

# CAPÍTULO 3

---

## PARADIGMAS DA COMUNICAÇÃO HUMANO- COMPUTADOR E O DESIGN DE INTERFACES

*Engineering, medicine, business,  
architecture, and painting are concerned not  
with how things are but how things might be  
– in short, with design*  
Simon (1982, pxi)



## INTRODUÇÃO

O processo de design em IHC tem sido naturalmente centrado no usuário e têm incorporado questões relativas a modelos cognitivos do processamento humano. Neste capítulo discutiremos a natureza da interação com computadores, com base nos fundamentos apresentados no capítulo anterior. Discutiremos o processo de design de várias perspectivas: da Engenharia Cognitiva à Engenharia Semiótica. O capítulo também explorará a influência de aspectos sociais e organizacionais do contexto do usuário e de sua tarefa no processo de design.

Existe uma grande influência de métodos da engenharia, em particular de Engenharia de Software em IHC, através do design e desenvolvimento de software. Por outro lado, teorias de design originais de contextos diversos, em particular o design industrial, arquitetônico e gráfico também têm influenciado a maneira como design de interfaces tem sido feito recentemente. Os modelos do processo de design em IHC envolvem desde uma discussão crítica dos ciclos de vida clássicos para o desenvolvimento de software, originais da Engenharia de Software, até modelos mais específicos do ciclo de design, como, por exemplo, o Modelo Estrela de Hix e Hartson (1993). Esses modelos apresentados também em Baecker *et al.* (1995), em geral envolvem além do desenvolvimento do sistema propriamente dito, um contexto mais amplo que inclui também questões de natureza social e de organização do trabalho.

Iniciamos examinando as bases do design em IHC, através da apresentação e discussão do paradigma que tem fundamentado toda a área. Serão apresentados e discutidos os princípios do Design Centrado no Usuário e a Teoria da Ação, com base no trabalho seminal de Norman e Draper (1986). Passaremos, então, a algumas extensões e variações desse modelo. Será discutido também o Design Participativo (Schuler e Namioka, 1993) como ilustração das abordagens que tentam assegurar a participação efetiva do usuário e seu contexto social no design. Essa abordagem, onde o usuário é parte ativa no processo de design e não simplesmente parte do processo final de avaliação do software, tem sido considerada principalmente para o design de sistemas colaborativos e de trabalho em grupo.

Como um dos suportes ao processo de design, será discutida a Engenharia de Usabilidade (Nielsen, 1993), que apresenta uma abordagem ao design de sistemas onde níveis de usabilidade são especificados *a priori*, e o sistema é construído com objetivos de alcançar essas medidas, conhecidas como métricas. Os métodos da Engenharia de Usabilidade têm sido bem aceitos pela indústria de software por proverem uma sistemática para teste de um produto durante desenvolvimento. Também popular na indústria de software é o Design baseado em *guidelines*, que parte do princípio de que um bom design resulta parcialmente de conhecimento e experiência do designer e da maneira como ele aplica esse conhecimento. O uso de *guidelines*, princípios e regras como suporte ao design será também discutido, segundo suas bases da teoria cognitiva.

Finalmente o capítulo apontará para propostas mais recentes de linguagens de design para interfaces e teorias fundamentadas no paradigma da comunicação, como é o caso da Semiótica Computacional (Andersen *et al.*, 1993, Andersen, 1997) e da Engenharia Semiótica (Souza, 1993). Estas últimas introduzem novas concepções para interface, e novos formalismos para análise, design e avaliação de software que instrumentam o designer segundo bases semióticas.

## ENGENHARIA COGNITIVA

***Method helps intuition if it is not transformed into dictatorship. Intuition augments method if it does not instill anarchy. In every moment of our semiotic existence, method and intuition complement each other.***

(Nadin, 1988, p.98)

Como vimos nos capítulos anteriores, nossa mente tenta fazer sentido das coisas que vemos ao nosso redor. Os objetos que fazem parte de nosso dia-a-dia são ótimos exemplos para pensarmos nesse processo. Frequentemente encontramos novos objetos (ou novas apresentações para antigos objetos) no dia-a-dia, enquanto estamos fazendo alguma outra coisa, realizando alguma tarefa. Somos distraídos da tarefa que estamos realizando por alguma coisa que deveria ser simples, e não causar esforço. A maneira como lidamos com essas situações é explicada, em parte pela psicologia dos fatores humanos, da cognição e do pensamento como visto no Capítulo 2. A informação expressa na aparência dos objetos, conforme já discutido anteriormente, de certa forma dirige nosso processo de interpretação e operação sobre esse objeto. A facilidade ou dificuldade com que operamos no mundo dos objetos é, portanto, devida à habilidade do designer em tornar clara a operação sobre o objeto, projetando uma boa imagem da operação e considerando outros elementos do universo de conhecimento do usuário. A Semiótica, conforme veremos mais adiante ainda neste capítulo, também explica essa relação entre o objeto, seu representante e o processo de interpretação, no plano dos signos que encontramos ao nosso redor.

Engenharia Cognitiva é um termo cunhado por Norman e Draper (1986), um tipo de “Ciência Cognitiva Aplicada”, conforme ele próprio categoriza, que tenta aplicar o que é conhecido da ciência ao design e construção de máquinas. Os conceitos da Engenharia Cognitiva formam as bases do paradigma dominante atualmente na área de IHC. Entre suas metas principais estão: entender os princípios fundamentais da ação humana que são relevantes à engenharia do design, indo além dos aspectos ergonômicos; criar sistemas “agradáveis de usar”, que possibilitem ao usuário um “engajamento prazeroso” na terminologia de Laurel (1990), indo além dos aspectos de facilidade de uso.

A Engenharia Cognitiva considera dois lados na interface: a do próprio sistema e a do usuário. A realização de tarefas complexas por não expertos é considerada uma atividade de resolução de problemas cujo processo é facilitado quando a pessoa possui um bom modelo conceitual do sistema físico.

A complexidade da tarefa é devida às naturezas diferentes das variáveis envolvidas. A pessoa possui metas e intenções – variáveis psicológicas – que se relacionam diretamente às necessidades da pessoa. A tarefa deve ser realizada em um sistema físico, com mecanismos físicos a serem manipulados, resultando em mudanças nas variáveis físicas e no estado do sistema. A pessoa interpreta as variáveis físicas em termos relevantes às suas metas psicológicas e traduz as intenções psicológicas em ações físicas sobre os mecanismos do sistema. Isso significa que há um estágio de interpretação que relaciona variáveis físicas e psicológicas, assim como funções que relacionam a manipulação das variáveis físicas às mudanças no estado físico do sistema.

Mesmo tarefas simples envolvem um número grande de aspectos a considerar. Consideremos, por exemplo, a tarefa de um marinheiro novato tentando manobrar um barco à leme. Há um único mecanismo de controle (o leme) que afeta uma única variável (a direção do barco). A complexidade é em parte devida à discrepância entre o modelo mental do marinheiro e o modelo conceitual do sistema. O mapeamento entre o movimento do leme e a direção do barco em geral é oposta ao que os novatos esperam.

Uma intenção é criada para satisfazer uma meta. No caso do barco à leme, a meta é alcançar um certo alvo. A diferença entre a meta desejada e o estado atual do sistema dá origem a uma intenção – em termos psicológicos – que é traduzida em uma sequência de ações: a especificação de que ações físicas serão realizadas nos mecanismos do sistema. Portanto, o caminho da intenção para a especificação de ações requer a consideração do mapeamento entre os mecanismos físicos e o estado do sistema e entre o estado do sistema e a interpretação psicológica resultante.

Norman e Draper (1986) propõe “uma teoria da ação” para entender “como as pessoas fazem as coisas”, distinguindo entre diferentes estágios de atividades. Na Teoria da Ação são diferenciados sete estágios da atividade do usuário, conforme ilustra a Figura 3.1. Muitos sistemas computacionais podem ser categorizados por quão bem suportam os diferentes estágios. Imagine, por exemplo, uma pessoa interagindo com um sistema computacional. As metas da pessoa são expressas em termos relevantes à pessoa (psicológicos). Os mecanismos e estados do sistema são expressos em termos relativos a ele (físicos). A discrepância entre variáveis físicas e psicológicas cria os pontos a serem considerados no design, análise e uso de sistemas, que Norman chamou de “golfos da execução e da avaliação”.

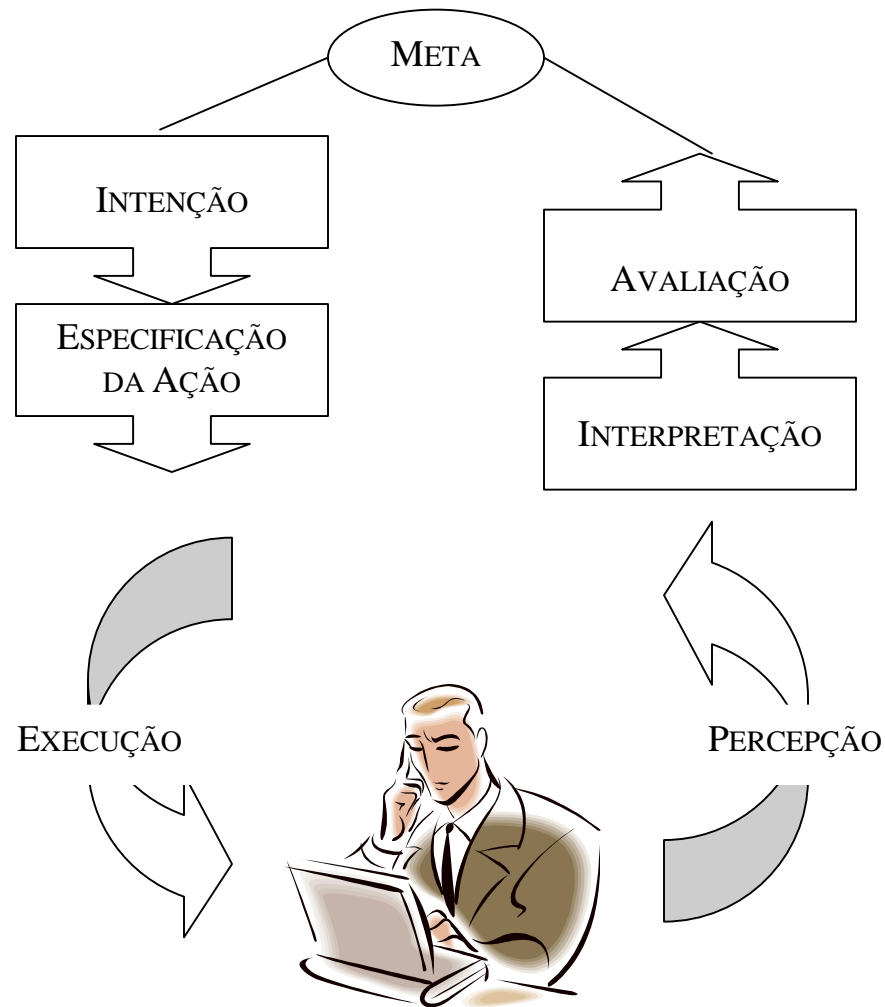


FIGURA 3.1 - OS SETE ESTÁGIOS DA TEORIA DA AÇÃO DE NORMAN

O **Golfo da Execução** envolve as atividades de formação da intenção, especificação da sequência de ações – o que não é trivial – e execução da ação através do contato com mecanismos de entrada da interface. O **Golfo da Avaliação** requer comparar a interpretação do estado do sistema com as metas e intenções originais. Começa com a apresentação de saída da interface, a partir da qual o usuário passa pela atividade de processamento perceptual da saída, interpretação e avaliação (comparação da interpretação do estado do sistema com a intenção e metas originais).

Possibilitar que o usuário atravessasse os golfos significa construir uma interface que se ajuste às necessidades do usuário, de forma que possa ser prontamente interpretada e manipulada. O usuário também pode, ele próprio, ao custo de seus esforços,

atravessar os golfos, criando planos, seqüências de ações e interpretações que movem a descrição de metas e intenções para mais próximo da descrição requerida pelo sistema físico (e não pela tarefa original).

Menus são exemplos de suporte aos estágios de execução e especificação de ações. A presença visual pode ajudar em vários estágios da atividade: como suporte à geração de intenções – lembrando o usuário do que é possível; como suporte à seleção de ação: itens visíveis atuam como tradução direta para ações possíveis; como suporte à execução se associado a dispositivo de apontamento; como suporte à avaliação: lembrando visualmente o que foi feito; como suporte à interpretação através do uso de determinadas representações.

Resumindo, a Engenharia Cognitiva conceitua interface pelos seus “dois lados”: o do sistema e o do ser humano. Estágios de execução e percepção (humanos) mediam entre representações físicas (do sistema) e psicológicas (do ser humano). Mecanismos de entrada/saída (do sistema) mediam entre representações psicológicas e físicas. Mudamos a interface, pelo lado do sistema, através de design apropriado. Muda-se a interface pelo lado humano, através de aprendizado e experiência. Na situação ideal, nenhum esforço psicológico deveria ser requerido para se atravessar os golfos.

Design de interface no paradigma da Engenharia Cognitiva, portanto, relaciona três tipos de conhecimento: de design, programação e tecnologia; de pessoas, princípios do funcionamento mental, comunicação e interação e conhecimento da tarefa. Somente o módulo da interface deve estar em comunicação com o usuário: do ponto de vista do usuário a interface “é” o sistema, conforme veremos na próxima seção.

## MANIPULAÇÃO DIRETA

*Beware the Turing tar-pit in which everything is possible but nothing of interest is easy* (Perlis, A., apud Hutchins *et al.* 1986)

*O que exatamente é a manipulação direta?*

Imagine-se dirigindo um carro em que, em vez de direção, pedais e câmbio tem apenas um teclado...e R20:E:A35 seria usado para “reduza para 20km/h, vire para a esquerda, acelere até 35 km/h”

Shneiderman (1983) foi quem usou o termo pela primeira vez para se referir a uma classe emergente de sistemas bastante atraentes na década de oitenta, como as primeiras planilhas eletrônicas, editores de texto, sistemas CAD, videogames, etc. Esses sistemas possuíam interfaces gráficas que permitiam operá-los “diretamente”

usando ações manuais em vez de instruções fornecidas via teclado. Tais sistemas mudaram o paradigma da interação humano-computador, do “diálogo” baseado em linguagem de comando para a “manipulação” baseada na linguagem visual (Frohlich, 1997). Em vez de um meio computacional abstrato, toda programação é feita graficamente, em uma forma que tenta casar com a maneira como pensamos no problema (Hutchins *et al.*, 1986).

Do mundo que se “comanda” passou-se para o mundo com o qual se “interage”. O primeiro marco em interface de manipulação direta é o *Sketchpad*, um programa para design gráfico criado por Sutherland em 1963. Seu trabalho é um marco não apenas pela prioridade histórica, mas por antecipar a concepção da saída na tela como “folha de papel”, o uso de dispositivos de apontamento e a importância de mostrar abstrações graficamente.

Nas interfaces de manipulação direta não há operações escondidas, sintaxe ou nomes de comandos para aprender. O único conhecimento requerido é no próprio domínio da tarefa. Excelentes exemplos de tais interfaces na época eram os editores WYSWYG (*what you see is what you get*). A diretividade em interfaces, entretanto, apresenta-se em vários níveis dentro de um *continuum*. Tomemos como exemplo um editor gráfico numa tarefa de criar ilustrações; a operação de mover um círculo mostra os graus de indireção que podem estar presentes na interface: A) o usuário aponta na tela o círculo e o move levando-o para a posição desejada com o dedo. B) a introdução do mouse coloca mais um grau de indireção, uma vez que para mover o círculo é necessário mover o mouse. C) com as setas de direção acrescenta-se mais um nível de indireção, uma vez que não há equivalência de movimentos. D) o uso de um comando para mover o círculo acrescenta mais um nível uma vez que a sintaxe e a semântica determinam o que acontece.

A “ilusão” da manipulação direta foi sumarizada por Shneiderman (1998) em três princípios de design:

1. Representação contínua do objeto de interesse;
2. Ações físicas (cliques, arraste, etc.) em vez de sintaxe complexa;
3. Operações incrementais reversíveis, cujo impacto no objeto de interesse é imediatamente visível.

Esses princípios são baseados na suposição de que a manipulação direta resulta em menor comprometimento de recursos cognitivos. Hutchins *et al* (1986) discutem as bases subjacentes para sistemas de manipulação direta.

Retomando a mudança paradigmática na natureza da interação humano-computador, na metáfora da conversação, a linguagem da interface é um “meio” no qual usuário e sistema têm uma conversação sobre um mundo assumido, mas não explicitamente representado. O usuário está em contato com estruturas lingüísticas que podem ser interpretadas como se referindo aos objetos de interesse.



Na metáfora do mundo-modelo, termo cunhado por Hutchins *et al* (1986) para representar o paradigma da manipulação direta, a interface é um mundo onde o usuário age. Esse mundo muda de estado em resposta às ações do usuário. Em vez de descrever as ações de interesse, o usuário realiza as ações.

Diretividade não é uma propriedade da interface isoladamente, entretanto. Dois aspectos são usados para se medir a diretividade: distância e engajamento. Distância refere-se à distância entre o pensamento de alguém e os requisitos físicos do sistema em uso; relação entre a tarefa que o usuário tem em mente e a maneira como a tarefa pode ser realizada através da interface. Engajamento é o sentimento de que se está manipulando diretamente os objetos de interesse.

O sentimento de diretividade é inversamente proporcional à quantidade de esforço cognitivo requerido para manipular e avaliar um sistema. Esse esforço cognitivo é resultado direto dos Golfos de Execução e Avaliação, conforme discutido na última seção. Assim, quanto mais as interfaces ajudarem a aproximar os Golfos, menor será o esforço cognitivo requerido e mais direto será o sentimento de interação resultante.

Independentemente da metáfora utilizada para a interface, seja ela a “conversação” ou o “mundo-modelo”, há uma “linguagem” da interface. E toda expressão na linguagem da interface tem um significado e uma forma. Essa independência entre forma e significado, para efeitos de análise, permite descrever duas propriedades da linguagem da interface, que Hutchins *et al* chamaram de Distância Semântica e Distância Articulatória.

**Distância semântica** (DS) reflete a relação entre as intenções do usuário e o significado na linguagem da interface. **Distância articulatória** (DA) reflete a relação entre a forma física de uma expressão na linguagem de interação e seu significado.

Duas perguntas básicas resumem a medida de diretividade semântica em uma interface: *É possível dizer o que se quer dizer nessa linguagem? As coisas de interesse podem ser ditas de forma concisa?*

Com relação ao Modelo da Teoria da Ação, a diretividade semântica pode ser medida no golfo de execução observando-se quanto da estrutura requerida é fornecida pelo sistema e quanto pelo usuário. Quanto mais estrutura o usuário tiver que prover, maior será a distância que ele terá que transpor. No golfo de avaliação a diretividade semântica pode ser observada pela quantidade de estruturas requeridas para o usuário determinar se uma dada meta foi alcançada ou não.

Como, então, reduzir a distância semântica? Pelo lado do sistema o designer pode construir linguagens especializadas de mais alta ordem, fazendo a semântica da entrada/saída do sistema “casar” com os modelos mentais de entrada/saída do usuário. Um exemplo simples dessa redução no golfo de avaliação é fazer a saída do

sistema mostrar os conceitos diretamente, em lugar de deixar o usuário computá-los mentalmente. Algumas representações gráficas para tabelas numéricas e sistemas *wysiwyg* para editores de texto são exemplos inquestionáveis de redução da distância semântica. Essa estratégia representa uma mediação entre intenções e expressão na linguagem. Portanto, a linguagem da interface deve prover uma maneira poderosa e produtiva para o usuário pensar sobre o domínio. Essa estratégia de redução da distância semântica, do ponto de vista de design, pode introduzir um conflito entre generalidade do sistema e orientação a um domínio específico.

O usuário também reduz a distância semântica, independentemente do sistema, construindo estruturas mentais para atravessar os golfos, aprendendo a pensar na mesma linguagem requerida pelo sistema. Assim, as metas iniciais e intenções do usuário são formadas na mesma linguagem requerida pelo sistema. Isso explica o “sentimento de diretividade” que o uso freqüente de uma interface – que não é de manipulação direta – pode resultar. Um usuário Unix, por exemplo, pode “sentir” a ação do “rm arq” como mais direta do que o “arrastar do arq para a lixeira”, uma vez que já tem a atividade de planejamento substituída pela atividade de recuperação da memória, nas ações requeridas para a tarefa. Como designers é essencial distinguir o comportamento automatizado – que pode contribuir para o sentimento de diretividade – da diretividade semântica propriamente dita. Esse fenômeno é explicado pela teoria do determinismo lingüístico de Worf, segundo a qual a maneira como pensamos em algo é moldada pelo vocabulário que temos para pensar (sobre).

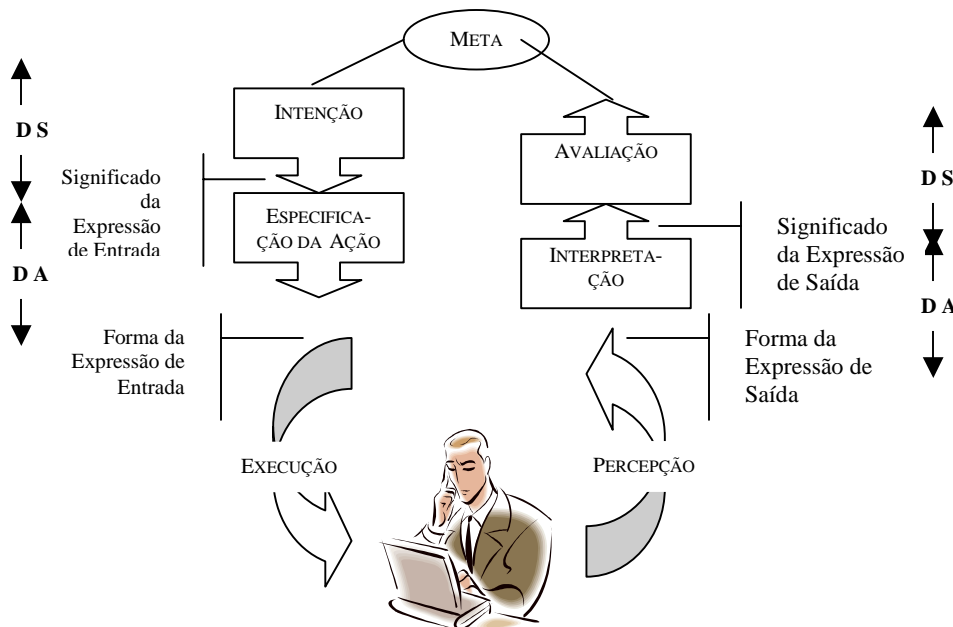
Deve-se notar que observar a interface isoladamente, em geral, não é suficiente para se determinar a distância semântica de um sistema. Hutchins *et al.* (1986) citam o exemplo dos instrumentos musicais: o teclado do piano é mais direto semanticamente que as cordas do violino, para a tarefa de produzir notas. Entretanto, o violino é melhor para controlar características mais sutis do som. A diretividade semântica é uma medida da distância entre a meta e intenção do usuário e o significado da expressão disponível na interface. Uma análise da natureza da tarefa sendo realizada é essencial para se determinar a diretividade semântica da interface.

Todo item de vocabulário em toda linguagem tem uma estrutura física. Por exemplo, a palavra na linguagem natural tem uma estrutura fonética e uma estrutura tipográfica. Assim também os itens que constituem a linguagem da interface têm uma estrutura física. Na entrada são exemplos: seqüências de caracteres que são teclados em uma interface de linguagem de comandos, ou movimentos e cliques no mouse ou, ainda, uma cadeia fonética. Na saída, cadeias de caracteres, mudanças em ícones, grafos, diagramas, animação, sinais audíveis, etc. A diretividade articulatória é uma medida da distância entre a forma física da expressão dos elementos da interface e seu significado.

Há maneiras de se criar linguagens de modo que relações entre os itens de vocabulário e seus significados não sejam arbitrários. Um bom exemplo disso na linguagem natural são as onomatopéias. Na linguagem da interface, a diretividade articulatória pode ser aumentada, por exemplo, permitindo-se ao usuário a especificação de uma ação imitando-a, ou permitindo que ele use conhecimento anterior para criar a relação forma-significado. O mouse é um bom exemplo de periférico que provê diretividade articulatória para tarefas representadas espacialmente. A intenção de mover o cursor na tela, e a ação sobre o mouse são movimentos similares.

Linguagens icônicas são exemplos de representações em que a forma da expressão é relacionada a seu significado. Conforme veremos na seção dedicada à Semiótica, entretanto, o abuso de utilização de símbolos gráficos em interfaces, que nem sempre correspondem ao conceito de ícone, não garantem a diretividade articulatória completa.

A Figura 3.2, adaptada de Hutchins *et al.* (1986) resume graficamente os conceitos tratados nesta seção.



**FIGURA 3.2** - DISTÂNCIAS SEMÂNTICAS E ARTICULATÓRIAS NOS GOLFOS DE EXECUÇÃO E AVALIAÇÃO

Resumidamente, para possibilitar o sentimento de engajamento direto na interface o design deve remover a percepção do computador como um intermediário,

possibilitando execução e avaliação diretas, representação contínua do estado do sistema e linguagens de entrada/saída inter-referenciais – a expressão de entrada deve incorporar ou usar a expressão de saída anterior, para criar a “ilusão” de manipulação direta dos objetos. A redução na carga cognitiva para manter mentalmente informação relevante sobre o estado do sistema e a forma de interação contribuem para o sentimento de engajamento.

Shneiderman (1983) sintetiza os benefícios da manipulação direta para a usabilidade de sistemas computacionais, nos seguintes aspectos:

- Facilidade de aprendizado do sistema.  
Novatos podem aprender rapidamente a funcionalidade básica;
- Facilidade de memorização.  
Usuários frequentes podem reter mais facilmente conceitos operacionais;
- Performance melhorada do experto no domínio da tarefa;
- Redução de mensagens de erro;
- Aumento no controle do usuário.  
Usuários têm a ansiedade reduzida, porque o sistema é compreensível e porque as ações são facilmente reversíveis.

Ao mesmo tempo, alguns elementos são críticos no design de interfaces de manipulação direta, em especial a qualidade da representação gráfica selecionada. A representação deve ter significado preciso para o usuário, o que veremos em maior detalhe na seção sobre Semiótica.

## MODELOS DO DESIGN DE SOFTWARE

*Developing user-oriented systems requires living in a sea of changes*

*Nobody can get it right the first time*

observações sobre design de sistemas, em Gould *et al* (1997)

Design de software, que costuma ser traduzido em nossa língua por “projeto de software”, tenta relacionar a forma e função de um sistema de software à estrutura do processo que produz esse sistema. A tradição herdada de princípios da engenharia apresenta uma abordagem à problemática da crise de software da década de 60, que se baseia na crença de que um processo rigoroso de transformação de requisitos em sistemas é a chave para um design confiável (Denning e Dargan, 1996).

O processo de design na Engenharia de Software (ES) parte de três pressupostos básicos: o resultado do design é um produto, seja ele um artefato, máquina ou

sistema; o produto é derivado de especificações fornecidas pelo cliente – em princípio com conhecimento suficiente e poder de computação essa especificação pode ser mecanizada. Finalmente, uma vez que o cliente e o designer concordaram com as especificações, há pouca necessidade de contato entre eles até a entrega do produto.

O “modelo cascata” (Boehm, 1995, p. 282) caracteriza bem a visão tradicional da ES para o desenvolvimento de software, como um conjunto de processos e representações produzidas de maneira linear (Figura 3.3).



FIGURA 3.3 - O MODELO CASCATA

O principal problema com o modelo cascata é que é impossível entender completamente e expressar os requisitos do usuário antes que algum design tenha sido feito. Além disso, as possibilidades de mudanças no software a partir da etapa

de manutenção são mínimas, em função dos comprometimentos e custos envolvidos ao longo da cadeia.

Em resposta aos problemas do modelo cascata, Boehm (1995, p.284) propõe o “modelo espiral” (Figura 3.4).

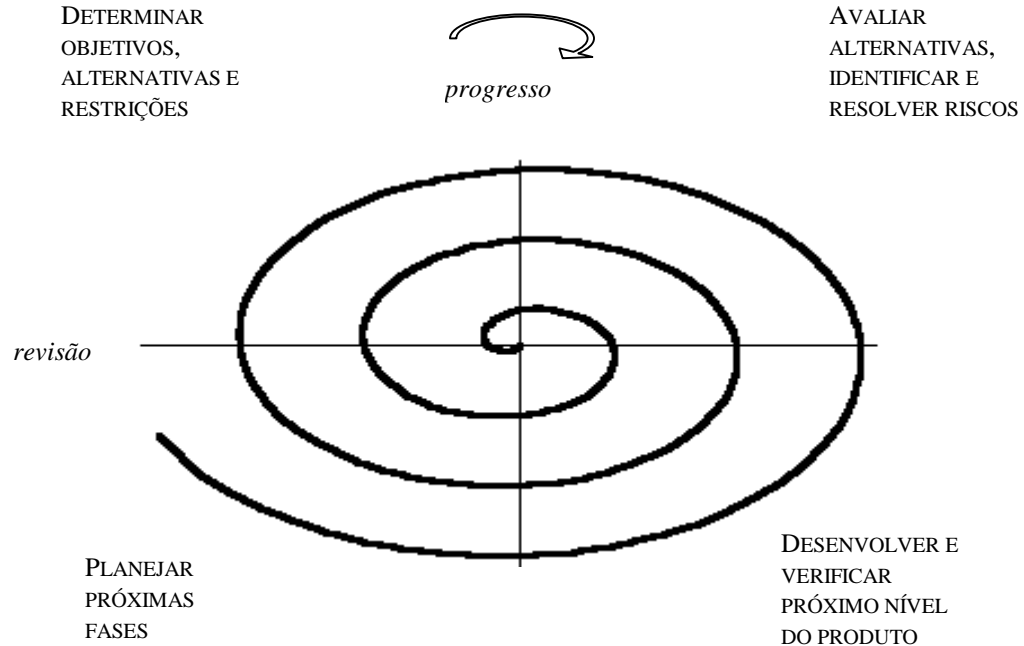


FIGURA 3.4 - O MODELO ESPIRAL

Embora ainda use os mesmos processos do modelo anterior – análise de requisitos, design e implementação – e seja orientado ao produto, o modelo espiral já mostra que várias interações são necessárias e introduz a idéia de prototipagem para maior entendimento dos requisitos. Mas o que leva a um bom design?

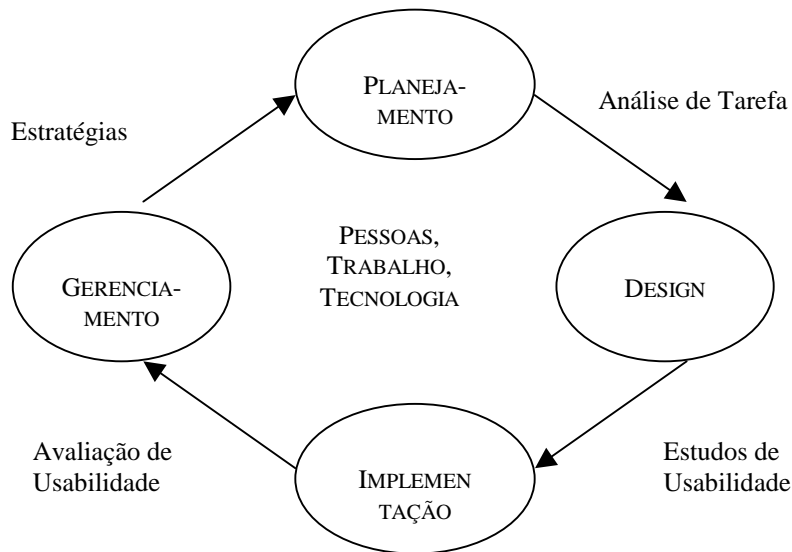
Peter Denning (quando presidente da ACM e editor da *Communications*) e Pâmela Dargan (software designer) entrevistaram designers de bons sistemas e observaram que suas respostas estavam longe da cultura da ES convencional. O processo de design em engenharia oferece pouca relação entre ações do designer e as necessidades dos usuários, produzindo uma “cegueira” no domínio de ações no qual os usuários vivem e trabalham (Denning e Dargan, 1996).

Como reação à problemática do design centrado no produto, surgiu na década de 80 a escola do design centrado no humano (DCH), que se fundamenta no entendimento

do domínio de trabalho no qual as pessoas estão engajadas e no qual interagem com computadores. Como pressupostos do DCH, Denning e Dargan (1996) destacam: o resultado de um bom design é a satisfação do cliente; o processo de design envolve uma colaboração entre designers e clientes – o design evolui e se adapta aos seus interesses (que também mudam) e esse processo é que produz uma especificação como subproduto. Fundamentalmente o cliente e o designer estão em constante comunicação durante todo o processo.

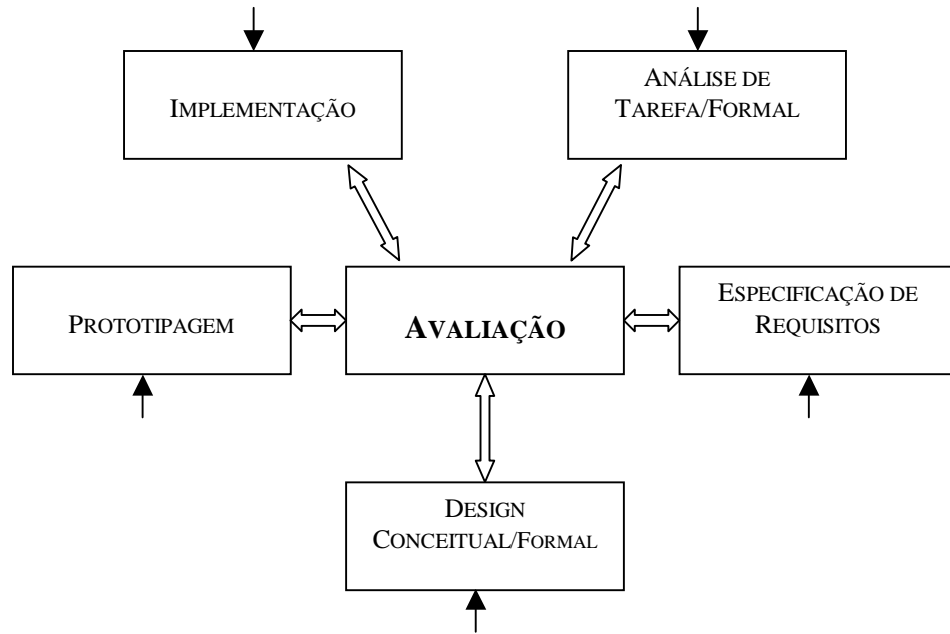
O DCH tem como objetivo produzir sistemas fáceis de aprender e usar, seguros e efetivos em facilitar as atividades do usuário. Reconhece a importância de testes frequentes com o usuário usando representações informais e prototipagem. O aspecto central do DCH é o envolvimento efetivo de usuários ao longo do processo de design, não apenas para “comentar” decisões do designer.

Vários modelos para o processo de design têm sido apresentados na literatura de IHC. Entre eles destaca-se o modelo de Eason (apud Preece *et al.* 1994, p.372), ilustrado na Figura 3.5. Nesse modelo, o processo de design é representado como um processo de natureza cíclica centrado em pessoas, trabalho e tecnologia e é ordenado e não *ad-hoc*.



**FIGURA 3.5** - O MODELO DE EASON

O “modelo estrela” (Hix e Hartson, 1993), derivado de extensa análise da prática corrente de design à época, é bastante popular entre a comunidade de IHC. Esse modelo, ilustrado pela Figura 3.6, apresenta uma abordagem ao desenvolvimento como “ondas alternantes”. As atividades são similares às do modelo cascata, mas a avaliação é central e o início do processo pode acontecer em qualquer uma das demais atividades.



**FIGURA 3.6** - O MODELO ESTRELA

As abordagens da ES e da IHC possuem forças e fraquezas complementares; juntas formam uma nova disciplina: a arquitetura de software. Shneiderman (1998) propõe um modelo para design baseado metaforicamente em 3 “pilares” (Figura 3.7). No início do processo o designer deve gerar (ou requerer) um conjunto de *guidelines* (princípios e regras de design, conforme veremos mais adiante neste capítulo). O segundo pilar é composto de ferramentas para prototipagem (HyperCard, Visual Basic, Borland Delphi, Visix Galaxy, Sun Java, etc.). O terceiro pilar é dedicado a testes de usabilidade – avaliação por expertos e testes com usuários.



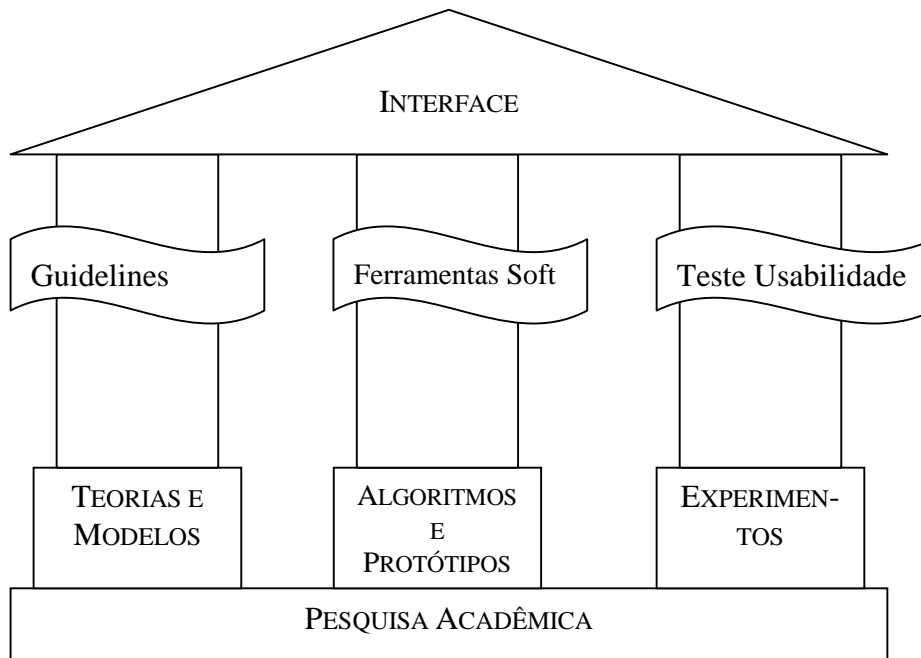


FIGURA 3.7 - O MODELO DE SHNEIDERMAN

Parece já haver um consenso de que qualquer metodologia de DCH deve mesclar-se à metodologias da ES. O modelo LUCID (Logic User-Centered Interface Design), antigo QUE (Quality Usability Engineering) (Kreitzberg, apud Shneiderman, 1998) representa esse esforço. O modelo é representado por uma seqüência de 6 fases, listadas a seguir:

1. Desenvolver o conceito do produto;
2. Realizar pesquisa e análise das necessidades – usando construção de cenários, design participativo, fluxo e seqüência de tarefas;
3. Criar conceitos e protótipos de telas – usando *guidelines*, guias de estilo e metáforas para o design;
4. Design iterativo e refinamento – expandindo o protótipo para sistema completo; inclui a avaliação por expertos e testes de usabilidade;
5. Implementação do software;
6. Suporte;

Vários aspectos influenciam e devem ser considerados na escolha do modelo de design em IHC; entre eles o tipo do sistema a ser desenvolvido, em termos do tamanho, complexidade e propósito. Considerar também se está sendo tratado o design de sistema completamente novo ou de re-design de sistema já existente.

No primeiro caso, há o problema de selecionar entre um grande conjunto de opções e diferentes implicações no design. No segundo caso, a liberdade do designer é severamente restringida por decisões anteriores de design original, linha do produto, etc. Considerar, ainda, as restrições inerentes ao sistema, suas condições de contorno, sistemas críticos em relação à segurança, por exemplo.

Das respostas de uma entrevista feita com designers bem sucedidos à questão de “o que leva a um bom design”, Denning e Dargan (1996, p.113) sintetizam as seguintes sugestões:

- Escolha um domínio no qual muitas pessoas estão envolvidas;
- Estude a natureza das ações dessas pessoas naquele domínio, especialmente em ações repetitivas; o que eles reclamam mais? Que ações gostariam de realizar?
- Defina software que imite padrões de ação incluindo funções que não poderiam ser feitas manualmente;
- Crie protótipos o mais cedo possível e observe como as pessoas reagem, o que “quebra” a experiência delas;
- Mantenha comunicação com eles.

## ENGENHARIA DE USABILIDADE

*It's not broken; that's how it's supposed to work;*

*We didn't anticipate THIS;*

*We are surprised that...*

*The help system will take care of this;*

*We'll take care of it in the NEXT release...*

Comentários de designers de sistemas, em Gould *et al* (1997)

Engenharia de Usabilidade é o termo que se usa para definir o processo de design de sistemas computacionais que objetivam a facilidade de aprendizado, de uso, e que sejam agradáveis para as pessoas.

O processo de design para usabilidade foi inicialmente uma recomendação de vários pesquisadores independentes (Gould e Lewis, 1985; Nielsen, 1992) e grupos de pesquisa na DEC e IBM em Engenharia de Usabilidade que, já na década de 80, constataram que confiar na experiência do designer e em padrões, *guidelines* ou em várias filosofias de design racionais e analíticas não era suficiente para chegar a bons sistemas de computador. A Engenharia de Usabilidade propõe a aplicação de métodos empíricos ao design de sistemas baseados no computador.

As fases do processo surgiram como consenso desses grupos iniciais e estenderam o ciclo tradicional de desenvolvimento que começava com a definição do produto e terminava com sua entrega. O processo de design para usabilidade possui 4 fases: pré-design, design inicial, desenvolvimento iterativo e pós-design.

A fase de pré-design é caracterizada pela busca de informação e conceituação sobre o usuário e seu contexto de trabalho e sobre sistemas relacionados, padrões de interface, *guidelines*, ferramentas de desenvolvimento, etc. A fase do design inicial é constituída da especificação inicial da interface. A próxima fase é a do desenvolvimento iterativo alimentado por feedback de testes até que os objetivos tenham sido alcançados. Finalmente há a fase do pós-design com a instalação do sistema no local de trabalho do usuário e acompanhamento com medidas de reação e aceitação do sistema pelo usuário final.

Os estágios do design para usabilidade ilustram os quatro princípios básicos que fundamentam esse processo: foco no usuário mais cedo, medição empírica, design iterativo e design integrado de todos os aspectos de usabilidade do sistema (Gould *et al.*, 1997).

O estágio de pré-design envolve conhecer os usuários: características individuais (nível escolar, idade, experiência no trabalho, no uso de computadores, sua tarefa atual, como lidam com emergências, etc.) e definir o que eles estarão fazendo com o sistema. Decisões de design subsequentes – organização do sistema, funções requeridas, interface – devem refletir essas respostas. Também nesse estágio cabe uma análise comparativa de produtos existentes (competidores) e testes com usuários no uso desses produtos. Ainda na fase de pré-design devem ser estabelecidas as metas de usabilidade para o sistema. As cinco características principais de usabilidade, como visto no Capítulo 1 são: facilidade de aprendizado, eficiência de uso, facilidade de retorno ao uso por usuários casuais, frequência e severidade dos erros dos usuários e satisfação subjetiva do usuário (Nielsen, 1992). Conhecer as metas de usabilidade clarifica o processo de design.

Vários métodos podem ser usados nesse estágio para conseguir o foco cedo e contínuo no usuário: visitas ao local de trabalho do usuário para conhecer a organização do trabalho, observação do usuário em seu trabalho, gravação em fita, do usuário trabalhando, análise de tarefa, design participativo, *think aloud* do usuário, etc.

Os estágios de design inicial e desenvolvimento iterativo têm como premissa básica que não se consegue que o sistema dê certo logo na primeira vez, não importando quão experiente o designer seja. Além disso, não se saberá se o sistema está funcionando até que se comece a testá-lo. Os objetivos desse estágio são, portanto, concretizar em um protótipo o design que segue de princípios de usabilidade e verificar empiricamente o design com usuários reais, para assegurar ter atingido as metas.

Na fase do design inicial é recomendado o uso de métodos participativos, uma vez que, embora os usuários não sejam designers, são muito bons em reagir a design que não os agrada ou não funciona na prática. É recomendado, ainda, o uso de *guidelines* gerais – aplicáveis a qualquer interface, *guidelines* de categoria específica

aplicáveis à classe de sistema sendo desenvolvido e *guidelines* específicas para o produto. Um exemplo de *guideline* geral para interfaces: “falar a língua do usuário”. *Guidelines* e Design Participativo serão discutidos separadamente em próximas seções deste capítulo.

O quarto princípio subjacente ao design para usabilidade – o design coordenado (desenvolvimento paralelo da funcionalidade, da interface, do *help*, do material de treinamento, etc.) já aparece nesta fase de design inicial, buscando consistência entre as diferentes mídias que compõem a interface, não apenas às telas. O uso de padrões chamados *de facto* e *in-house*, aumentam o re-uso de código e facilitam a documentação.

A fase do desenvolvimento iterativo é baseada na prototipagem e testes empíricos a cada iteração do ciclo de desenvolvimento. Avaliação qualitativa é aplicada ao sistema em processo de design para verificação dos aspectos da interface que funcionam e principalmente dos que causam problemas. A avaliação heurística é um dos métodos bastante utilizados nesta fase e será tratado no Capítulo 4. Em interfaces quase terminadas são feitas medições quantitativas para checagem das metas.

Muito importante nesta fase é o design *rationale*, um registro que explicita cada decisão de design. Um formalismo que pode ser utilizado para esse registro é o gIBIS (graphical Issue-Based Information System). O *design rationale*, além de manter a memória do processo de design, ajuda a manter a consistência ao longo de diferentes versões do produto.

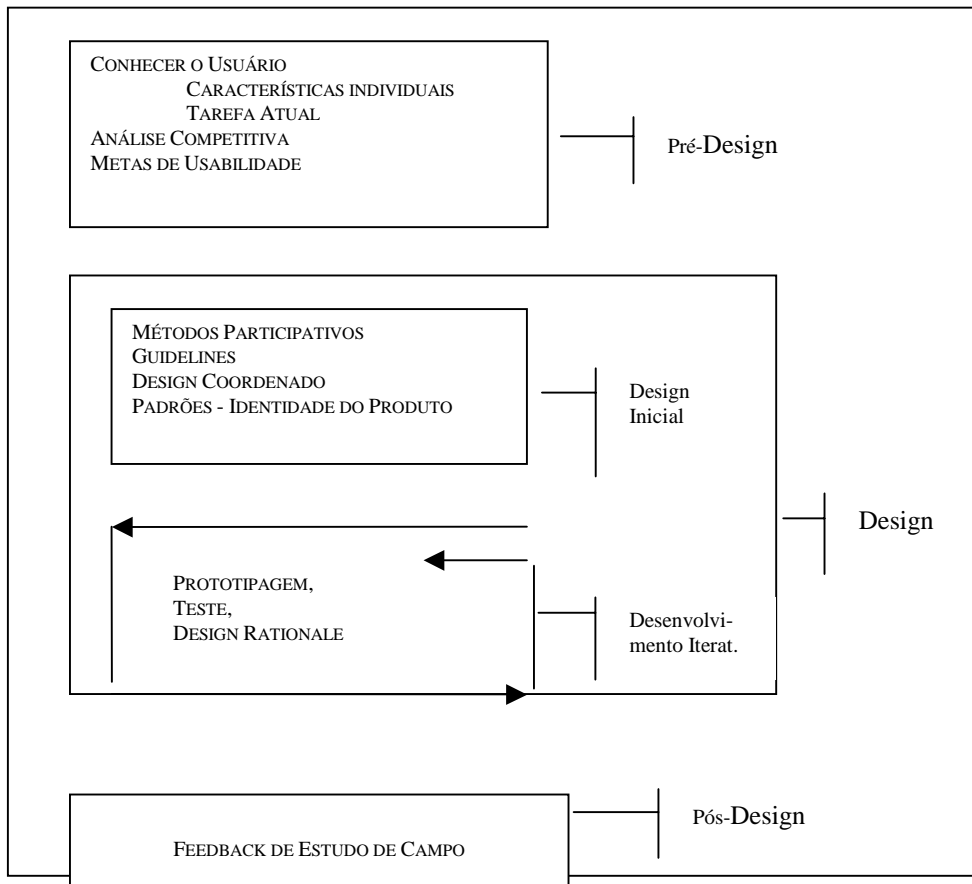
O estágio de pós-design caracteriza-se por conduzir estudos de campo do produto em uso, para obter dados para nova versão e produtos futuros. Esses estudos devem ir além do registro imediato de reclamações buscando avaliar o impacto do produto na qualidade do trabalho do usuário. Os usuários devem ser visitados em seu local de trabalho e devem ser colecionados registros de sessões de uso do sistema para análise.

Com o objetivo de priorizar métodos de usabilidade, Nielsen (1992) mostra os resultados de um questionário respondido por engenheiros de usabilidade a duas questões: quais métodos teriam usado em projetos recentes e o impacto do método na usabilidade do sistema. Os cinco métodos mais usados (de uma lista de 33) foram: visita ao local de trabalho do usuário no pré-design, design iterativo, design participativo, prototipagem (usando ferramentas computacionais) e análise de produtos competidores. Os 6 métodos de maior impacto na usabilidade foram: design iterativo, análise da tarefa do usuário, teste empírico com usuários reais, design participativo, visita ao local de trabalho do usuário e estudo de campo para verificar como o produto é usado depois de sua instalação.

Entre os principais benefícios da Engenharia de Usabilidade citados na literatura está o tempo economizado em não implementar funções que a análise de usabilidade mostrou não serem utilizadas pelos usuários. Estudo de caso documentado e reportado em Nielsen (1992) mostrou também uma economia financeira equivalente ao dobro do que foi investido, com redução de treinamento para determinados produtos. Além da economia de tempo e dinheiro, a adoção de produtos adicionais é quase certa, se são fáceis de usar.

*Design para Usabilidade: porque não?* Ainda há os que acreditam que o processo seria encurtado, que iteração é apenas refinamento, que há falta de ferramentas que facilitem o design iterativo e ainda há quem espere uma “abordagem científica e analítica que leve a uma boa interface na primeira vez...”

A Figura 3.8 ilustra, de forma resumida, o Modelo de Engenharia de Usabilidade.



**FIGURA 3.8 - O MODELO DA ENGENHARIA DE USABILIDADE**

## O USO DE *GUIDELINES* EM DESIGN

### *Falar a língua do usuário*

*Guidelines* são muito populares em design de interfaces por constituírem um *framework* que orienta o designer na tomada de decisões consistentes através dos elementos que constituem o produto. São muito utilizadas por fabricantes que definem, com elas, uma certa identidade à marca. Possuem formas variadas, várias origens – artigos acadêmicos, manuais, estilos associados a marcas, etc. Devem ser entendidas e aplicadas de forma contextualizada.

O uso de *guidelines* não deve ser entendido como “receita de design”, mas sim como um conjunto de princípios norteadores do design. Preece *et al.* (1994) argumenta que o verdadeiro sentido das *guidelines* é o de princípios em alto nível, largamente aplicáveis. A seguir comentaremos algumas.

### *Falar a língua do usuário*

Língua deve ser entendida de forma ampla, no contexto sócio-cultural estabelecido da população de usuários. Envolve conhecer essa população, estar atento para as diferentes necessidades do usuário, promover sua satisfação pessoal e permitir que ampliem e facilitem a realização de suas tarefas. Uma interface que fale a língua do usuário ajuda-o a atravessar o golfo de execução e interagir com o sistema. Só para citar um contra-exemplo bastante simples, num editor de textos para crianças, havia como opção de menu para escolha do tipo de letra, o termo “cursiva”. Certamente essa não é uma palavra do vocabulário infantil – os usuários finais do sistema eram crianças em processo de alfabetização – que conhecem as “letras de forma” e “letras de mão”.

Ao mesmo tempo em que é um princípio e como tal bastante geral, sua aplicação não é simples: envolve reconstruir os sistemas semióticos de uma população que só se conhecerá ao longo do processo de design.

### *Reduzir a carga cognitiva*

Isso significa que o usuário não deve ter que se “lembrar” de grande quantidade de informação para usar bem o sistema. Como vimos no capítulo 2, a teoria psicológica para a capacidade da memória humana define em 7+-2 itens de informação (*chunks*) a capacidade de nossa memória de curta duração. Sobrecarrega-la significa exigir maior processamento cognitivo para atividades de uso do sistema nem sempre relevantes à tarefa propriamente dita. Quantas vezes, ao navegar pela Internet, você se esqueceu do que estava buscando inicialmente?

### ***Criar para o erro***

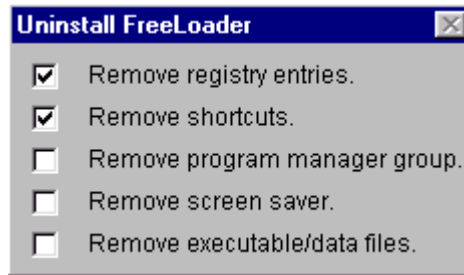
Pressupõe a observação geral sobre design de que mesmo que se tenha feito o melhor sistema possível, usuários – tanto os novatos quanto os experientes – cometerão erros ao usá-lo. O design que considera a condição humana do erro deve forçar ações que previnam ou dificultem o erro do usuário. Prover ações reversíveis ajuda a minimizar a ansiedade e o medo do novato de “destruir alguma coisa”. Mensagens de erro efetivas e feedback ajudam o usuário a saber o que fazer quando o resultado de suas ações não produz o que ele espera, atravessando o golfo da avaliação e continuando o ciclo de interação com o sistema. Um bom exemplo de aplicação desta *guideline* é a possibilidade de desfazer operações (*undo*) e repetir operações (*redo*) presentes em algumas interfaces.

Um contra-exemplo bastante simples é a mensagem dos *browsers* para endereços inválidos de páginas Web: “*ERROR 404*”

### ***Manter consistência***

Como vimos na seção anterior, consistência emerge do uso de padrões, que são mantidos ao longo do design de todos os componentes que constituem o produto. Consistência também é derivada do uso apropriado de metáforas que ajudam o usuário a construir e manter um modelo mental apropriado do sistema – idealmente coincidindo com o modelo mental do próprio designer.

A imagem ao lado é a janela que é mostrada ao se desinstalar o *FreeLoader*, um *browser* de internet *off-line* (Shame, 1999). Os *checkboxes* convidam à manipulação do usuário, sugerindo que ele pode indicar quais componentes do software devem ser desinstalados. Ao contrário, porém, os *checkboxes* são usados nesse software para mostrar o progresso do estado de desinstalação! Conforme a desinstalação progride, o sistema vai assinalando nos *checkboxes* as partes já completadas.



Além da necessidade de manter a consistência, *guidelines* devem ser aplicadas de forma cuidadosa, uma vez que, como princípios gerais, são interpretadas. Durrett e Trezona (apud Preece *et al.* 1994 p.490) apresentam um princípio de design prescrevendo o número de cores a serem usadas na tela, sugerindo que “não deve ser esperado que o usuário médio lembre (o significado de) mais que 5 a 7 cores...*displays* deveriam ter não mais que 4 cores...”

A *guideline* apresentada mostra uma distorção na interpretação da teoria psicológica na qual a *guideline* foi baseada. O uso exagerado de cores deve ser evitado por

problemas perceptuais – de distração principalmente e não por problemas de cognição – significado, lembrança.

Esse tipo de distorção mostra que simplesmente “aplicar” *guidelines* não leva a um bom design. Princípios de design devem ser interpretados e traduzidos em estratégias que produzam regras de design não-ambíguas, apropriadas ao sistema. Essas regras de design, embora também sejam chamadas de “*guidelines*” por alguns autores, são instruções que podem ser seguidas pelo designer sem exigir muita interpretação. Exemplos de regras: “usar DD-MM-AA para entrada numérica”, “posicionar o botão OK no canto inferior direito da tela”. Regras são mais comuns nas *guidelines* de determinados fabricantes, para definir a identidade visual da interface e garantir consistência tanto no produto, quanto entre produtos de um mesmo fabricante.

A aplicação de *guidelines*, embora não seja trivial, ajuda o designer a focar no que é necessário e a lidar com restrições e compromissos de design. A definição de *guidelines* a serem utilizadas no design de determinado sistema deve ser acompanhada de exemplificação de seu uso, exceções e dados psicológicos que a justificam. A Tabela 3.1 mostra um exemplo de *guideline* para “formato consistente”, extraída de Smith e Mosier, (apud Preece *et al.* 1994 p. 491).

<b>Guideline</b>	<b><i>Adotar uma organização consistente para as posições na tela, dos vários elementos do sistema</i></b>
<b>Exemplo</b>	Posição para título Área para dados de saída Área para opções de controle Área para instruções Área para mensagens de erro Área para entrada de comando
<b>Exceção</b>	Pode ser desejável mudar formatos para distinguir entre tarefas diferentes
<b>Comentário</b>	Consistência ajuda na orientação do usuário

**TABELA 3.1** - EXEMPLO DE PROPOSTA DE *GUIDELINE*

A avaliação de *guidelines* não é tarefa simples; exige conhecimento especializado nas teorias subjacentes que suportam a *guideline*, em dados de uso disponíveis que permitam generalizações, deduções sobre os efeitos da não observância do princípio, etc. Entretanto, fica mais fácil julgar sua aplicabilidade a contextos de design quando apresentadas com seus respectivos argumentos e teorias que as fundamentam, como ilustrado na Tabela 3.1



## METÁFORAS NO DESIGN DE INTERFACES

*...a good metaphor is essential to an easy-to-use human interface.*  
(Erickson, 1990)

Metáforas nos ajudam a construir Modelos Mentais sobre o artefato com o qual interagimos e, muitas vezes elas representam nossos Modelos Mentais (ver capítulo 2), permitindo-nos usar conhecimento de objetos concretos, familiares e experiências anteriores para dar estrutura a conceitos mais abstratos.

Lakoff e Johnson (1980) descrevem metáforas como o entendimento e a experimentação de uma coisa em termos de outra. Erickson (1990) define metáfora como um emaranhado invisível de termos e associações que é subjacente à maneira como falamos e pensamos sobre um conceito. É essa estrutura estendida que faz da metáfora parte essencial e poderosa de nosso pensamento. Nossa linguagem é baseada em abstrações metafóricas, como introduzido no Capítulo 1. Muitas coisas são associadas a “dinheiro”, por exemplo, o “tempo”: gastamos, perdemos, economizamos, roubamos de alguém... A interface de nossos sistemas computacionais está repleta de elementos metafóricos, a começar dos termos “interface”, “interação”. Na Desktop, a mais famosa das metáforas em interfaces, lidamos com “objetos” na tela, “pincéis” de desenho, “assistentes”, em diferentes tipos de “diálogo”.

Na metáfora, a comparação entre os domínios origem e destino é implícito. Embora uma metáfora sugira o relacionamento entre os dois domínios, é deixado para o usuário elaborar, descobrir, construir os detalhes da relação (Neale e Carroll, 1997). Bruner (1960) considera as metáforas como um mecanismo de sustentação (*scaffolding*) para o aprendizado, possibilitando que informação previamente aprendida torne-se aplicável a novas situações. O foco no uso de metáforas em interfaces evoluiu da motivação inicial como facilitadora do aprendizado para incluir a facilidade de uso.

## OLHANDO MAIS DE PERTO O ASSUNTO

*Até que ponto e como elas ajudam o usuário a interagir com sistemas computacionais?*

Pessoas usando o processador de texto pela primeira vez vêem similaridade com a máquina de escrever – ambos têm elementos em comum: um teclado, barra de espaço, tecla de retorno. Ambos têm, também, relações em comum: somente um caractere pode ser teclado por vez, ao pressionar-se uma tecla, o caractere correspondente aparece em um meio visível, etc. Essa similaridade é que permite

que o sujeito ative o MM da máquina de escrever para interpretar e predizer como o processador de textos funciona. Elementos e relações são, portanto, carregados de um domínio familiar para um domínio não familiar.

Preece *et al.* (1994) distingue dois tipos de metáforas: as verbais e as virtuais. Metáforas podem ser fornecidas na forma de instrução escrita ou falada. Junto com o processador de texto, por exemplo, instruções de como os arquivos são criados, armazenados e recuperados no sistema podem ser apresentadas em termos do “arquivo de aço”. Em vez de criar metáforas verbais, a Xerox criou a metáfora na interface, com base no escritório real, tornando virtual o topo da mesa de escritório. *Star* apresentava o equivalente eletrônico para os objetos físicos do escritório – papel, pastas, documentos, arquivos, bandejas, e ações similares: abrir, fechar, copiar, etc. Em vez de serem entidades abstratas, com nomes arbitrários, arquivos foram transformados em representações pictóricas, fáceis de identificar e entender.

A metáfora “verbal” convida o usuário a “perceber” as similaridades e diferenças entre o sistema e o domínio familiar. A metáfora “virtual” é parte da interface, e combina o sistema e o domínio familiar em uma nova entidade. Através de metáforas virtuais o usuário é levado a desenvolver um MM mais próximo do mundo metafórico – MM funcional e não um modelo do sistema subjacente (MM estrutural).

Nem tudo é similar, entretanto. Há um ponto em que a metáfora deixa de acomodar características do sistema novo. Por exemplo, a tecla de retrocesso no processador de textos também apaga o último caractere. O *Shift-Caps*, em teclas com mais de um tipo, não aciona o tipo superior, só funcionando para expressar letras maiúsculas. Essa perda de paralelismo entre o domínio familiar e o não familiar apresenta contraposições às expectativas do usuário de como os elementos e suas relações funcionam. Há propriedades que não são mapeáveis de um domínio para outro. Em algum momento o usuário precisará entender como o sistema novo funciona, como um sistema computacional que é. A metáfora do desktop, na realidade é uma composição de metáforas, assim criada para permitir flexibilidade de ação. Por exemplo, a barra de rolagem é um objeto que não existe no escritório real. Menus e janelas foram emprestados de outros contextos.

Uma metáfora na interface que sugira o MM incorreto causa dificuldades ao usuário. Por exemplo, para eliminar arquivos e documentos, a lixeira é uma metáfora intuitiva. Entretanto, no Macintosh a metáfora foi estendida para incluir uma função nova: o “*eject*” do disquete. O arraste do disquete para a lixeira para retirá-lo do computador é incompatível com a associação metafórica anterior e causava problemas conceituais ao usuário, que tinha medo de ter o conteúdo de seu disquete deletado.



[illegible]

Em relação à estrutura, o objetivo é verificar quanto de estrutura a metáfora provê para o usuário pensar no sistema. Em relação à aplicabilidade deve ser verificado quanto da metáfora é relevante ao problema; metáforas que podem conduzir o

Em relação à estrutura, o objetivo é verificar quanto de estrutura a metáfora provê para o usuário pensar no sistema. Em relação à aplicabilidade deve ser verificado quanto da metáfora é relevante ao problema; metáforas que podem conduzir o

usuário na direção errada ou levantar falsas expectativas devem ser evitadas. Quanto à representação, metáforas ideais têm representações visuais distintas e palavras específicas associadas. Em relação à adequação à audiência, deve-se verificar se o público-alvo entende a metáfora; ela seria inútil em caso negativo. Estrutura adicional é uma característica desejável, uma vez que extensão do sistema pode ser necessária mais tarde.

Madsen (1994, p. 59-60), com base em estudos de casos realizados e coletados da literatura, propõe uma série de *guidelines* para o design baseado em metáforas. O autor distingue 3 diferentes atividades nesse processo: (1) geração de metáforas candidatas à aplicação no design; (2) avaliação com relação à adequação ao domínio particular de tarefas e (3) desenvolvimento ou seja adaptação da metáfora à situação de design. Detalharemos a seguir as *guidelines* sugeridas para cada fase.

- **Fase (1) – geração de metáforas:**  
*Observar como os usuários entendem seus sistemas computacionais.*  
*Construir sobre metáforas já existentes.*  
*Usar artefatos predecessores como metáforas.*  
*Notar metáforas já implícitas na descrição do problema.*  
*Procurar eventos do mundo real que exibam aspectos chave.*
- **Fase (2) - avaliação de metáforas candidatas ao design:**  
*Escolher uma metáfora com uma estrutura rica.*  
*Avaliar a aplicabilidade da estrutura.*  
*Escolher uma metáfora adequada à audiência.*  
*Escolher metáforas com significado literal bem entendido.*  
*Escolher metáforas com uma distância conceitual entre a fonte e o significado metafórico.*  
*Ter pelo menos um conceito como “ponte” entre o significado literal e o metafórico.*  
*Não necessariamente incorporar a metáfora no design final.*
- **Fase (3) - desenvolvimento do sistema propriamente dito:**  
*Elaborar o conceito principal.*  
*Procurar novos significados para o conceito.*  
*Reestruturar a nova percepção da realidade.*  
*Elaborar suposições tornando explícito o que a metáfora esconde e o que ela salienta.*  
*Contar a estória da metáfora, falando do domínio alvo como se ele fosse o domínio fonte.*  
*Identificar as partes não usadas da metáfora.*  
*Gerar situações de conflitos.*

Na pesquisa mais recente, as metáforas são concebidas como mapeamentos entre domínios que possibilitam ao usuário usar conhecimento e experiências específicos

de um domínio familiar, para entender e se comportar em situações que são novas. Nesse processo, além do uso de representações emprestadas de domínio familiar, vários autores sugerem que a concepção do design baseado em metáforas deva incorporar também o contexto de uso de tais representações, resultando em influências maiores no design de sistemas.

O Jogo do Alvo (Nied, 2000) é um exemplo do uso do tiro ao alvo como metáfora para visualização de conceitos de Controle Estatístico de Processo (CEP). Como mostra a Figura 3.9, a disposição dos tiros no alvo representa visualmente a distribuição de medidas de determinada peça, em relação a um valor nominal almejado (o centro do alvo).



FIGURA 3.9 - TELA DO JOGO DO ALVO

O alvo é uma metáfora já implícita na descrição do problema, uma vez que o objetivo no processo de manufatura é conseguir produzir peças cuja medida seja o mais próxima possível do valor nominal (o centro). Além de possuir um significado literal bem entendido, a estrutura da metáfora serve para representar condições de estabilidade do processo, pela distribuição dos tiros no alvo.

Para o trabalhador da linha de manufatura, a metáfora do alvo consegue captar abstrações do processo e facilitar a interpretação de situações reais sem exigir dele sofisticação matemática. O alvo permite reestruturar uma nova percepção da realidade da manufatura com base na associação de conceitos do CEP com configurações de tiros no alvo.

## DESIGN BASEADO EM CENÁRIOS



A mudança de paradigma que nos leva a enxergar os computadores não apenas como artefato tecnológico, mas principalmente como artefato da cultura humana, leva-nos a repensar os objetivos e métodos de design e desenvolvimento de sistemas.

Como artefato tecnológico, o computador deve produzir resultados corretos, ser confiável, executar eficientemente, ser fácil – ou possível de manter. Como artefato da cultura, entretanto, outros requisitos são necessários: os computadores devem ser acessíveis a pessoas de outros domínios de conhecimento, não envolvidos diretamente com a tecnologia; para tal devem ser fáceis de aprender e de usar. Além de atingir as expectativas das pessoas, devem estender as atividades humanas melhorando a qualidade de vida e levando o usuário à satisfação de suas experiências de trabalho, educação e lazer.

Como implicação dessa nova concepção do uso de computadores, são necessários métodos flexíveis e ricos para incorporar descrições de usuários potenciais e dos usos que eles podem fazer de um sistema imaginado à lógica do design. Idealmente o próprio usuário deveria ser envolvido nesse processo.

Carrol (1997) propõe para o processo de design, buscar novos vocabulários e representações – acessíveis aos próprios usuários – para discussão e caracterização de design em termos de atividades projetadas. Além disso, tais representações devem ser integradas e coordenadas com outras produzidas ao longo do desenvolvimento do sistema. A avaliação de alternativas de design deve ser feita, também, com critérios orientados ao uso e integrados à avaliação tradicional de correção, confiabilidade, eficiência, etc.

Cenários foram propostos por Carrol (1997) como um meio de representar, analisar e planejar como um sistema computacional pode causar impacto nas atividades e experiências do usuário. Um cenário é uma descrição em geral narrativa, mas também em outros formatos, que as pessoas fazem e experimentam conforme elas imaginam ou tentam fazer uso de sistemas e aplicações.

Um exemplo simples de cenário mostrando a narrativa de uma situação na qual uma pessoa (usuário) descreve como ela imagina o uso de um sistema prospectivo, é mostrado a seguir.

**“Cenário 1:** *formação em planejamento e gerenciamento sem interromper o trabalho normal*

*A fábrica X tem um problema. Está sendo pressionada pela matriz, que fica em outro país, para melhorar a formação e capacidade de planejamento e gerenciamento de seus funcionários, estimulando-os a se familiarizar com sistemas novos como JIT, TQC, TPM e outros. Vários exercícios, folhetos, cursos de treinamento impressos e em vídeo são recebidos da matriz para treinar e estimular a formação de funcionários em todos os níveis nos novos sistemas. Só que estes exercícios e cursos requerem a presença de vários funcionários ao mesmo tempo, de vários setores da fábrica durante muitas horas, interferindo com a produção da fábrica.*

*A matriz também sugere treinamento e ensaios com clientes e fornecedores. Mas os principais clientes e fornecedores de X estão dispersos por várias partes do Brasil e é inviável convidá-los para participar dos cursos e exercícios presenciais. Os exercícios não são utilizados e a formação dos funcionários fica postergada.*

*Consultado, o Nied-Unicamp sugere o desenvolvimento de sistemas computacionais de suporte à atividade de formação a distância e o uso da Internet, Intranet e Extranet montados pela empresa, para implementar os exercícios e cursos para os funcionários junto com os clientes e fornecedores. Todos estes funcionários podem participar destas atividades de formação em horas convenientes à não-interrupção da produção.”* (extraído de Mazzone, J. documento interno Nied-Unicamp)

Cenários iniciais do tipo exemplificado ajudam o designer a clarificar as metas de design. Desenvolver e explorar o espaço de design com um mínimo de comprometimentos ajuda o designer a entender o que é necessário fazer e conhecer as percepções individuais do usuário para com o sistema, atitudes em direção à colaboração, etc.

Cenários podem captar conseqüências e compromissos de design a serem analisados. Por exemplo, do cenário pode-se extrair considerações do tipo:

*“as redes de comunicação baseadas no computador são um meio bastante interessante para a situação descrita no cenário 1, mas essa solução de design deve considerar se há cultura do uso desse meio na fábrica X, além das questões tecnológicas relacionadas à velocidade da informação nessas redes”*

Embora a forma mais comum para representar cenários seja a textual, podem estar também na forma de *story-boards*, *cartoons* anotados, maquetes em vídeo, protótipos em *script*, etc. O formato usado para a expressão de cenários tem sido bastante flexível e variado entre os praticantes do design baseado em cenários. Karat e Bennett (apud Preece *et al* 1994) propõem os seguintes componentes para cenários:

**Nome** – um rótulo curto para referência a um cenário específico;

**Descrição** – em geral texto ilustrando uma situação específica;

**Lógica Essencial** – com relação ao usuário, representações e ações que devem estar disponíveis ao usuário, independentemente de aspectos relacionados à implementação; com relação ao sistema, informações necessárias para que o sistema funcione como requerido;

**Passos Genéricos** – seqüência de passos que o usuário realizaria, independentemente de aspectos de implementação;

**Passos específicos** – seqüência de ações do usuário seguidas de feedback do sistema, considerando possibilidade de ações erradas do usuário.

Os níveis de descrição variam, podendo ser articulados em diferentes níveis de granularidade para especificar mais precisamente a funcionalidade do sistema. A seguir exemplificamos um segundo cenário para sistema relatado no Cenário 1, com base em sistema desenvolvido e relatado em Baranauskas *et al* (1999):

**Cenário 2** – *Iniciando Jogo da Fábrica: atividade síncrona a distância e baseada em sistema computacional*

<descrição> - *Funcionários da fábrica X conectam-se via Internet e iniciam Jogo da Fábrica: uma simulação de conceitos e processos de manufatura com objetivo de formação.*

<lógica essencial> - (usuário) *Cada usuário em seu local de trabalho, ao conectar-se ao sistema, vê a tela inicial do jogo e informações sobre a conexão dos demais participantes, cada um ocupando uma célula da linha de manufatura representada no jogo. Cada usuário saúda os demais que estão conectados a distância, pelo canal de comunicação, aguardando início do jogo.*

(sistema) *Informação necessária para conexão: IP da máquina servidora*

<passos genéricos> - *Buscar opção de conexão no menu. Entrar com dados solicitados.*

<passos específicos> - *Selecionar “.....*

*...*

*Entrar com IP do servidor”*



O cenário identifica o usuário como tendo certas motivações para o uso do sistema, descreve as ações tomadas e razões para essas ações. Para o designer, ajuda a visualizar aspectos da atividade e experiência adquirida ou necessária do usuário.

Carrol (1997) propõe o uso de cenários ao longo de todo o ciclo de design e desenvolvimento do sistema. Durante a fase de análise de requisitos designers e usuários negociam explicitamente os cenários e descrições do domínio de uso do sistema. Descrições de cenários hipotéticos facilitam na descoberta das necessidades do usuário que não são óbvias ou aparentes para eles próprios.

Cenários podem ser a unidade de análise para desenvolver o design *rationale*, explicando as decisões de design através de cenários particulares de interação do usuário e da análise de cenários alternativos.

Na fase de design propriamente dita, cenários podem ser analisados para identificar os objetos centrais do domínio do problema e articular o estado, comportamento e interação funcional dos objetos de design. A abordagem *use-case* em análise orientada a objetos faz uso desse tipo de visualização de situações concretas com o objetivo de identificar objetos computacionais essenciais no sistema.

Na fase de avaliação, cenários podem ser usados para se coletar informação detalhada de como os usuários percebem o sistema. Design de telas podem ser apresentados a usuários potenciais que tentam explicar o que pensam ser possível fazer e efeitos esperados de suas ações.

Mesmo na fase de implementação o uso de cenários apresenta benefícios, uma vez que ajuda a manter os designers focados em dar suporte às atividades dos usuários.

A documentação do sistema pode facilitar, também, o próprio treinamento do usuário; as pessoas fazem melhor uso da documentação se ela é apresentada no contexto de tarefas típicas que eles têm que realizar, especialmente as mais críticas.

A proposta de design baseado em cenários, de certa maneira, se contrapõe à abordagem convencional para o ciclo de design e desenvolvimento de sistemas. Na abordagem estabelecida tradicionalmente, as descrições são abstratas, com foco em tipos genéricos, o processo é completo, exaustivo, e orientado à tecnologia, os métodos são formais e rigorosos e os resultados bem especificados. Na abordagem baseada em cenários as descrições são concretas, com foco em instâncias particulares, o processo é fragmentado e aberto, orientado ao trabalho, o método é informal e coloquial e os resultados imaginados apenas.

Uma contribuição importantíssima da abordagem dos cenários no processo de design é o estabelecimento de um canal de comunicação de mão dupla, usuário-designer. Cenários são a língua franca da ação e experiência do usuário final.

A proposta do design baseado em cenários pressupõe a visão de sistemas computacionais como transformadores das tarefas do usuário e de suas práticas sociais. Estórias são elementos coesivos importantes em qualquer sistema social; os cenários que a equipe compartilha motivam e direcionam a construção do grupo. Representam, ainda, uma busca pelo equilíbrio entre intuição criativa e análise, entre a flexibilidade e a informalidade, necessários para a evolução sistemática em direção a um sistema usável.

## DESIGN PARTICIPATIVO

*Design with the user,  
rather than design for the user...*  
(Kuhn e Winograd, 1996)

Conforme o título indica, o Design Participativo (DP) caracteriza-se pela participação ativa dos usuários finais do software ao longo de todo o ciclo de design e desenvolvimento. Mais do que serem usados como fontes de informação ou serem observados em sua rotina de trabalho, ou no uso do produto, os usuários finais trazem contribuições efetivas em todas as fases do ciclo de design e desenvolvimento, que refletem suas perspectivas e necessidades. A participação do usuário não é restrita aos estágios de testes de protótipos ou avaliação, mas acontece ao longo do processo de design e desenvolvimento.

O Design Participativo em geral acontece no local de trabalho, incorporando o usuário não somente como sujeito de observação e experimentos, mas como membro da equipe de design. Três características específicas definem o DP: ele é orientado ao contexto (de trabalho), envolve colaboração em vários níveis e apresenta uma abordagem iterativa ao design.

Em uma mesa redonda na Conferência sobre Design Participativo de 1994, Tom Erickson, da Apple Computer (citado em Kuhn e Winograd, 1996), definiu quatro dimensões ao longo das quais a participação do usuário pode ser medida: 1. a diretividade da interação com os designers; 2. a extensão do seu envolvimento no processo de design; 3. o escopo de participação no sistema como um todo; 4. o seu grau de controle sobre as decisões de design.

O movimento do Design Participativo teve origem no início da década de 70, na Noruega, com Kristen Nygaard (um dos criadores de Simula), quando este colaborou com o sindicato para criar o *Codetermination Agreement*, especificando os direitos dos trabalhadores de participar em decisões de design relativas ao uso de novas tecnologias no trabalho.

Outro marco inicial do DP foi o Projeto DEMOS, que ocorreu na segunda metade da década de 70 e envolveu uma equipe interdisciplinar de pesquisa nas áreas de Ciência da Computação, Sociologia, Economia e Engenharia. Era financiado pela *Swedish Trade Union Federation* e o mote do projeto era *Trade Unions, Industrial Democracy and Computers* (Kuhn e Winograd, 1996).

Entre as motivações para o uso de abordagens participativas em design estão: a questão da democracia, o compromisso com o desenvolvimento organizacional, a eficiência, expertise, qualidade potenciais, e a efetividade do ponto de vista epistemológico.

A idéia do DP foi concebida em sua formulação escandinava original, no contexto de um movimento em direção à democracia no local de trabalho: o desenvolvimento de competências e o poder de o trabalhador exercer influência em decisões que afetariam seu trabalho. Nessa proposta há um compromisso implícito com o desenvolvimento organizacional, através da crença de que o sistema terá mais chances de ser aceito se seus usuários finais estiverem envolvidos no processo.

Eficiência, expertise e qualidade seriam conseqüências dessa abordagem. A efetividade do design e desenvolvimento de software é aumentada se inclui a expertise dos próprios usuários. A eficiência é aumentada se os usuários finais provêm entradas para outros design e feedback em um design completo. A qualidade no design e no sistema resultante é aumentada através de um melhor entendimento do trabalho do usuário e melhor combinação do background dos diversos participantes.

Finalmente a efetividade epistemológica é sustentada pela premissa básica de que nenhuma pessoa ou disciplina, isoladamente, tem todo o conhecimento necessário para o design do sistema.

A concepção original do DP é, portanto, sustentada pela crença de que o trabalho democrático no nível de design tem o potencial de melhorar ambos: tanto o processo de desenvolvimento do software quanto o trabalho dos usuários. Nesse sentido, democracia, epistemologia e efetividade comercial caminham juntas.

Convém ressaltar aqui que a terminologia do “design participativo” tem sido usada em modelos de design que não necessariamente concordam com as motivações do DP, para simplesmente expressar alguma forma de participação do usuário, ou para se referir ao uso isolado de métodos do DP.

Desejamos distinguir essa apropriação da linguagem e de alguns métodos, do DP propriamente dito. Nosso objetivo nesta seção é, portanto, clarificar o conceito de participação como infraestrutura e referencial teórico subjacente para um grupo de

métodos e processos complexos, que não são lineares. Discutiremos, a seguir, uma taxonomia para práticas participativas em design.

Os métodos em DP caracterizam-se pelo uso de técnicas simples e pouco comprometimento com recursos; as técnicas de *brainstorming*, *storyboarding* e de *workshops* são bastante utilizadas.

Embora seu uso seja mais conhecido na fase de design propriamente dito, as práticas participativas estendem-se ao longo de todo o ciclo de vida (convencional) do software, desde a fase de identificação do problema, passando pelas fases de levantamento e análise de requisitos, design, avaliação, customização e re-design.

A seguir apresentaremos resumidamente alguns métodos que podem ser utilizados em diferentes momentos do processo de design e desenvolvimento do software. Nosso objetivo é ilustrar, do ponto de vista pragmático, o DP. Para uma visão mais abrangente, Muller (1997) apresenta uma coleção bastante extensa de 61 práticas participativas utilizadas em design.

Em nossa descrição, abordaremos o que o método faz, que materiais usa, como as pessoas se comunicam umas com as outras, como utilizam os materiais, como tomam decisões, quem é envolvido no trabalho e se existem papéis especiais de certos membros da equipe, quais os benefícios que são produzidos, como o resultado é usado.

***Storytelling Workshop*** é um método usado na fase de identificação e clarificação do problema de design.

Cada participante de um grupo de usuários finais e facilitadores (max. de 20 pessoas) traz para a oficina duas histórias curtas sobre o uso de sistemas computacionais – em geral experimentadas em seu trabalho. Uma história deve ser positiva e outra negativa com respeito ao resultado desse uso. Os participantes compartilham suas histórias, comentando semelhanças e contrastes de suas experiências. Nenhum material especial é requerido. Como resultados da oficina, são apontados: uma coesão aumentada entre os usuários finais e entre esses últimos e os designers, reconhecimento das dificuldades e consciência de que elas não são únicas, conhecimento de características e dificuldades da população de usuários pelos designers.

***Picture Card*** é um método utilizado na fase de análise de requisitos, em situações nas quais os usuários finais e profissionais de design e desenvolvimento do software não compartilham, ainda, a mesma linguagem. Eles se comunicam usando cartões pictóricos para desenvolver a representação do trabalho.

São utilizados como material cartões contendo figuras de objetos e eventos do mundo de trabalho do usuário. Esses cartões são agrupados em seis categorias:

Pessoa, Ação, Estação, Ferramenta, Evento, Local (PAEFEL). Os cartões são arranjados em seqüências lineares começando com as categorias PAEFEL e refinando-as em subclasses específicas, refletindo as histórias e cenários do ambiente de trabalho.

Como resultado, as histórias contadas pelos usuários, inicialmente expressas através de cartões, são traduzidas em texto; para o designer servem para tornar explícito o dialeto do trabalho e alimentar o dicionário de objetos, ações, etc.

O tamanho do grupo deve ser pequeno, com alguns usuários “contadores de histórias”, o designer que controla a mesa de cartões e observadores.

**HOOTD- Hierarchical Object-Oriented Task Decomposition** é um método para ser utilizado na fase inicial de design, embora também possa ser útil na fase de análise.

Participantes decompõem uma descrição de tarefa em objetos e ações e assinalam grupos desses objetos a janelas de interface. Cartões são usados como material.

Cada participante, em paralelo, escreve cada tarefa – representada por um substantivo e um verbo – em seu cartão. Os participantes, então em grupo, ordenam esses cartões em pilhas segundo critério de escolha do grupo. O esquema é registrado. Reordenam, então, segundo outros critérios, registrando todos os esquemas. Escolhem um dos esquemas. Cada pilha do esquema escolhido torna-se uma tarefa do domínio, contendo em uma janela da interface os objetos e ações da pilha de cartões.

Como resultados tem-se a definição das janelas de interface e seus respectivos objetos. É recomendável que o grupo não seja grande. Análise de tarefas (com o GOMS, por exemplo) pode ser usado como método formal complementar.

**BrainDraw** é um método participativo para uso na fase de design propriamente dita. É constituído de um *brainstorming* cíclico, gráfico, com o objetivo de preencher rapidamente um espaço de várias opções de design para a interface.

O material é composto de papel e canetas arranjados em uma série de estações de desenho colocadas em círculo. Cada participante faz um desenho inicial em uma das estações. Ao final de um intervalo de tempo estabelecido, cada participante move-se para a estação seguinte e continua o desenho lá encontrado. O processo continua rodando até que todos tenham colaborado na criação de cada um dos outros participantes.

Como resultado tem-se a geração de muitos design candidatos à interface do sistema, cada um deles tendo a participação de todos os envolvidos. Cada design resultante é a fusão da idéia de todos e não são idênticos uma vez que cada um deles teve um início diferente.

Um grupo de tamanho médio (aproximadamente 10 pessoas) de usuários finais, designers e artistas são os participantes potenciais. Alternativamente ao movimento dos participantes pelas estações de desenho, os desenhos podem rodar pelos participantes colocados em círculo.

**Icon Design Game** é um método participativo que pode ser utilizado na fase de design para a criação dos ícones e símbolos gráficos da interface. Um dos participantes (*sketcher*) desenha ícones enquanto que os outros tentam “adivinhar” o conceito que o *sketcher* está tentando expressar. Os desenhos tornam-se rascunhos para a criação de ícones.

O material utilizado é composto de papéis de desenho e canetas. O participante no papel de *sketcher* seleciona um conceito e tenta comunicar ao grupo apresentando desenhos relacionados ao conceito. O grupo tenta descobrir enquanto um observador toma notas sobre desenhos que parecem mais efetivos ou mais confusos. Os desenhos que expressam melhor o conceito são passados para a produção gráfica dos ícones.

Como resultado tem-se, portanto, *sketches* de ícones para arte final. Pode-se usar o método, também na escolha de metáforas para a interface.

Dependendo do tamanho do grupo, este pode funcionar no estilo cooperativo ou subdividido em vários grupos para produção competitiva dos desenhos.

**CISP – Cooperative Iterative Storyboard Prototyping** é um método que pode ser usado em várias fases do ciclo de design e desenvolvimento: análise de requisitos, design e avaliação.

Uma equipe de designers e usuários gera e modifica cooperativamente designs de interfaces, avaliam interfaces existentes comparando alternativas. Um software associado ao método ou outro ambiente para criação de *storyboards*, como o *HyperCard*, *Borland Delphi*, etc. em geral são utilizados como material.

O processo envolve iterações de 3 passos principais: exploração do *storyboard* para realização da tarefa pelo usuário final, enquanto o software registra; avaliação do *storyboard*, através de análise e discussão do registro da interação; modificação do *storyboard*.

Como resultado tem-se o *storyboard* ou o protótipo melhorados e o registro da interação dos usuários. Podem ser usados em complemento ao CISP na fase de avaliação, métodos de inspeção de usabilidade (que serão apresentados no Capítulo 4).

**Buttons Project** é um método para ser utilizado no pós-design, na customização do sistema pelo usuário final.

O material utilizado para compartilhar customização é um software que suporta o design de funções customizáveis. Usuários compartilham suas customizações enviando botões uns para os outros.

Através de templates os usuários especificam funcionalidades em botões. Enviam esses botões uns para os outros. Receptores de botões podem modificá-lo.

Como resultado tem-se nova funcionalidade compartilhada entre os usuários, além do registro das inovações na forma de customizações executáveis.

**Priority Workshop** é um método utilizado no re design de um sistema. Usuários e designers colaboram na prática do re design através de uma sequência de oito atividades conduzidas em formato de workshop.

O processo é iniciado com uma discussão introdutória de objetivos. Segue-se uma apresentação dos usuários sobre características positivas, negativas e desejáveis no sistema. Segue-se uma apresentação dos designers sobre planos e prioridades relativas ao sistema. A quarta atividade é a exploração conduzida em pequenos grupos de protótipos (em papel) alternativos. Segue-se uma discussão em plenário. Um sumário de prioridades e qualidades são rotuladas com + ou - pelos usuários. Segue-se uma discussão das conseqüências – para os usuários – das mudanças. O método termina com uma discussão final de planos de continuidade do processo.

Como resultado têm-se decisões sobre características a serem incluídas e/ou modificadas no re-design do sistema.

A Tabela 3.2 mostra a distribuição dos métodos participativos descritos nas diferentes etapas do processo de design.

Pré-design	Design	Avaliação	Pós-design
Identificação/Requisitos	Inicial/Iterativo	Testes	Customização/redesign
StoryTelling	HOOTD	CISP	Buttons
PictureCard	BrainDraw		Priority Workshop

**TABELA 3.2** - MÉTODOS PARTICIPATIVOS EM DIFERENTES ETAPAS DO DESIGN

Muller (1997) discute a relação entre práticas participativas e modelos formais e contratuais de desenvolvimento de software e comenta que certas metodologias orientadas a objeto encorajam a construção de *use cases*, conforme comentado anteriormente, como cenários para atividades do usuário relacionadas ao sistema.

Embora a terminologia seja semelhante, o foco da atenção nos *use case* é o software e o paradigma ainda é o orientado ao produto. O modelo de casos de uso não é substituto para o trabalho com usuários finais, uma vez que a definição das ações do usuário é feita pelos designers.

Também, normas como a ISO9001 para assegurar a qualidade do produto, encorajam um acordo ou contrato entre a organização que desenvolve o software e a organização onde o usuário trabalha. O usuário não é representado formalmente nessa relação. O foco do padrão é a qualidade técnica e necessidades das gerencias em contraste com a qualidade de uso, estética e necessidades do usuário final.

Há vários questionamentos sobre as dimensões éticas e políticas na forma de participação do usuário no design do sistema. Algumas técnicas de extração do conhecimento, por exemplo, nos sistemas especialistas, enfatizam a “participação” dos trabalhadores com o objetivo de aumentar o conhecimento que será usado pelo profissional de desenvolvimento.

Muller (1997) cita também a questão do usuário como “objeto”; certos testes de usabilidade tratam os usuários como indicadores de medidas da produtividade associada ao produto, sem considerar suas necessidades, conforto, qualidade do ambiente de trabalho. Não é o usuário quem escolhe quais atributos de sua experiência são relevantes.

Há ainda o resultado da participação de determinados clientes potenciais para determinação de atributos atrativos ao mercado, para uso em campanhas de marketing. Outro questionamento que se coloca é a ilusão de controle que a participação pode dar ao usuário enquanto que o poder de decisões continua sob o controle da hierarquia superior da organização.

Derivadas do DP, várias abordagens ao design com foco em aspectos da “democracia industrial” são discutidas na literatura: *situated activity* (Suchman, 1987), *work-oriented design* (Ehn, 1988), *design for learnability* (Brown, Duguid, 1992), *situated design* (Greenbaum, Kyng, 1991), entre outras.

## MÉTODOS ETNOGRÁFICOS EM DESIGN DE INTERFACES

*Theorizing or reflecting and engaging in some form of practice are essential to every kind of human endeavor, be it research, design, or any other form of work practice*

Suchman e Trigg (1995, p.238)

Ao contrário dos métodos experimentais em Psicologia, onde os sujeitos são submetidos a situações criadas em laboratório, antropólogos e sociólogos usam



métodos etnográficos para estudar pessoas *in the wild*, em seu habitat nativo. A meta da antropologia é aprender sobre todos os aspectos de uma cultura. A observação participativa, um dos principais métodos da etnografia, ajuda o pesquisador a enxergar o mundo através dos olhos do nativo (Nardi, 1997). A observação participativa desenvolveu-se no final do século 19 e início do século 20, quando antropólogos americanos foram a campo estudar e documentar aspectos da cultura de sociedades americanas nativas.

A abordagem etnográfica em design de software é tratada no trabalho pioneiro de Suchman (1987), onde ela propõe que o ideal para a investigação da tecnologia em uso é aquele em que a atividade de trabalho ocorre naturalmente em cenários construídos pelos próprios participantes.

Entre os objetivos e tarefas principais da abordagem etnográfica em design está o entendimento da prática corrente do trabalho das pessoas usando tecnologias. Podemos estar interessados em entender como as pessoas usam um espaço compartilhado de desenho para um trabalho conjunto de design, a partir do cenário dos próprios participantes. Ou podemos estar interessados em entender como as pessoas usam o protótipo de uma nova ferramenta para fazer seu trabalho, em cenário que pode ser construído pelo pesquisador.

Vários tipos de registros da observação podem ser realizados, de forma a captar em diferentes mídias, diferentes aspectos do ambiente observado. Em registros orientados ao ambiente propriamente dito, uma ou várias câmeras de vídeo são posicionadas de forma a cobrir o máximo possível da atividade sendo analisada, no espaço físico. Por exemplo, numa secretaria de atendimento ao público, uma câmera captaria o balcão de atendimento aos usuários, outra o espaço físico do arquivo de aço que guarda documentos em papel, outra o computador. Esses registros poderiam mostrar a frequência de cada tipo de atividade – atendimento no balcão, uso de sistemas computacionais, uso do arquivo em papel, procedimentos e rotinas usuais, tipos de interrupção, relações entre as diferentes atividades, etc.

O registro pode ser orientado à pessoa, quando estamos interessados em entender o trabalho do ponto de vista de uma determinada pessoa em determinada função no ambiente de trabalho. Uma câmera acompanha a pessoa na seqüência de atividades que realiza. Dificuldades, ações repetitivas, re-trabalho são exemplos de dados que poderiam ser extraídos desse registro.

O registro pode ser, ainda, orientado a um objeto ou artefato tecnológico, para captar, por exemplo, situações de uso desse artefato. Na análise de usabilidade de um protótipo de sistema, por exemplo, uma câmera pode captar toda a seqüência de ações de um usuário interagindo com o sistema. O objetivo, neste caso, é analisar a interface com respeito à adequação ao trabalho a ser realizado.

Quando o objetivo é entender a tarefa, são feitos registros de pessoas que têm uma meta em comum, em diferentes locais e momentos da composição de determinada tarefa. Tomando como exemplo uma secretaria de cursos na Universidade, uma determinada tarefa – a geração de determinado relatório – pode envolver também a secretaria dos departamentos, que possui parte da informação necessária ao relatório – dados de publicações de docentes, por exemplo. Um registro orientado à tarefa envolve, portanto, registros de sub-tarefas realizadas por pessoas diferentes em diferentes locais. Relações de precedência, pressuposição, interdependência, etc. podem ser captadas desses registros.

Após o registro de uma extensa atividade, a segunda parte dos procedimentos etnográficos é a transcrição do registro para a análise inicial. No caso do registro em vídeo, o primeiro passo deve ser a descrição dos eventos observados, indexando-os cronologicamente. Uma transcrição cuidadosa do material deve incluir também as dimensões não vocais da interação: gestos, postura, por exemplo. Ainda, ações das pessoas usando máquinas ou sistemas computacionais devem ser concatenadas com registros obtidos no próprio sistema (*logs*) para análise da interação pessoa-computador e análise da inter-relação com atividades de outras pessoas.

Na análise da interação, o objetivo é descobrir as regularidades na ação das pessoas no uso dos recursos do ambiente, com outras pessoas e com o sistema computacional (ou artefato). Para isso são construídas “coleções”: instâncias de interação que queremos ver como uma classe (Suchman e Trigg, 1995). Ao colecionar essas instâncias, as características comuns e distintivas ficam mais visíveis.

Embora a fita de vídeo não elimine a necessidade de interpretação do analista, ela corrige nossa tendência de “ver em uma cena o que esperamos ver”, conforme já discutimos no Capítulo 2, Suchman e Trigg, (1995) citam um exemplo bastante interessante desse fenômeno: quando em certas circunstâncias um casal é observado sorrindo um para o outro, estando próximos fisicamente, é comum a inclusão de “toques” no relato do observador, mesmo quando não existiram na realidade.

A análise da interação é um intensivo e extenso trabalho de ver e rever várias vezes o registro feito, transcrevendo e buscando seqüências relevantes à análise. A fita de vídeo traz informações que a edição de texto – no caso de registros da observação através de anotações – não possibilita. É muito difícil captar em palavras os movimentos de mover o cursor, por exemplo, interpretar ao mesmo tempo a expressão do usuário, etc. A densidade dos detalhes de comportamento, e a ausência de vocabulário para anotações, faz do registro em vídeo uma das formas mais utilizadas nos métodos etnográficos.

### **OBSERVAÇÃO DIRETA OU INDIRETA?**

Usuários individualmente podem ser observados diretamente, fazendo seu trabalho normal ou tarefas específicas para a situação de observação. O observador, então,

toma notas de comportamentos interessantes ou registra seu comportamento de outras maneiras, por exemplo, medindo o tempo de realização de seqüências de ações.

A observação direta é considerada o método de observação mais invasivo, uma vez que o usuário fica o tempo todo consciente de que está sendo observado por outra pessoa e sua performance está sendo monitorada. Como efeitos pode haver alteração no comportamento e no nível de performance desse usuário. Essa alteração tanto pode desfavorecer quanto favorecer os resultados de uso do sistema ou artefato. Como favoreceria? Alguns usuários podem ter sua performance na tarefa melhorada, não propriamente pelos benefícios do uso do sistema, mas também por se sentirem “valorizados” ao receberem maior atenção de outros (observadores no caso). Esse fenômeno é conhecido como “efeito Hawthorne”.

Outra desvantagem da observação direta é que ela permite apenas um passo na coleta de dados, o tempo real da observação. Além disso, é nesse mesmo tempo que o avaliador deve tomar decisões do que é importante registrar. Finalmente, o avaliador raramente consegue um registro completo da atividade do usuário, embora o uso de uma notação abreviada e de um *checklist* previamente estabelecido ajudem no registro de ocorrência de determinados eventos.

A observação indireta através de gravação em vídeo cria uma distância maior entre o observador e o usuário, minimizando o sentido “invasivo” da observação. A gravação pode ser sincronizada com outros registros da interação do usuário com o sistema – arquivos *log*, por exemplo, gerando quadro completo da interação. Vários aspectos da atividade podem ser captados por câmeras diferentes; uma pode focalizar o teclado e a tela enquanto a outra focaliza o usuário, registrando para onde ele olha na tela, se consulta outro material, linguagem do corpo, etc.

Como haverá muito mais dados a analisar, o que consumirá mais tempo do designer, a observação precisa ser planejada. Deve ser definido quando começar e terminar, onde colocar fisicamente o equipamento, etc. Mesmo sendo menos invasivo do que a observação direta, os usuários são conscientes de que seu comportamento está sendo gravado. Uma maneira de reduzir o impacto é deixar o equipamento no local da gravação, por vários dias antes de começar (Preece *et al.*, 1994).

A análise dos dados coletados em vídeo pode ser baseada em tarefas, quando o objetivo da análise é saber como os usuários lidam com a tarefa, onde estão as maiores dificuldades e o que poderia ser feito. A observação pode referir-se ao contexto de trabalho do usuário, ainda sem o sistema computacional – alvo do processo de design, ou pode referir-se à tarefa mediada por versões do protótipo do sistema em processo de design.

A análise pode, ainda, ser baseada na performance do usuário, gerando esquemas classificatórios para freqüências de acertos, de erros, tempo gasto para realização de

partes da tarefa, frequência de uso de determinados elementos de interface, tempo usado em atividades cognitivas: pausas em comandos, entre comandos, em leituras inspecionando áreas de interface, etc.

Protocolos pós-evento são, também, elementos de informação importantes na análise da observação. Os usuários são convidados a ver a gravação, comentar sobre suas ações. Quando são convidados a participar da análise dos dados, eles são estimulados a relembrar detalhes úteis sobre seus problemas. Essa releitura do usuário pode, entretanto, trazer uma “racionalização” de suas ações que poderiam não corresponder exatamente ao acontecido de fato.

Protocolos pós-evento são essenciais em situações de observação de tarefas que requerem cuidadosa concentração do usuário e o seu tempo é crítico como, por exemplo, em salas de controle de tráfego aéreo, salas de terapia intensiva e outras situações na área de saúde.

Protocolos verbais são registros das falas do usuário e representam uma dimensão a mais à informação coletada, pois expressam parte da atividade cognitiva subjacente ao comportamento físico – ações, postura, gestos – do usuário. O espectro da observação é aumentado com informações sobre a maneira como o usuário planejou realizar a tarefa, a sua identificação de nomes de menus e ícones, suas reações quando alguma coisa “dá errado”, seu entendimento de mensagens fornecidas pelo sistema, sentimentos subjetivos expressos no tom de voz, em comentários que faz, etc.

Os protocolos verbais são obtidos com o método do “pensar alto” (*think aloud*): o usuário é convidado a dizer em voz alta o que está pensando enquanto realiza a tarefa. Informações extraídas de protocolos verbais são riquíssimas para o processo de análise, mas representam um esforço cognitivo extra para o usuário na realização da tarefa. O usuário deve fazer, ao mesmo tempo, a tarefa propriamente dita e falar sobre suas ações e o que está pensando. Além do nível meta cognitivo envolvido (pensar sobre o pensar), sabemos da Psicologia Cognitiva que nossos mecanismos perceptuais, cognitivos e motores são pouco eficientes em manter nossa atenção dividida por mais que poucos minutos. Um bom exemplo simples desse esforço, discutido no capítulo 2, é a tarefa de conversar enquanto se dirige um automóvel.

Outro problema a lidar no registro de protocolos verbais é como prevenir longos períodos de silêncio do usuário, uma vez que pensar, fazer e falar ao mesmo tempo não é uma situação natural. Criar um arranjo para a situação a ser observada de modo que dois usuários trabalhem juntos na realização da tarefa e possam conversar entre si. Outra estratégia seria permitir que o usuário faça perguntas ao observador, que deve responder minimamente, apenas para dar continuidade à tarefa. Podem também ser usados pelo observador, *promptings* do tipo “como você faz...?”, “porque você faz ...”, etc. Essa forma de protocolo da interação será revista no Capítulo 4, quando discutirmos testes de usabilidade.

Um método de registro bastante popular para captar informações de uso de versões mais completas de protótipos de sistemas computacionais é o *logging*. O método não requer a presença do pesquisador e parte do processo de análise pode ser automatizado; não é um método invasivo, embora possa levantar questões éticas: os usuários devem ser informados desse registro. Já existem várias ferramentas para se fazer o *logging* de software. Laboratórios de usabilidade costumam fazer uso dessas ferramentas. Há ferramentas que registram cada tecla que o usuário pressiona e o tempo exato do evento. Outras registram a interação entre o usuário e o sistema, de forma que o observador pode vê-la exatamente como ocorreu. Sistemas de *playback* (Neal e Simons, 1983, apud Preece et al., 1994) possibilitam que o observador veja na tela de seu monitor colocado em sua sala, as entradas do usuário e as respostas do sistema. O observador pode, ainda, adicionar comentários para cada operação do usuário.

A escolha do método de observação a ser utilizado em geral é um compromisso entre o tempo a ser gasto e a profundidade da análise. Um feedback informal sobre determinadas tarefas mediadas pelo sistema em design pode ser obtido em poucos dias, através de observação direta ou indireta. Para um entendimento mais detalhado das ações do usuário, observação indireta combinando gravação de vídeo com *loggings* são mais adequadas. É necessário coletar e analisar protocolos, selecionando medidas de performance relevantes, em geral revendo a fita várias vezes. Essa atividade consome tempo de análise numa proporção de 5 para 1 em relação ao tempo de registro. O desenvolvimento de ferramentas, para análise de vídeo baseada em computador, já aparece na literatura em caráter experimental (Suchman e Trigg, 1995; Preece et al., 1994).

A aplicação da análise de vídeo em design é uma constante no DIAL – *Designer Interaction Analysis Lab*, um laboratório de design da Xerox no Centro de Pesquisa de Palo Alto. Nele, um grupo de antropólogos, cientistas da computação e designers desenvolve métodos que encorajam o movimento da análise para o desenvolvimento e para a análise novamente, e cooperam na aplicação desses métodos em problemas concretos de design de sistemas.

A meta da abordagem etnográfica em design é associar intuições e possibilidades tecnológicas a um entendimento detalhado da prática do trabalho real. Suchman e Trigg (1995) ilustra essa abordagem com o design do sistema Commune - uma ferramenta multi-usuário para desenho. Nesse projeto, o grupo busca entender como o uso do espaço compartilhado de trabalho suporta e é organizado pela estrutura de atividade de desenho.

A Figura 3.10, adaptada de Suchman e Trigg (1995, p.238) ilustra os fundamentos da abordagem etnográfica em design, enquanto representa graficamente o mote do início desta seção. Design, prática e pesquisa são as três perspectivas necessárias ao processo de criação. Dependendo do vértice em que nos encontramos, temos uma determinada visão do problema e essa posição não deve ser fixa; a visão a partir de

cada uma delas, em maior ou menor grau, é necessária no processo de criação dos artefatos tecnológicos que mediam nossas tarefas.

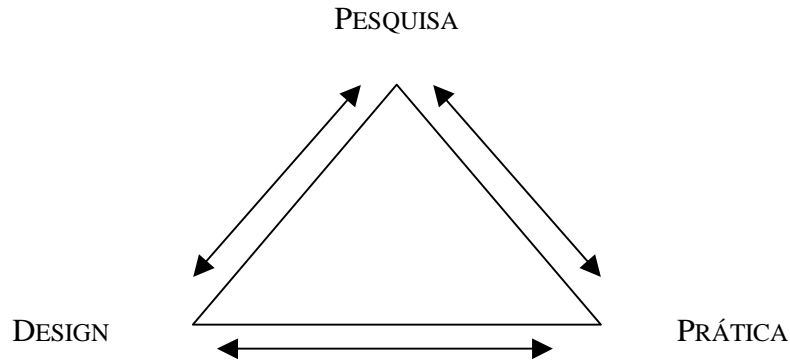


FIGURA 3.10 - ABORDAGEM ETNOGRÁFICA EM DESIGN

## SEMIÓTICA EM SISTEMAS COMPUTACIONAIS

*The pencil of the drawing program is not a real pencil that can be used to chew on, it merely stands for a pencil, represented by a collection of pixels on the screen*  
(Andersen, 1997, p.1)

As abordagens cognitivas à conceituação e ao design de interfaces, como apresentamos anteriormente estão fundamentadas principalmente nas propostas de Card, Moran e Newell (1983) do Modelo do Processador de Informação Humano (discutido no Capítulo 2) e na Teoria da Ação e Engenharia Cognitiva de Norman e Draper (1986), discutidas neste capítulo. Nesse paradigma, a interação do ser humano com o computador é governada por atividades de interpretação e avaliação realizada por usuários que têm o desafio de traduzir metas para eventos de entrada no computador e julgar reações do sistema a partir de sua percepção de elementos de saída do sistema computacional. Aspectos de comunicação são associados à diretividade semântica e articulatória e à inter-referencialidade dos elementos da entrada e saída do sistema.

Mesmo sendo o computador membro da classe dos artefatos simbólicos, somente em anos recentes, na medida em que deixou de ser ferramenta exclusiva de especialistas, e a sofisticação do software cresceu, é que essa natureza simbólica passou a atrair a atenção de grupos que estudam fatores humanos e interfaces. Embora ainda possa ser considerado uma “ferramenta”, em analogia a outras como, por exemplo,

máquinas de escrever, pincéis de pintura, pastas de arquivos, etc., o computador difere destas ferramentas por não existir ou ser usado primariamente como objeto físico, mas sim como sistema de signos. Conforme bem coloca Andersen (1997, p.1), o lápis do programa de desenho não é um lápis real, ele meramente “representa” (está para) um lápis, através de uma coleção de pixels na tela.

Os sistemas computacionais estão, cada vez mais, mediando nossas ações. Em particular, com a nova tendência de uso da tecnologia de redes de computadores, com espaços virtuais compartilhados e trocas de mensagens ele passou a ter funções similares às de outras mídia, onde a importância da Semiótica como referencial já é bem estabelecida. Na perspectiva semiótica o papel do computador é basicamente o de um *medium* – *uma substância na qual signos podem ser manifestados para uso em comunicação* (Andersen, 1997, p. 333).

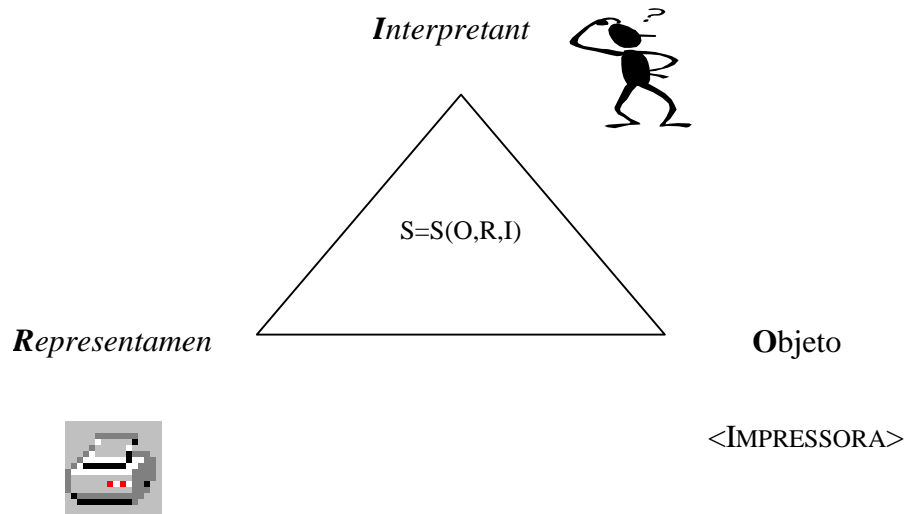
Semiótica como disciplina teve seu desenvolvimento a partir dos trabalhos do filósofo norte-americano Charles Sanders Peirce (1839-1914) e do lingüista suíço Ferdinand de Saussure (1857-1915). Peirce e Saussure formam as duas matrizes principais da Semiótica contemporânea. Os trabalhos de Saussure têm origem na lingüística enquanto que a Semiótica de Peirce é desenvolvida dentro de um corpo filosófico e é concebida como Lógica. Alguns autores diferenciam as duas vertentes usando o termo “semiótica” apenas à linha desenvolvida por Peirce enquanto usam o termo “semiologia” para as teorias derivadas da proposta original de Saussure. Não faremos distinção da terminologia e conceitos neste livro. Uma apresentação e discussão mais detalhadas das duas abordagens podem ser encontradas em Oliveira e Baranauskas (1998 a). Os conceitos básicos são apresentados aqui segundo a proposta de Peirce.

A Semiótica objetiva estudar os signos e sistemas de signos. Um signo é qualquer coisa que está no lugar de outra coisa sob determinados aspectos ou capacidades, para alguém (Peirce, CP2.228). Isto é, qualquer marca, movimento físico, símbolo, sinal, etc. usado para indicar e “transportar” pensamentos, informações e comandos constituem signos (Sebeok, 1994, p.xi). Uma foto é um signo na medida em que ela “está para” os elementos nela representados, para alguém que a interpreta. Se, na interpretação de alguém, a palavra “amarelo” está para a cor amarelo, a pronúncia da palavra “cavalo” está para o animal cavalo, fumaça está para fogo, o desenho de uma impressora na tela de um computador está para imprimir, então a palavra “amarelo”, a pronúncia de “cavalo”, a fumaça e o desenho da impressora na tela são todos exemplos de signos. Observe que, sem o signo, nossa comunicação no mundo seria muito pobre, uma vez que seríamos obrigados a nos comunicar fazendo uso, apenas, dos próprios objetos a que queremos nos referir, conforme discute Santaella (1996, p.7).

A Semiótica tem por objetivo a investigação de todas as linguagens possíveis, ou seja, a investigação de qualquer fenômeno como fenômeno de produção de significado e sentido. Seu campo de atuação é vasto; é matéria semiótica qualquer

signo produzido ou interpretado por nós, seres humanos, ou por outros animais, plantas, protozoários, fungos e bactérias, artefatos desenvolvidos por alguma entidade viva ou super-natural (Sebeok, 1994, p.6).

Na Semiótica Peirceana, o signo é apresentado como uma relação triádica entre o objeto – aquilo que é representado, o *representamen* – aquilo que representa e o interpretante – o processo de interpretação, conforme ilustra a Figura 3.11.



**FIGURA 3.11** - O SIGNO DE PEIRCE COMO UMA RELAÇÃO TRIÁDICA, EXEMPLIFICADO

O *representamen* representa o objeto, sob certos aspectos e capacidades; ele não é o objeto. O interpretante não é o intérprete do signo, mas sim um processo relacional criado na mente do intérprete. Peirce refere-se à “mente” como um conceito formal, não propriamente na acepção psicológica do termo. O *representamen* se coloca em uma relação triádica com seu objeto de modo a determinar que o interpretante assuma a mesma relação com o objeto. A relação triádica é genuína, no sentido de que seus 3 membros estão por ela ligados de modo a não consistir em nenhum complexo de relações diádicas (Peirce, 1974).

Da definição de signo para Peirce, decorre o conceito de semiose ilimitada – *semiosis*, ilustrado pela Figura 3.12. O interpretante é um processo de geração infinita de significações: aquilo que é um terceiro numa relação triádica, torna-se primeiro em outra relação triádica. O interpretante determinado por um objeto transforma-se em um *representamen* de um outro signo que remete a outro objeto, num processo que determina um novo interpretante e assim sucessivamente.



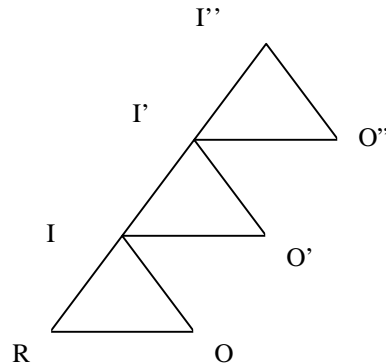


FIGURA 3.12 - O PROCESSO DE SEMIOSE ILIMITADA

Algumas coisas têm como razão primária funcionar como signo, por exemplo, letras, sons vocais, sistemas de computador, etc. Um sistema de reserva de vôos “está para” aviões, lugares, vôos, etc. A representação de um objeto e as conseqüentes interpretações dessa representação podem ser classificadas nas categorias icônica, indicial ou simbólica. **Representações icônicas** são baseadas nas semelhanças e características comuns ao objeto a que se referem; desenho de uma impressora na interface de determinado software é um ícone para a impressora real e a tarefa de imprimir. Representações que guardam a relação de causa e efeito entre objeto e *representamen* são chamadas índices; fumaça usada como *representamen* para fogo ou o desenho de uma ampulheta “significando” o correr do tempo são exemplos de **representações indiciais**. Representações baseadas em convenções estabelecidas são chamadas **símbolos**, a exemplo da linguagem natural e de formalismos lógico e matemático. Palavras reservadas em linguagens de programação são exemplos de símbolos. A Figura 3.13, adaptada de Nadin (1988) ilustra de forma operacional essa classificação dos signos.

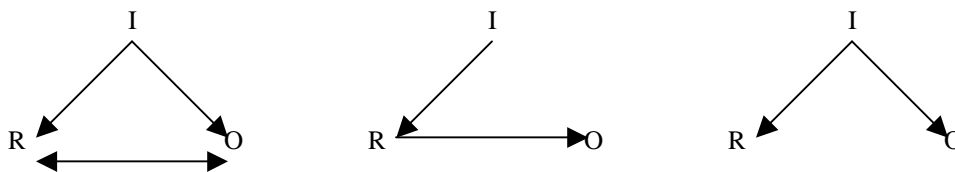




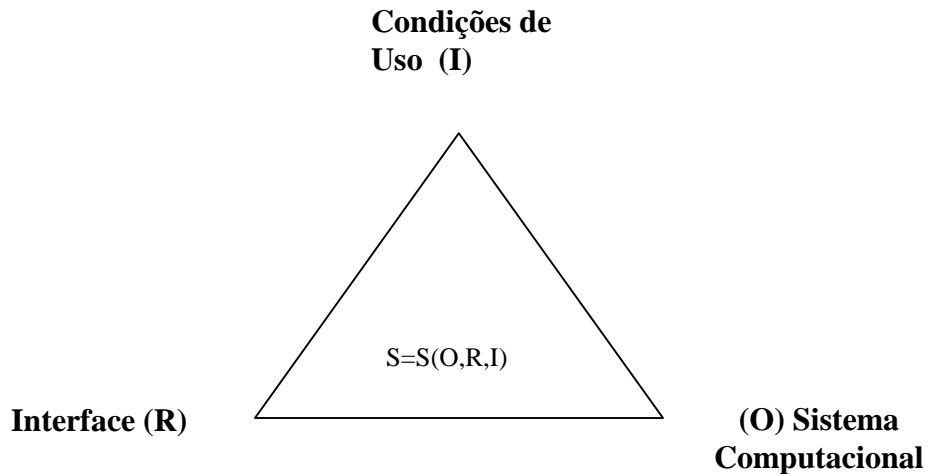
FIGURA 3.13 REPRESENTAÇÕES ICÔNICA, INDICIAL E SIMBÓLICA (ADAPTADO DE NADIN, 1988, p.55)

Como a tecnologia sobre a qual construímos interfaces muda muito rapidamente, princípios semióticos fornecem fundamentos para o design e avaliação de interfaces de forma mais compreensível. Por exemplo, o entendimento do que um ícone representa – e não propriamente o que ele “retrata” – é essencial no design da linguagem da interface. Consideremos, por exemplo, os sinais gráficos que popularmente chamamos de ícones na interface; na verdade não são todos ícones,

conforme conceito apresentado, mas muitos deles guardam uma relação simbólica com seus referentes no mundo. Dos vários exemplos que podemos encontrar nas interfaces gráficas, considere o botão de *forward* da ferramenta de *email* Eudora Light: . A informação codificada fisicamente para comunicar informação nesse botão estabelece uma relação simbólica (baseada em convenção) com seu referente – o programa *Forward*. Ao contrário, o sinal  (presente no botão de *forward* do *Netscape Mail*) estabelece uma relação icônica com seu referente, pois neste caso o conjunto de características do sinal e do referente no mundo têm intersecção não vazia.

Entre os esforços de trazer a Semiótica ao contexto do design de software, vários autores dentre eles Nadin (1988), Andersen (1990, 1997) e Souza (1993) fizeram contribuições seminais.

Nadin (1988) introduz uma das primeiras tentativas de aplicar a Semiótica ao design de interfaces, com base na teoria de Peirce. A Figura 3.14 mostra como a unidade básica da Semiótica, isto é, o conceito de signo, é entendido por Nadin no contexto do computador.



**FIGURA 3.14** - O CONCEITO DE SIGNO NO CONTEXTO DO COMPUTADOR (ADAPTADO DE NADIN, 1988, p.58)

Na proposta de Nadin, a interface do sistema – o *representamen*, ou aquilo que representa o objeto, torna “transparente” através das três representações possíveis (icônica, indicial, simbólica), qualquer tarefa ou ação a ser realizada com a utilização do computador. Segundo o mesmo autor, se no processo de constituição da unidade interface-aplicação-condições para uso e avaliação, representada na Figura 3.13, um dos elementos impõe requisitos que afetariam sua relação de reciprocidade, a

necessidade de melhorar o design do artefato deve ser reconhecida. São exemplos de condições onde essa relação é afetada: interfaces não suficientemente transparentes para as aplicações que representam, aplicações difíceis de usar ou inapropriadas à atividade desejada.

Na perspectiva de Nadin, o foco da atenção no design de software deve ser colocado na semiótica da comunicação de sua interface. Um escritório (real) não é uma coleção de arquivos, calculadoras, etc. – isto é, uma coleção de “objetos concretos”, mas um ambiente onde comunicação é necessária (troca de documentos, armazenamento e recuperação de dados, planejamento, etc.). Comunicação é entendida como a atividade semiótica que coloca usuário e designer juntos através de um intermediário que são as linguagens que eles usam. Designer e usuário, como parte de uma dada cultura, compartilham convenções estabelecidas e participam no estabelecimento de novos sistemas de signo, se necessário. Linguagens de programação são exemplos de sistemas semióticos que tornam possíveis novas ferramentas de natureza cognitiva (Nadin, 1988, p.70).

Andersen (1990, 1997) encontrou na escola européia criada por Saussure e desenvolvida por Hjelmslev, o substrato teórico que o levou a propor a “Semiótica Computacional” – uma aplicação da Semiótica não apenas ao design de interfaces, mas também à programação, análise e projeto de software.

A interface é definida por Andersen (1997, p.143) como uma coleção de signos baseados no computador, isto é, uma coleção das partes do software que podem ser vistas ou ouvidas, usadas e interpretadas por uma comunidade de usuários. Para Andersen, o design da interface deve emergir de padrões de uso, ou seja, da maneira como o usuário faz uso do “dialeto” baseado no computador. Design é visto como um processo iterativo no qual propostas são continuamente desenvolvidas, usadas e avaliadas. Em cada iteração desse processo de design, há um conjunto de signos para ser analisado. As relações entre as unidades constituintes da linguagem da interface são analisadas e como resultado modificações são propostas com o objetivo de adaptar o design dos signos da interface a padrões de uso do dialeto baseado no computador.

Os signos baseados no computador, diferentemente dos signos usados nas linguagens verbais, são transientes no sentido de que, ao longo do tempo, podem alterar suas características como cor, ou posição que ocupam na tela. Para acomodar essa característica, Andersen propõe que a análise de cadeias de signos baseadas no computador leve em conta cadeias concorrentes, isto é, aquelas compostas de signos e partes de signos que ocorrem juntos no mesmo ponto do tempo e cadeias sequenciais – cadeias ou partes de signos que ocorrem um após o outro em diferentes pontos no tempo. Exemplos do uso da abordagem de Andersen em análise, design e re-design de interfaces podem ser encontrados em Baranauskas *et al.* (1998), Prado e Baranauskas (1999) e Rossler (2000), entre outros.

Souza (1993) propõe a Engenharia Semiótica, para o design de linguagens de interface de usuário. Na Engenharia Semiótica (ES), a interface é entendida como um artefato de meta-comunicação, isto é, a interface é composta por mensagens enviadas do designer para o usuário e cada mensagem, por sua vez, pode enviar e receber mensagens do usuário. Nesse sentido, a interface cumpre dois papéis: (1) comunicar a funcionalidade da aplicação (o que a interface representa, que tipos de problemas está preparada para resolver) e o modelo de interação (como se pode resolver um problema); (2) possibilitar a troca de mensagens entre o usuário e a aplicação.

Na Engenharia Semiótica o foco está na comunicação unidirecional e indireta do designer para com os usuários. O designer cumpre um papel comunicativo explícito ao utilizar a interface para dizer algo ao usuário. Por sua vez, o usuário cumpre os papéis de agente da interação e de receptor da comunicação indireta do designer.

Em sua proposta original, a Engenharia Semiótica apoia-se na teoria da produção de signos de Eco (1997), para definir *guidelines* teoricamente motivadas para design de linguagens de interface de usuário. Atualmente um conjunto considerável de trabalhos estende e dá corpo à ES (Leite e Souza, 1999; Prates e Souza, 1999; Martins e Souza, 1998; Barbosa e Souza, 1999).

Um novo entendimento para o conceito de interface a partir de bases semióticas está sendo proposto por Oliveira (2000), onde a interface é entendida como um espaço de comunicação para entidades humanas e não-humanas (botões, heróis em jogos, janelas, etc.) que participam do “jogo semiótico” comunicando-se pela sua aparência e pela sua capacidade de produzir e interpretar signos. Nessa proposta, bases semióticas são aplicadas ao design das entidades propriamente ditas, sua consubstanciação na interface e ao design da comunicação entre elas (Oliveira e Baranauskas, 1999).

O desenvolvimento das teorias cognitivas em Interação Humano-Computador trouxe-nos uma visão do computador como ferramenta cognitiva que nos possibilita aumentar nossas capacidades de entendimento, memorização, tomada de decisão, etc. As abordagens semióticas ao design de software permitem-nos considerar, não apenas os aspectos imediatos (físicos) da interação com computadores, mas também seu aspecto interpessoal e cultural focando na expressão e interpretação dos elementos na interface do software. A visão da interação mediada por sistemas semióticos representa um paradigma relativamente recente para o design de software; resultados de pesquisa têm sido discutidos nos principais fóruns de IHC, no exterior, (por exemplo, CHI2000) e em nosso país (IHC99, IHC98, entre outros).

## REFERÊNCIAS:

Andersen, P.B.(1990, 1997) *A Theoy of Computer Semiotics. Semiotic Approaches to Construction and Assessment of Computer Sustems.* Cambridge: Cambridge University Press.

Andersen, P.B., Holmqvist, B. e Jensen, J.F.(1993) *The Computer as Medium*, Cambridge University Press.

Baecker, R. , Grudin, J., Buxton, W. e Greenberg, S. (eds.)(1995) *Readings in Human-Computer Interaction: Toward the Year 2000.* San Mateo, CA: Morgan Kaufmann.

Card, S.K., Moran, Newell, A.(1983) *The Psychology of Human-Computer Interaction*, Lawrence Erlbaum Associates

Baranauskas, M.C.C., Gomes Neto, N.G., Borges, M.A.F. (1999) Gamming at Work: a Learning Environment for Synchronized Manufacturing. *Proceedings of The International Conference on Engineering and Computer Education*, ICECE99 Rio de Janeiro, Brasil.

Baranauskas M C C, Rossler, F, Oliveira, O L (1998) Uma Abordagem Semiótica à Análise de Interfaces: um estudo de caso. Em *Atas do I Workshop sobre Fatores Humanos em Sistemas Computacionais*, Maringá, PR, p. 75-84.

Barbosa S D e Souza C S (1999) Making More Sense out of Users' Utterances. Em *Atas do II Workshop sobre Fatores Humanos em Sistemas Computacionais*, Campinas, SP, p.29

Boehm, B.W. (1995) A Spiral Model of Software Development and Enhancement. Em *Readings in Human-Computer Interaction: Toward the Year 2000*, R. Baecker, J. Grudin, W. Buxton, S. Greenberg, (eds.) San Mateo, CA: Morgan Kaufmann.

Brown, J.S., Duguid, P. (1992) Enacting Design for the Workplace. Em *Usability: Turning Technologies into Tools*, P. Adler, T. Winograd (eds). NY:Oxford University Press

Bruner, J.S. (1960) *The Process of Education*. London: Oxford University Press

Carrol, J.M. (1997) Scenario-Based Design. Em *Handbook of Human-Computer Interaction*, M.G.Helander, T.K.Landauer, P.V.Prabhu (eds) 2nd.edition, Elsevier.

CHI2000 <http://peirce.inf.puc-rio.br/chi2000ws6/>

Denning, P. Dargan, P. (1996) Action-Centered Design. Em *Bringing Design to Software*, T. Winograd (ed) Ma:Addidon-Wesley, pp. 105-127.

Eco, U. (1997) Tratado Geral de Semiótica. Editora Perspectiva, São Paulo, Brasil.

Ehn, P. (1988) *Work-Oriented Design of Computer Artifacts*. NJ: Lawrence Erlbaum Ass.

Erickson, T.D. (1990) Working with Interface Metaphors. Em B. K. Laurel (ed) *The Art of Human-Computer Interface Design*. Reading: Addison-Wesley Publishing Company.

Frohlich, D.M. (1997) Direct Manipulation and Other Lessons. Em *Handbook of Human-Computer Interaction*, M.G.Helander, T.K.Landauer, P.V.Prabhu (eds) 2nd.edition, Elsevier.

Good, M. (1995) "Participatory Design of a Portable Torque-Feedback Device." Em *Human-Computer Interaction: Toward the Year 2000*, R. Baecker, J. Grudin, W. Buxton, e S. Greenberg (eds) pp 293-303. Morgan Kaufmann Publishers, inc.

Gould, J.D., Boies, S.J., Ukelson, J. (1997) How to Design Usable Systems. Em *Handbook of Human-Computer Interaction*, M.G.Helander, T.K.Landauer, P.V.Prabhu (eds.) Elsevier Science, pp. 231-254

Gould, J.D., Lewis, (1985) Designing for Usability – key principles and what designers think. *Communications of the ACM*, 28,300-311

Greenbaum, J., King, M. (1991) *Design at Work*. Hillsdale, NJ:Lawrence Erlbaum Ass.

Grudin, J. (1995) Interactive Systems: Bridging the gaps between developers and users. Em *Human-Computer Interaction: Toward the Year 2000*, R. Baecker, J. Grudin, W. Buxton, e S. Greenberg (eds) pp 293-303. Morgan Kaufmann Publishers, Inc.

Hix ,D.e Hartson, H.R.(1993) *Developing User Interfaces: Ensuring Usability through Product and Process*. New York: John Wiley.

Hutchins, E.L., Hollan, J.D., Norman, D.A. (1986) Direct Manipulation Interfaces. Em *User Centered System Design: New Perspectives on Human-Computer Interaction*, D.A. Norman, e S.W. Draper, (eds.) (1986) Hillsdale, NJ: Lawrence Erlbaum Associate Publishers.

IHC98 Atas do I Workshop sobre Fatores Humanos em Sistemas Computacionais  
<http://www.inf.puc-rio.br/~ihc98>

IHC99 *Atas do II Workshop sobre Fatores Humanos em Sistemas Computacionais*  
<http://www.unicamp.br/~ihc99>

Kuhn, S., Winograd, T. (1996) Participatory Design. Em *Bringing Design to Software*, Ma:Addison-Wesley.

Laurel, B. (ed) (1990) *The Art of Human-Computer Interface Design*, Reading, Mass.:Addison-Wesley.

Lakoff, G. e Johnson, M.(1980) *Metaphors we Live By*. Chicago:University of Chicago Press.

Leite J. e Souza C.S. (1999) Uma Linguagem de Especificação para a Engenharia Semiótica de Interfaces de Usuário. Em *Atas do II Workshop sobre Fatores Humanos em Sistemas Computacionais*, Campinas, SP, p.23

Madsen, K.H. (1994) A Guide to Metaphorical Design. *Communications of the ACM* vol.37, n.12, pp. 57-62.

Martins I.H. e Souza C.S. (1998) Uma Abordagem Semiótica na Utilização dos Recursos Visuais em Linguagens de Interface. Em *Atas do I Workshop sobre Fatores Humanos em Sistemas Computacionais*, Maringá, PR, p. 38-47.

Mazzone, J. (2000). Comunicação Pessoal em Relatório Projeto Fapesp (documento Interno do NIED – Unicamp)

Muller, M. (1997) Participatory Practices in the Software Lifecycle. Em *Handbook of Human-Computer Interaction*, M.G.Helander, T.K.Landauer, P.V.Prabhu (eds.). Elsevier Science, pp. 255-297

Nadin M.(1988) Interface Design. *Semiótica*, v.69, n.3/4, p.269-302.

Nardi, B. (1997) The Use of Ethnographic Methods in Design and Evaluation. Em *Handbook of Human-Computer Interaction*, M.G.Helander, T.K.Landauer, P.V.Prabhu (eds.) Elsevier Science, pp. 361-366

Neale, D.C., Carroll, J.M. (1997) The Role of Metaphors in User Interface Design. Em *Handbook of Human-Computer Interaction*, M. Helander, T.K. Landauer e P. Prabhu (eds.) (1997) chapter 20, Elsevier Science pp. 441-462.

Nied (2000) *Jogo do Alvo*. <http://www.nied.unicamp.br>. Consulta em 3/4/2000.

Nielsen, J. (1992) Usability Engineering, *Computer*, March.

- Nielsen, J. ,(1993) *Usability Engineering*. Boston: Academic Press.
- Norman, D. (1993) *Things that Make us Smart* Reading, Ma:Addison Wesley
- Norman, D. (1996) Design as Practiced. Em *Bringing Design to Software* T. Winograd (ed) Ma:Addison-Wesley p. 233 - 251.
- Norman, D.A. e Draper, S.W. (eds.) (1986) *User Centered System Design: New Perspectives on Human-Computer Interaction* Hillsdale, NJ: Lawrence Erlbaum Associate Publishers.
- Oliveira, O L. e Baranauskas, M.C.C. (1998<sup>a</sup>) A semiótica e o Design de Software. Relatório técnico IC98-09 (disponível via Web: <http://www.dcc.unicamp.br/ic-tr-ftp/1998/Titles.htm>)
- Oliveira, O L e Baranauskas, M C C (1998b) Semiotic Proposals for Software Design: Problems and Prospects. Relatório técnico IC98-10 (disponível via Web: <http://www.dcc.unicamp.br/ic-tr-ftp/1998/Titles.htm>)
- Oliveira OL. e Baranauskas M.C.C. (1999) Communicating Entities: a Semiotic-Based Methodology for Interface Design. Em *Human-Computer Interaction – Ergonomics and User Interface*, H.J. Bullinger e J. Ziegler (eds) vol.1. London: Lawrence Erlbaum Associates Publishers.
- Oliveira, O L (2000) *Design da Interação em Ambientes Virtuais: uma abordagem semiótica*. Tese de Doutorado. Instituto de Computação – Unicamp.
- Peirce, C.S. (1974) *Collected Papers of Charles Sanders Peirce*. Vols. 1-6. C. Harshorne e P. Weiss (eds). Cambridge:Harvard University Press.
- Prates e Souza (1999) Um Modelo de Apoio à Expressão de Projetistas de Interfaces Multiusuário. Em *Atas do II Workshop sobre Fatores Humanos em Sistemas Computacionais*, Campinas, SP, p.21
- Prado e Baranauskas (1999) Projeto Granel: Investigando possibilidades da abordagem Semiótica em Design de Interfaces. Em *Atas do II Workshop sobre Fatores Humanos em Sistemas Computacionais*, Campinas, SP, p.17
- Preece, J., Rogers, Sharp, H., Benyon, D., Holland, S., Carey, T. (1994) cap. 18 em *Human-Computer Interaction - Methods for User-Centred Design*, Addison Wesley, pp. 371-382.



Rossler, F (2000) *Contribuições da Semiótica ao Redesign de Interfaces para Ferramentas de Comunicação Eletrônica*. Dissertação de Mestrado. Instituto de Computação-Unicamp.

Santaella, M.L. (1996) O que é Semiótica. 12.ed São Paulo:Editora Brasiliense.

Schuler, D. e Namioka, A.(eds.) (1993) *Participatory Design: Principles and Practices*, Hillsdale,NJ: Lawrence Erlbaum Ass..

Sebeok, T.A. (1994) *Signs – An Introduction to Semiotics*. Toronto:University of Toronto Press Incorporated.

Shame (1999) *Hall of Shame*. <http://www.iarchitect.com/mshame.htm>. Consulta em 3/4/2000

Shneiderman, B. (1998) *Designing the User Interface*. Reading,MA: Addison-Wesley.

Shneiderman B. (1983) Direct manipulation: a step beyond programming languages, *IEEE Computer*, 16(8),57-69.

Simon (1982) *The Sciences of the Artificial*. Cambridge, MA:MIT Press.

Souza, C.S.,(1993) The Semiotic Engineering of user interface Languages, *International Journal of Man-Machine Studies*, n. 39, 753-733

Suchman, L. (1987) *Plans and Situated Actions*. Cambridge:Cambridge University Press.

Suchman, L. Trigg, R.H.(1995) Understanding Practice: Video as a Medium for Reflection and Design. Em *Human-Computer Interaction: Toward the Year 2000* R. Baecker, J. Grudin, W. Buxton, e S. Greenberg (eds) pp 293-303. Morgan Kaufmann Publishers, Inc.

Winograd, T.,(1996) *Bringing Design to Software*, Ma:Addison-Wesley.





