita una richiesta di conferma (bottone di 'OK') che ha lo scopo di assicurare al sistema che l'utente abbia letto il messaggio.

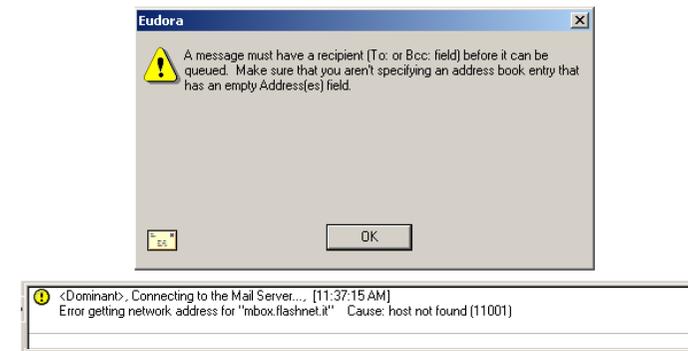


Figura 2.7: Messaggi di 'segnalazione di errore'

**Esempio sul Caso di Studio 2.8**

Alcuni esempi di funzioni di assistenza che il SIMB dovrebbe includere:

- Nel task di 'ricerca cartella', messaggi del genere: "L'archivio delle cartelle è complesso: l'operazione di ricerca può richiedere un certo tempo" oppure "Il numero della cartella è errato: deve contenere almeno 7 caratteri", oppure "Non esiste nessun Mario Rossi nell'archivio, ma c'è una Maria Rossi: sei sicuro di non aver sbagliato?..."
- Nel task di 'cancellazione di cartella', il tipico messaggio: "Sei sicuro di voler cancellare questa cartella?". Lo stesso messaggio sarebbe invece ridondante per funzioni meno rischiose, come la cancellazione di singoli dati, la visualizzazione di parti della cartella, ecc.

**g. flessibilità:****Definizione 2.7.**

"una interazione è flessibile quando è in grado di cambiare il suo comportamento in relazione alle caratteristiche di diverse categorie di utenti".

Questa proprietà richiede che l'interfaccia sia capace di *riconoscere* l'utente con cui sta interagendo e di *scegliere*, fra diverse alternative, quella che meglio si adatta alle sue esigenze. Una forma semplice di flessibilità è quella realizzata nella maggior parte delle interfacce grafiche, nelle quali le funzioni più comuni possono essere attivate mediante la manipolazione di oggetti grafici o mediante comandi da tastiera. Questa forma di flessibilità nasce dai risultati di una serie di esperimenti, che hanno mostrato che la forma d'interazione preferita dagli utenti (con mouse o da tastiera) dipende dal loro livello di esperienza. La maggior parte dei sistemi ad interazione grafica, tuttavia, non include funzioni di 'riconoscimento' delle caratteristiche degli utenti e di 'adattamento automatico' delle modalità d'interazione: lascia piuttosto agli utenti la possibilità di scegliere in modo autonomo la modalità da adottare. Analoga considerazione vale per le comuni funzioni che consentono di adattare gli oggetti mostrati nelle toolbar. Descriveremo invece più dettagliatamente (nei Capitoli 3, 5 e 6) come si possano progettare e realizzare sistemi che adottano forme più raffinate di personalizzazione.

**Esempio sul Caso di Studio 2.9**

Nel SIMB, alcune funzioni richiedono competenze mediche (ad esempio, la codifica delle diagnosi o l'inserimento di dati relativi alla visita); altre (come la ricerca di una cartella o l'analisi statistica dei dati) richiedono soltanto competenze nell'uso di sistemi computerizzati, che una Segretaria può acquisire facilmente. Le competenze menzionate possono variare nel tempo e a seconda della tipologia di medici e dell'organizzazione dell'ambulatorio. Realizzare un unico sistema che possa essere utilizzato in entrambe le situazioni organizzative descritte richiede la progettazione di un'interfaccia flessibile, e cioè adattabile alle diverse categorie di utenti.

**Esercizio 2.6**

Fai qualche esempio di flessibilità, in un software che conosci. Quali funzioni sono attivabili sia da tastiera che da mouse? In quali casi e in che senso è possibile modificare le toolbar del sistema? Conosci dei siti web personalizzati? Quali forme di personalizzazione adottano?

**2.3. Come progettare interfacce usabili**

Progettare interfacce che posseggano le proprietà che abbiamo descritto nella Sezione precedente richiede metodo, esperienza e spirito critico. Imparare a riconoscere elementi di 'non usabilità' nelle interfacce di uso comune aiuta certamente a evitare errori nella progettazione di nuovi sistemi. In questo senso, è sicuramente utile leggere gli esempi pubblicati sul sito che abbiamo segnalato nella Scheda 1.1. Questi esempi mostrano che gli errori sono diffusi nelle interfacce di sistemi anche noti, molto di più di quanto non s'immagini. Il risultato che gli utenti di queste applicazioni notano, però, è una più o meno grave difficoltà d'uso: errori frequenti, tempi lunghi di apprendimento o di esecuzione, generica fatica o poca gradevolezza del lavoro.

Per aiutare a mettere in atto il rispetto delle proprietà di cui abbiamo parlato, alcuni esperti hanno definito delle 'linee-guida', o 'regole': le più note, le otto 'Regole d'Oro' proposte da Schneiderman (vedi Scheda 2.2).

**Scheda 2.2: Le otto 'Regole d'oro' per la progettazione di interfacce usabili**  
 proposte da B Schneiderman

- Strive for *consistency*: use consistent sequences of actions in similar situations, identical terminology in prompts, menus etc, consistent commands throughout.
- Enable frequent users to use *shortcuts*: abbreviations, special keys hidden commands and macro facilities reduce the number of interactions and the pace of interaction.
- Offer *informative feedback*: for every operator action, there should be some system feedback.
- Design dialogs to yield *closure*: the informative feedback at the completion of a group of actions gives the operator the satisfaction of accomplishment and the indication that the way is clear to prepare for the next group of actions.
- Offer simple *error handling*: the user should not have to retype the entire command, but only need repair the faulty part.
- Permit easy *reversal of actions*: this relieves anxiety, since the operator knows that errors can be undone; the units of reversibility may be a single action, a data entry or a complete group of actions.
- Support *internal locus of control*: users should be made the initiators of actions rather than the responders.
- Reduce short-term *memory load*: displays should be kept simple, frequent window motion should be reduced, online access to command syntax forms etc should be provided.

Come si può notare, alcune di queste regole fanno esplicito riferimento alle proprietà di cui abbiamo parlato; *consistency* è il nostro concetto di consistenza, *memory load* richiama la complessità mentre *informative feedback*, *closure*, *error handling* e *reversal of actions* sono alcune delle forme di assistenza che abbiamo menzionato. Infine, gli *shortcut* sono uno dei modi per ridurre la complessità funzionale.

Applicare, nella progettazione dell'interfaccia, i risultati della fase di *user e task analysis* aiuta a rispettare le proprietà di usabilità:

- l'osservazione del modo in cui gli utenti lavorano permette di definire modalità d'interazione che rispettino il più possibile le regole di comportamento acquisite e quindi siano *naturali e complete*. Il grado di *flessibilità* che l'interazione deve possedere dipende da quanto diverse sono le caratteristiche degli utenti potenziali individuati: in particolare, i loro livelli di esperienza e le funzioni che ogni categoria comunemente svolge o è abilitata a svolgere.
- Le funzioni che l'applicazione è in grado di compiere devono essere visualizzate in modo chiaro per l'utente e secondo criteri di uniformità. L'uniformità della rappresentazione contribuisce in modo determinante alla *consistenza visiva*: per questo (come abbiamo detto nel Capitolo 1), nel progettare un'interfaccia occorre, prima di tutto, definire una *metafora*<sup>1</sup> d'interazione e cercare di rispettarla in tutte le fasi: la metafora stabilisce una corrispondenza fra task da rappresentare e oggetti che li rappresentano. L'esempio più noto, in questo campo, è quello di Word (o anche WordPad) nel quale la metafora applicata è quella della 'scrivania': i simboli utilizzati per rappresentare le diverse funzioni riproducono gli oggetti che normalmente si trovano (o si trovavano) sulla scrivania di chi compone un documento: dischetti, stampanti, archivi cartacei, forbici, eccetera. Questi simboli devono essere collocati sul display secondo una logica che rispecchi l'organizzazione gerarchica dei task.
- Il modo di raggruppare i subtask rappresentati in un singolo layout dev'essere definito cercando di conciliare, allo stesso tempo, esigenze di *complessità visiva e funzionale* e tenendo conto, di nuovo, della struttura della gerarchia dei task: non troppi oggetti allo stesso tempo ma sequenze brevi di comandi elementari, soprattutto per eseguire i task più comuni; raggruppamento, nello spazio, di task che appartengono alla stessa categoria, ecc.

Per rispettare esigenze di *consistenza*, funzioni simili dovranno essere rappresentate in modo simile e, se presenti nello stesso layout, in posizioni vicine fra loro (ad esempio, creando gruppi di icone nelle toolbar). Se una funzione è attivabile in diverse fasi dell'interazione, l'oggetto grafico che la rappresenta deve essere sempre lo stesso e deve essere collocato sempre nella stessa posizione.

Vediamo ora un esempio che illustra se e come questi criteri sono stati rispettati nella progettazione di un software di uso comune mentre vedremo meglio, in seguito, come il linguaggio grafico (simboli con cui rappresentare i diversi task) possa essere definito in modo da rispettare criteri di *consistenza visiva*.

### Esempio 2.9:

Consideriamo l'unica finestra in cui si svolge l'intero lavoro di Paint, che è mostrata in figura 2.7. Si nota, innanzitutto, che la metafora d'interazione utilizzata è quella del 'tavolo da disegno', in cui sono presenti gomma, penna, pennelli, lente d'ingrandimento, barattolo di colori. Questa finestra mostra tre toolbar (oltre alla barra di stato):

<sup>1</sup> Secondo il Dizionario Zanichelli, la *metafora* è una "Figura retorica che consiste nel sostituire una parola o un'espressione con un'altra in base ad un rapporto di palese o intuitiva analogia fra i rispettivi significati letterali". L'uso di questo termine in HCI è descritto nella Scheda 1.5.

- Un *menu di bottoni*, in alto, che rappresentano i task che consentono di manipolare il file-immagine, l'immagine o sue parti;
- Un *menu di icone* (verticale, a sinistra), che rappresentano i task che consentono di 'costruire' l'immagine;
- una *toolbar in basso*, che mostra la tavolozza di colori.

A ciascuno dei bottoni della prima toolbar è associato un menu a tendina, che specifica i subtask nei quali si decompone il task relativo al bottone. Ad esempio:

```
File → Nuovo
Apri
Salva
Salva con nome
.....
Anteprima di stampa
Imposta pagina
Stampa
```

Applicando la stessa analisi agli altri elementi nelle tre toolbar, si può notare che l'interfaccia di Paint rappresenta la gerarchia dei task mostrata in figura 2.8. Task e subtask di 'Organizza finestra di lavoro' sono associati al bottone 'Visualizza'. Task e subtask di 'Gestisci file immagine' sono associati al bottone 'File'. Task e subtask di 'Gestisci immagine o sue parti' sono associati in parte ai due bottoni 'Modifica' e 'Immagine', in parte alle altre due toolbar. In particolare, le icone nella toolbar verticale sono organizzate in modo da localizzare funzioni simili in posizioni vicine fra loro (vedi figura 2.9).

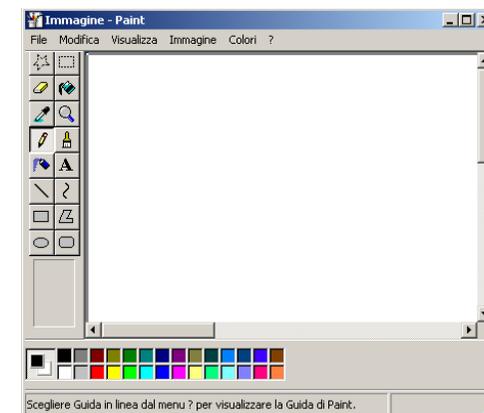


Figura 2.8: la finestra principale di Paint

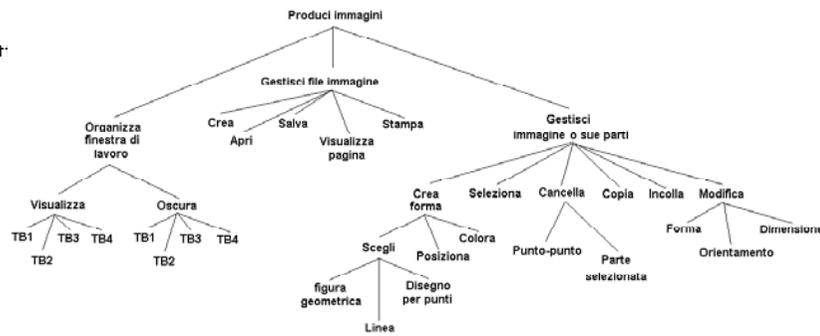


Figura 2.9: gerarchia dei task in Paint



Figura 2.10: Funzioni di 'Crea immagine' in Paint

### Esercizio 2.7

Analizza le toolbar di Word e WordPad in figura 2.3. Noterai che, nelle toolbar di icone, gli oggetti sono disposti in gruppi: qual è, secondo te, la logica seguita in questo raggruppamento?

## 2.4. Come valutare l'usabilità

Riprendiamo la distinzione, introdotta all'inizio, fra variabili indipendenti e dipendenti che caratterizzano l'usabilità. Ricordiamo che le *variabili indipendenti* sono quelle che influenzano l'andamento di un fenomeno (nel nostro caso, l'usabilità), mentre le *variabili dipendenti* descrivono il fenomeno stesso. Nella Sezione 2.2 abbiamo elencato le principali variabili indipendenti (natura, completezza, consistenza, non

complessità, non ridondanza, assistenza e flessibilità) ed abbiamo dato delle linee-guida su come realizzare interfacce che abbiano queste proprietà. Ragioneremo ora su come valutare ipotesi di progetto o prototipi in funzione delle due categorie di variabili.

Una premessa generale: indipendentemente dal metodo utilizzato, occorre ricordare che una valutazione è sempre *comparativa*: l'ipotesi esaminata va cioè confrontata con uno *standard* definito in fase di progetto, oppure con una *soluzione alternativa*. Lo standard viene definito per ogni task in termini di valori accettabili di ciascuna delle variabili dipendenti considerate (tempo di apprendimento ed esecuzione, rischio di errore, livello di persuasione ecc). I valori sono stabiliti in base all'analisi di come gli utenti potenziali svolgono il loro lavoro: dipendono dalla *frequenza* con cui il task viene eseguito, dalle *condizioni* in cui il lavoro viene eseguito e dall'importanza di ciascuno degli obiettivi definiti. Per i task più frequenti, se l'obiettivo dell'automazione è estendere le funzioni esistenti, un criterio minimo è che il tempo di esecuzione dopo l'automazione non peggiori. Se invece l'obiettivo principale è accelerare i tempi di esecuzione, gli standard saranno (ovviamente) più stringenti. Il tempo di apprendimento sarà invece un parametro critico di valutazione per i task meno frequenti. La gradevolezza dell'interazione sarà importante in applicazioni che sono orientate ad attività voluttuarie (ad esempio, un'agenda elettronica per adolescenti, un gioco di ruolo) ma anche in quell'insieme di applicazioni che si chiamano di 'edutainment', in cui si mescolano obiettivi di formazione ed intrattenimento (ad esempio, l'insegnamento della matematica ai bambini). Nei sistemi di supporto alle decisioni, il livello di persuasione sarà il parametro fondamentale da misurare, ... eccetera.

Se l'interazione viene realizzata seguendo il metodo della progettazione iterativa, dopo la prima fase di analisi di utenti e task sarà necessario intrecciare fasi di progettazione ed eventualmente di realizzazione di prototipi con fasi di valutazione. La valutazione può essere orientata a verificare l'usabilità di un prototipo già realizzato oppure a mettere a confronto ipotesi di progetto prima ancora di tradurle in prototipo. Realizzare studi sperimentali di valutazione è lungo e costoso, così come realizzare prototipi: quindi, il risparmio di risorse si basa su una combinazione di metodi di valutazione diversi. Nel descriverli, distingueremo le seguenti tipologie:

### a *metodi analitici:*

non coinvolgono un campione dell'utenza finale ma esperti nel settore o progettisti. Possono essere effettuati mediante:

#### a.1. *valutazione da parte di esperti ('Walkthrough')*

In questo primo (e più ovvio) metodo *analitico*, il progettista e un gruppo di esperti 'scorrono' i diversi task dell'applicazione simulandone l'uso da parte dei diversi utenti potenziali e annotando i problemi di usabilità riscontrati. Lo studio si conclude con un *rapporto*.

#### a.2. *valutazione 'formale' o semi-formale*

Questo metodo analitico si basa su una descrizione (formale o semi-formale) del sistema, che consente di *predire come il sistema si comporterà* in diversi contesti. Possono essere utilizzati, in questi studi, i metodi formali che descriveremo nel Capitolo 4 (estensioni di Grammatiche e Automi): i risultati che si possono ottenere dipendono, come vedremo, dal metodo utilizzato, e comprendono valutazioni di

variabili indipendenti (come la consistenza o la complessità) e, in altri casi, stime di variabili dipendenti, come i tempi di esecuzione. Il loro *vantaggio* principale è l'economicità: si può fare una prima valutazione del sistema già nella fase di progetto, prima di realizzare prototipi. Il loro *svantaggio* principale è nella difficoltà di prevedere come realmente gli utenti utilizzeranno il sistema finale: quali errori faranno, quali modi d'interazione privilegeranno, eccetera.

**b. metodi sperimentali o empirici:**

I metodi sperimentali hanno lo *svantaggio* principale di essere lunghi e costosi da realizzare, con il rischio di produrre risultati errati se non ben progettati. Il loro principale *vantaggio* è che esaminano situazioni in cui effettivamente il sistema verrà utilizzato e forniscono risultati quantitativi. Coinvolgono un numero statisticamente significativo di 'soggetti' rappresentativi dell'utenza potenziale. Ad ogni soggetto viene chiesto di compilare un questionario preliminare, che ha la funzione di raccogliere i dati principali sull'utente stesso; gli o le viene quindi chiesto di utilizzare la parte di sistema che si vuole valutare. L'esperimento si completa con la raccolta di due tipi di dati:

**b.1. valutazioni mediante osservazione:**

viene osservato il comportamento del soggetto durante l'esecuzione del o dei task inclusi nello studio e vengono misurati tempi e frequenze di errori; queste misure possono essere effettuate automaticamente e in modo trasparente al soggetto (cioè da programma) oppure impiegando un osservatore esterno.

**Esempio 2.10**

Un metodo comunemente adottato per valutare un sito web consiste nel tener traccia dei tempi di osservazione di ogni pagina, oltreché del percorso di navigazione. I tempi misurano la complessità di ogni singolo nodo, mentre il percorso permette di rilevare errori nella navigazione, misurando così indirettamente la naturalezza dell'interazione.

**b.2. valutazioni mediante questionario:**

le variabili indipendenti e/o dipendenti sono misurate mediante un questionario compilato da ogni soggetto alla fine della fase d'interazione, le cui domande si propongono di valutare ricordo, apprendimento, persuasione, oltre a stime soggettive di utilità del sistema, di gradevolezza dell'interazione, eccetera. Ad esempio: "Quanto hai trovato facile/difficile eseguire questo compito?". Oppure: "Quanto hai trovato facile/difficile imparare ad eseguirlo?", "Quanto hai trovato naturale l'interazione?", "Quanto hai trovato piacevole usare il sistema?", "Credi che esegueresti l'azione che ti è stata suggerita?", ecc. La stima soggettiva di ciascuno di questi parametri viene quantificata in una scala numerica (detta *scala di Likert*) i cui valori interi variano in un intervallo la cui ampiezza dipende dalla precisione con cui si vuole (e si ritiene di poter) effettuare la valutazione (da (1-3) a (1-9)). Domande con risposte chiuse possono essere utilizzate, invece, per valutare il grado di ricordo o di apprendimento.

**Esempio sul Caso di Studio 2.10**

Una valutazione empirica può essere effettuata, per il SIMB, mettendo a confronto uno o più prototipi che differiscano per il mapping fra task e oggetti grafici (control), per il raggruppamento di control nell'interfaccia, per il linguaggio dei comandi ecc. Alternativamente, un singolo prototipo potrà essere valutato rispetto agli standard (per tempi di esecuzione e errori) definiti inizialmente. In questo caso, gli standard definiti saranno diversi in relazione alla frequenza di esecuzione dei task e ai rischi che eventuali errori comportano. Ad esempio: saranno definiti tempi brevi per la ricerca e la visualizzazione delle cartelle; più lunghi per l'elaborazione statistica dei dati; rischio minimo di errore per diagnosi e terapia; più elevato per i dati anagrafici, ... ecc

**Esempio 2.11:**

Analizziamo i questionari che sono stati utilizzati in uno studio di valutazione condotto all'Università di Reading: l'obiettivo del sistema era convincere i soggetti ad adottare una dieta 'corretta'. Lo studio metteva a confronto due messaggi, il primo dei quali informava sulle conseguenze positive di una dieta 'corretta', mentre il secondo mostrava le conseguenze negative di una dieta 'scorretta'. L'obiettivo dello studio era determinare l'alternativa più efficace, da adottare quindi nel sistema finale.

La Scheda 2.3 mostra il questionario preliminare, orientato a conoscere le caratteristiche principale dei soggetti inclusi nello studio (età e sesso) e a valutare il loro grado di conoscenza delle diete corrette, prima di leggere il messaggio. I dati raccolti con questo questionario venivano utilizzati per verificare che i due gruppi di soggetti (conseguenze positive vs conseguenze negative) fossero confrontabili.

La scheda 2.5 mostra il questionario principale: qui, le domande 1-7 mostrano esempi di valutazioni soggettive di persuasione (1), soddisfazione (2), comprensibilità del messaggio (3), ecc, con scale di Likert. Le domande 8-11 hanno, invece, l'obiettivo di rilevare il grado di ricordo delle informazioni contenute nel messaggio, da parte dei soggetti inclusi nello studio. L'analisi dei dati confrontava i valori di queste variabili nei due gruppi di soggetti (conseguenze positive vs conseguenze negative). L'alternativa scelta era quella in cui le valutazioni soggettive e il grado di ricordo avevano valori più elevati, con livello di significatività statistica accettabile).

**Esercizio 2.8**

Progetta uno studio di valutazione empirico per il sistema di descrizione di oggetti in un Museo e per una biglietteria automatica in una stazione ferroviaria. Quali sono le differenze? Cosa misureresti nei due casi? Come? Con quali tipi di questionario o di osservazione diretta? Quali situazioni metteresti a confronto nei due casi?

**b.3. valutazioni mediante simulazione ('Wizard of Oz')**

Questo metodo può essere applicato *prima di realizzare un prototipo*, quando la sua realizzazione è complessa e costosa. Si simula l'uso del prototipo da parte di un campione di utenti che utilizzano quello che *credono* essere un sistema implementato, mentre un programmatore ben addestrato (il 'mago'), che

opera in una stazione remota, inserisce manualmente le risposte ai loro comandi<sup>2</sup>. Si tratta quindi, in sostanza, di un metodo allo stesso tempo sperimentale (perché basato sulla realizzazione di uno studio su un campione di utenti) ma che può essere effettuato in una fase intermedia dello sviluppo del progetto.

Studi basati su questo metodo richiedono la realizzazione di un software basato sull'architettura illustrata nella figura 2.11. I soggetti coinvolti nello studio (il campione di utenti) utilizzano il lato client, mentre il 'mago' (un esperto nel dominio che conosce bene il sistema) utilizza il lato server. Sul lato client, viene mostrata l'interfaccia del sistema che si intende valutare. Sul lato server, vengono mostrate le risposte fra le quali il mago deve scegliere quella più opportuna da mostrare al soggetto. Alla fine dell'esperimento di interazione, al soggetto viene chiesto di compilare un questionario del tipo di quelli descritti nella prossima sezione. La Scheda 2.3 illustra un tool per lo sviluppo di studi WoZ su 'Agenti Animati Conversazionali' (vedi Capitolo 6), realizzato da Giuseppe Clarizio nell'ambito della sua Tesi di Laurea in ICD.

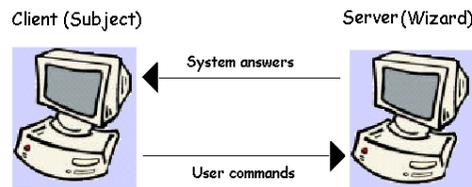


Figura 2.11: L'architettura di un tool per la realizzazione di studi Wizard of Oz

#### Esempio sul Caso di Studio 2.11

Consideriamo il caso in cui, per facilitare il lavoro del medico, si voglia realizzare un sistema di aiuto alla scelta della terapia, in linguaggio naturale. Un sistema di questo genere è complesso da costruire; in particolare, i suggerimenti diagnostici possono essere formulati in modo diverso: ad esempio, formulando i suggerimenti sotto forma di 'ordine' piuttosto che di 'illustrazione di alternative'; oppure fornendo più o meno dettagli sulle ragioni di una scelta.

Per valutare l'alternativa migliore, si può realizzare un simulatore in cui l'utente inserisce i dati relativi ai problemi presentati dal paziente e il mago sceglie il suggerimento diagnostico, selezionandolo all'interno di un insieme di messaggi pre-compilati. I soggetti coinvolti nell'esperimento vengono divisi in diversi gruppi, a ciascuno dei quali vengono forniti i suggerimenti, formulati secondo uno dei metodi che si vogliono mettere a confronto. Un gruppo riceverà (nel nostro esempio) i suggerimenti in forma di ordine, un gruppo in forma di illustrazione di alternative, ecc. Alla fine dell'interazione, viene chiesto al soggetto di compilare un questionario che ha l'obiettivo di raccogliere una valutazione *soggettiva* del sistema ('ti piace', 'useresti questo sistema', 'condividi il suggerimento diagnostico', ecc) oltre a dati *oggettivi* di vario genere, come il ricordo del suggerimento diagnostico.

<sup>2</sup> L'espressione 'Wizard of Oz' (Mago di Oz) è mutuata dalla famosa favola.

**Scheda 2.4:** Questionario Preliminare sulle Caratteristiche dei Soggetti  
(da una *studia di valutazione dell'Università di Reading, 2003*)

=====

**Some questions about you.....**

1) How much would you say you know about healthy eating?

1	2	3	4	5	6
<i>Very little knowledge</i>					<i>Considerable knowledge</i>

2) To what extent do you consider yourself to be a healthy eater?

1	2	3	4	5	6
<i>Not at all</i>					<i>Extremely</i>

3) To what extent do you try to eat a diet that is rich in vitamins and minerals?

1	2	3	4	5	6
<i>Never</i>					<i>Always</i>

4) What do you see as the main benefits (if any) and costs (if any) of eating a diet that is rich in vitamins and minerals?

Benefits: \_\_\_\_\_

Costs: \_\_\_\_\_

Age \_\_\_ Sex M / F (please circle one)

*End of questionnaire*

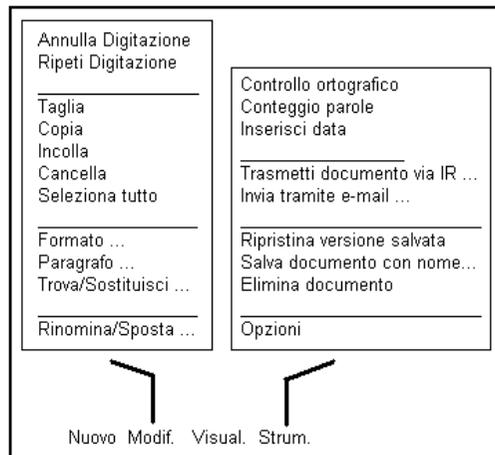
=====



- La *difficoltà di digitare dati* da tastiera e la limitata efficienza dei metodi di riconoscimento dei caratteri scritti a mano suggerisce di limitare al minimo l'uso di questa modalità d'interazione, ricorrendo ove possibile a scelta di item da liste predefinite;
- Chi usa un palmare userà molto probabilmente, in situazioni diverse, anche sistemi tradizionali: è quindi fondamentale che vengano rispettati i criteri di *consistenza esterna* con questi sistemi.

**Esempio 2.12**

Windows CE è la versione di Windows per palmari. La sua consistenza 'esterna' con le versioni più recenti di Windows per pc è particolarmente importante, visto il rapporto main-satellite fra i due sistemi e quindi il continuo spostamento d'uso da un sistema all'altro, da parte della larga maggioranza degli utenti di un palmare. La figura 2.12 riproduce l'organizzazione della finestra principale di Pocket Word (la versione Windows CE di Word). Confrontando questa figura con (ad esempio) l'interfaccia di Word 97, si nota che la toolbar in basso contiene un sottoinsieme delle funzioni della main toolbar di Word 97 e ne rispetta l'ordine, mentre i menu a tendina (ad esempio, di Modifica e di Strumenti) mescolano un sottoinsieme degli elementi nei dei menu a tendina corrispondenti in Word 97 con altre funzioni che, in questo software, sono individuabili come subtask di altri task principali.



**Figura 2.12:** Parte dell'interfaccia di Pocket-Word

**Esercizio 2.9**

Quali delle differenze fra l'interfaccia di pocket PC mostrata in figura 2.12 e quella di una versione di Word che conosci ti sembrano fonti d'inconsistenza particolarmente dannose, e perché?

In definitiva: le applicazioni realizzate su un palmare hanno, necessariamente, una complessità visiva molto minore di quelle realizzate su un pc. D'altra parte, la loro complessità funzionale è molto maggiore e quindi richiede la messa in opera di metodi particolari.

**Esercizio 2.10**

Rifletti su un text editor per pc e per palmare. Quali elementi di usabilità ti sembrano importanti nei due casi? Come ritieni che la scelta di questi elementi influenzi il progetto dell'interazione?

**Scheda 2.5: Il Comunicatore Philips per bambini (Preece et al, 2002)**

Utenti: ragazzi fra i 7 e i 12 anni (soprattutto ragazze);

Funzioni: scambio di messaggi e accessori.

Utenti e esigenze molto simili, quindi non necessaria l'adattività.

Metodo di progettazione: participatory design e iterative prototyping.

Criteri di usabilità: 'being enjoyable, entertaining and fun', piuttosto che l'efficienza 'incoraggiare la creatività' nei ragazzi

Hardware: un palmare ad hoc con schermo touch-sensitive input vocale e con la penna comunicazione a infrarossi output scritto e vocale

Metafora d'interazione:

un mondo in cui gli utenti possano muoversi liberamente, prendendo oggetti e attivando applicazioni.

Funzioni:

calendario  
sveglia  
album di fotografie  
oroscopo  
scrittura di lettere  
composizione di musica  
disegno  
invio di messaggi a strumenti simili

Metodo:

low-fidelity prototyping e prototipo definitivo mediante confronto ripetuto con gli utenti.