2
Trabalhos Relacionados e Fundamentação Teórica

Neste capítulo apresentamos o estado da arte dos sistemas extensíveis e alguns conceitos básicos de EngSem. Na Seção 2.1 descrevemos os conceitos e definições dos sistemas extensíveis segundo Dieterich et al. (1993) e Assis (2005), assim como as técnicas de sistemas extensíveis que iremos tratar neste trabalho, com base em *End User Development*. No final desta seção apresentamos uma classificação dos mecanismos de extensão apresentados na literatura. Na Seção 2.2 apresentamos a Engenharia Semiótica, descrevendo alguns dos seus principais conceitos, além de sua contribuição para EUD. Na Seção 2.3, apresentamos como geralmente é feita a análise de usuários e tarefas segundo a proposta de Hackos & Redish (1998), utilizada como base para nossa proposta.

2.1.
Sistemas Extensíveis

Sistemas extensíveis são desenvolvidos de maneira que possam evoluir com o tempo. Mais precisamente, esses sistemas podem ser configurados pelos usuários finais, adicionando ou removendo funcionalidades ou modificando as existentes.

Segundo Sutcliffe (2003), há três dimensões em sistemas extensíveis nas quais podemos classificar o design (Figura 1). Uma delas se refere ao **escopo** do domínio da aplicação. Alguns ambientes EUD podem ser completamente **específicos** ao domínio e às tarefas, enquanto outros são mais **gerais**. Outra dimensão trata da **representação** e classifica a comunicação com os usuários. Ela pode ser realizada de forma natural, e por isto **concreta**, ou não natural, **abstrata**. A terceira dimensão é formada pela **iniciativa**, quando o usuário participa ativamente, tomando decisões sobre as extensões ou quando o usuário participa passivamente, deixando que o sistema tome as decisões e execute as tarefas por conta própria e a partir da observação sobre o usuário.

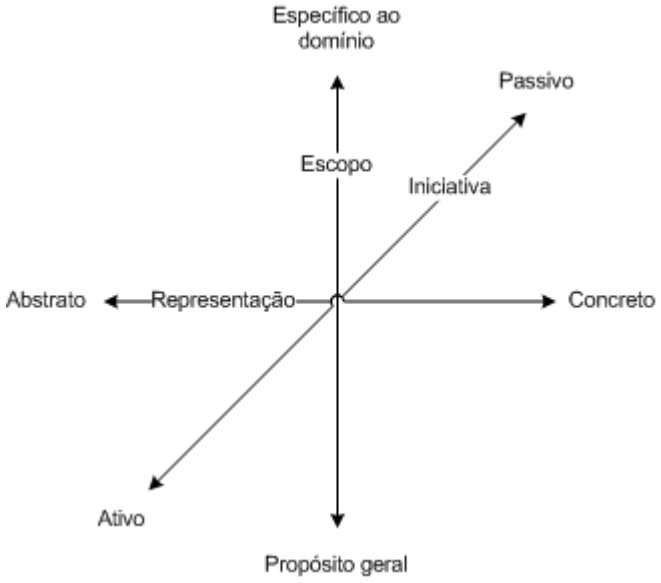


Figura 1: Dimensões de EUD

Há dois tipos de usuários, os especialistas no domínio e os novatos. Independente de qual categoria um usuário possa se encaixar, eles podem conhecer ou não as técnicas e linguagens de programação. Quando tratamos de criar ambientes para usuários finais ou sistemas que possam ser, de alguma forma, extensíveis, podemos ter dois tipos de atividades:

(1) As que permitem que os usuários ativem alguns parâmetros, escolham entre alternativas de comportamentos (ou apresentações ou mecanismos de interação) que existem na aplicação.

(2) As que implicam modificação através de algum paradigma de programação, criando ou modificando o artefato de software.

Antes de se tomar a decisão sobre o quê adaptar e como, é necessário analisar o projeto de desenvolvimento da aplicação sob diversas perspectivas. Dieterich et al. (1993) apresentam o espaço de design para adaptar interfaces dividido em cinco etapas.

Etapa 1: Estágios e agentes no processo de adaptação

A adaptação pode ser controlada de diversas formas. Segundo os autores, há quatro estágios de adaptação de uma tarefa. A **iniciativa** (*initiative*) da adaptação é a decisão do agente (que pode ser o sistema, representando o designer ou o próprio usuário) em sugerir a adaptação. Subseqüentemente, são **propostas alternativas** (*proposal*) de adaptação. Depois, deve tomar a **decisão** (*decision*) sobre qual alternativa escolher para que ocorra a sua **execução** (*execution*).

Seguindo esses estágios, a adaptação pode ser classificada como mostra a figura a seguir.

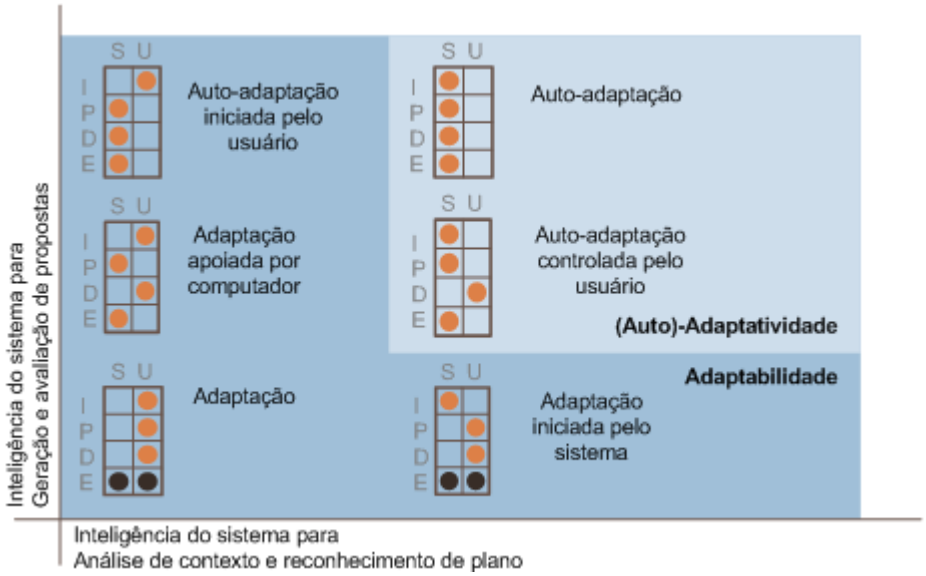
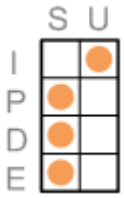


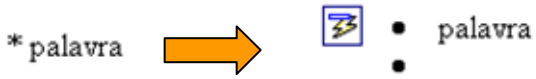
Figura 2: Classificação segundo os estágios da adaptação e o agente

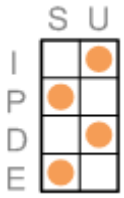
[Legenda: S - sistema / U – Usuário / I – iniciativa / P – proposta / D – decisão / E – execução]

A seguir, exemplificamos cada tipo de adaptação definido pelos autores.

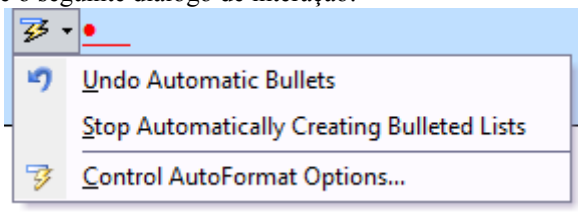


Auto-adaptação iniciada pelo usuário
Quando um usuário do Microsoft Word digita '*' seguindo de espaço e uma palavra, o sistema automaticamente transforma a seqüência "* palavra" em uma lista.





Adaptação apoiada por computador
Seguindo o exemplo anterior, quando o sistema executa a adaptação, aparece o seguinte diálogo de interação:



Neste caso, o sistema propõe alternativas de adaptação e o usuário pode selecionar uma opção, deixando que o sistema execute a alternativa escolhida.

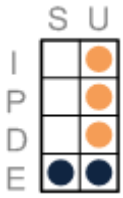
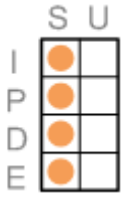
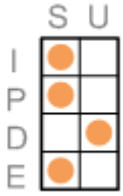
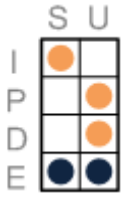
	**Adaptação** Na gravação de macro, o usuário toma a iniciativa sobre a adaptação, mostra como as interações devem ocorrer e o sistema a executa. Num primeiro momento o próprio usuário executa a adaptação, e posteriormente delega a execução para o sistema.
	**Auto-adaptação** [Exemplo fictício] Um celular (sistema) é capaz de reconhecer o ambiente em que o usuário se encontra através da rede. Sabendo que ele está em seu escritório, ele automaticamente altera o tom do celular para um tom baixo. Percebendo que o usuário está em casa, ele aumenta o tom.
	**Auto-adaptação controlada pelo usuário** O Eager (Cypher, 1991) apóia o usuário na execução de tarefas repetitivas. Ele constantemente observa as ações do usuário e, quando detecta uma atividade repetitiva, ele escreve um programa que executa a tarefa para o usuário. Porém, o usuário faz a escolha se deseja passar o comando da execução das tarefas para o Eager.
	**Adaptação iniciada pelo sistema** O sistema detecta que o usuário sempre muda a impressora sob diversas circunstâncias e sugere que ele vá ao Painel de Controle alterar a impressora *default* do sistema.

Tabela 1: Esquema de classificação com exemplos: tarefas e agentes envolvidos no processo de adaptação

Etapa 2: Constituintes adaptados

Dieterich et al. (1993) dividem os constituintes que podem ser adaptados em dois grupos: os relacionados a adaptação da comunicação e os relacionados a adaptação da funcionalidade. No primeiro grupo, os usuários se restringem a realizar as tarefas propostas pelo designer, porém podem escolher como desejam interagir com o sistema. No segundo, o usuário pode associar funcionalidades novas ou até mesmo mais complexas ao sistema.

Alguns sistemas apresentam determinada habilidade para correção de erros, enquanto outros possuem um sistema de *help* ativo. Ambos tentam detectar erros dos usuários e fornecem apoio para a recuperação dos mesmos. Estes constituintes são definidos como funções genéricas, propostas pelo designer para apoiar o usuário durante a interação.

Do ponto de vista da adaptação da apresentação, alguns sistemas apresentam diferentes estilos de interação de acordo com o usuário (especialistas, intermediários ou novatos). Algumas soluções são as opções de "*Query and*

Answer" (perguntas que são feitas ao usuário, pelo sistema, para classificar o nível de conhecimento do usuário), seleção de menus e linguagem de comando.

As adaptações de funcionalidade, mais complexas, permitem que os usuários realizem suas tarefas e aloquem a execução das rotinas no sistema. Alguns sistemas, porém, permitem que as tarefas sejam dinamicamente alocadas tanto pelos usuários quanto pelo sistema. Outros podem realizar tarefas completas, propor soluções ou deixar que a execução seja realizada pelos usuários. A simplificação das tarefas pode ser realizada através de gravação de macros. Os autores relacionam os constituintes de acordo com os níveis que uma adaptação pode ter.

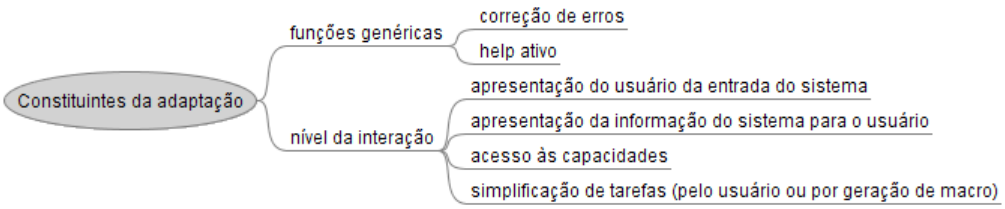


Figura 3: Constituintes da adaptação

No Microsoft® Word® 2003 o usuário pode alterar diversas opções fornecidas pelo designer. Na Figura 4 podemos ver exemplos de alteração da **apresentação de informações do sistema** e da **apresentação de entrada de dados**.

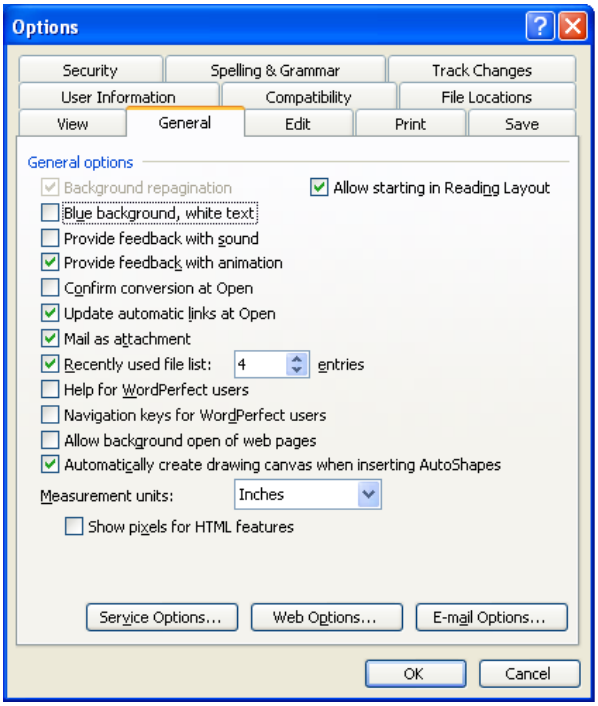


Figura 4: Microsoft® Word® 2003 Tools Options

Desta forma, consideramos como sendo os **constituintes da adaptação** as respostas à pergunta "o que deve ser adaptado?".

Etapa 3: Informações a serem consideradas para a adaptação

Alguns sistemas possuem a característica de se adaptarem às diferenças existentes entre os usuários, considerando a cognição e a personalidade. Além disto, pode ser necessário considerar aplicações para um tipo *default* de usuário, grupos de usuários ou um tipo individual. É importante também pensar nas regras de ergonomia para um sistema, além da interface propriamente dita, assim como restrições de design. Isto deve ser feito para que a consistência da interface seja garantida.

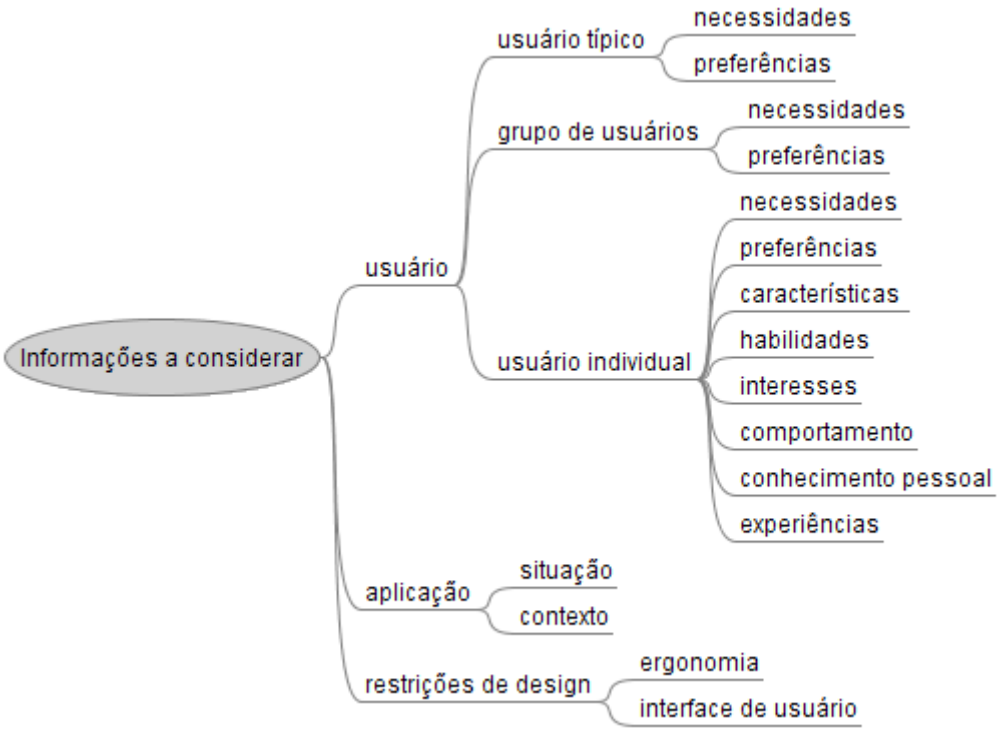


Figura 5: Informações a considerar sobre adaptações

Desta forma, os autores propõem considerar as informações mostradas na Figura 5.

O pacote Microsoft® Office® 2003 é formado por diversos sistemas como editor de textos, editor de apresentações, gerenciador de e-mails etc. Tendo em vista as diversas **necessidades** e as **preferências** dos usuários, o designer optou por deixar a decisão sobre qual sistema deve ser instalado por conta do usuário. Outra informação considerada pelo designer para a tomada desta decisão foi o **contexto da aplicação**. Este pacote pode ser usado por profissionais, usuários

domésticos, estudantes e outros. Pode ser o caso de um estudante precisar de um editor de texto e um editor de apresentações, e um usuário doméstico, num primeiro momento, precisar apenas de um gerenciador de e-mails. Posteriormente, este usuário doméstico pode desejar instalar um editor de apresentações para poder visualizar alguns arquivos recebidos por e-mail. Em nossa proposta, consideramos essas informações como respostas à pergunta "em função do que adaptar?".

Etapa 4: Objetivos da adaptação

Dentre os objetivos de uma adaptação, o principal é apresentar ao usuário uma interface fácil de usar, eficiente e efetiva, caso contrário todo o intuito de tornar adaptável o sistema pode ser inválido. Desta forma, a interface deve ser adequada para grupos heterogêneos e que considere o crescimento da experiência do usuário.

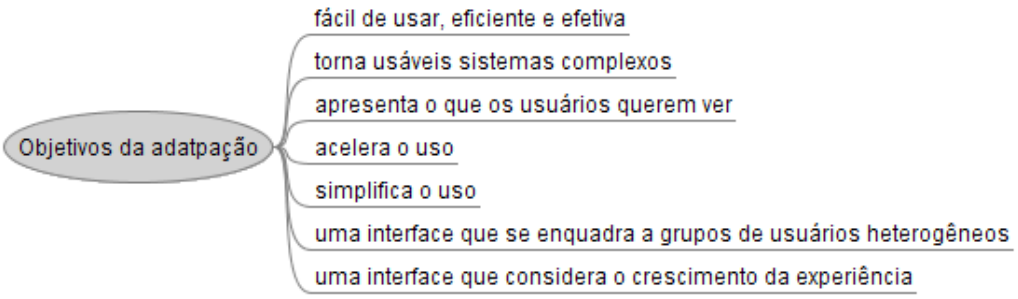


Figura 6: Objetivos da adaptação

Em se tratando de um sistema adaptativo, uma interface fácil de usar pode englobar conceitos de adaptação de conteúdo, onde o sistema decide o que mostrar ao usuário.

Um exemplo típico desta adaptação é o help ativo. Versões diferentes de help podem ser criadas pelo designer de acordo com os diferentes tipos de usuários. Para os usuários novatos, até mesmo a forma de descrição da ajuda deve ser cuidadosamente elaborada. Para esses usuários, mais ajuda pode ser fornecida, enquanto que descrições pontuais ou mais formais podem ser mostradas a usuários mais experientes. Na medida em que o usuário for aprendendo com o sistema, ele pode se tornar um usuário experiente. Ao notar isto, o sistema automaticamente enquadra o sistema de ajuda ao perfil do usuário.

Voltando ao exemplo do *Tools/Options* do Microsoft® Word® 2003 (Figura 4), podemos citar como objetivos para a tomada de decisão sobre essa extensão: apresentar o que o usuário deseja ver e eficiência.

Esses objetivos são considerados, na nossa proposta, como resposta à pergunta "por que adaptar?".

Etapa 5: Estratégias da adaptação

Das informações necessárias sobre as estratégias da adaptação, devemos determinar quando o sistema ou usuário devem interagir. Os autores definem isto como o *timing* da adaptação.

A adaptação pode ocorrer antes do uso do sistema, durante sessões ou entre as sessões. A adaptação que ocorre antes do uso do sistema deve ser considerada pelo designer de acordo com as necessidades de um usuário típico, ou até mesmo um grupo de usuários. Assim, as necessidades de um usuário único não devem ser levadas em conta neste aspecto, porém o sistema pode fornecer a possibilidade ao usuário de alterar suas configurações antes do uso do sistema. A adaptação durante o uso, a mais interessante das adaptações, ocorre continuamente. Ela é a mais importante porque possui as maiores possibilidades de se adequar às necessidades dos usuários. Porém, deve-se tomar o cuidado para que não ocorra uma adaptação regressiva, e o usuário se confunda pela mudança da interface exatamente no momento em que ele estiver "entendendo" o sistema. A adaptação entre as sessões permite estratégias complicadas de adaptação. Ela se baseia nas necessidades do usuário da última sessão. O problema deste tipo de adaptação ocorre quando o usuário demora a usá-la novamente, e assim deve realizar novo trabalho de aprendizado.

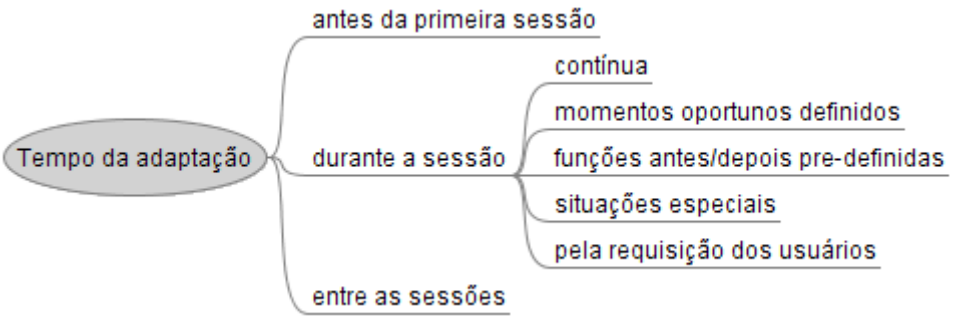


Figura 7: Tempo das adaptações

Quando um usuário usa o Microsoft® Word® 2003, a menos que ele tenha configurado para que o sistema não realize esse processo, o sistema cria um *bullet*

sempre que o usuário realiza a seqüência de ações em uma nova linha: do documento 1) digitar *; 2) acionar a tecla espaço; 3) digitar algum texto; e 4) acionar a tecla enter.Este exemplo mostra uma adaptação contínua para situações especiais (deve ser realizada uma seqüência de ações).

Na nossa proposta, as questões relacionadas às estratégias de adaptação são endereçadas pelas perguntas como adaptar, quando a adaptação inicia, até quando fica em vigor e ao quê a adaptação está vinculada.

***

Os autores tratam ainda dos diversos modelos que podem compor o projeto de desenvolvimento para aplicações extensíveis. A seguir mostramos algumas possíveis funções de adaptação e como elas são refletidas pelos componentes de software e pela arquitetura da interface relacionados.

O modelo estrutural da interface pode ser descrito como um contêiner de modelos elementares, com foco nos variados aspectos funcionais relevantes para a interação adaptativa. Os modelos podem ser representados implicitamente através da interface ou explicitamente como uma base de conhecimento ou parte dela. Eles podem estar distribuídos por toda a interface ou centralizados em componentes separados.

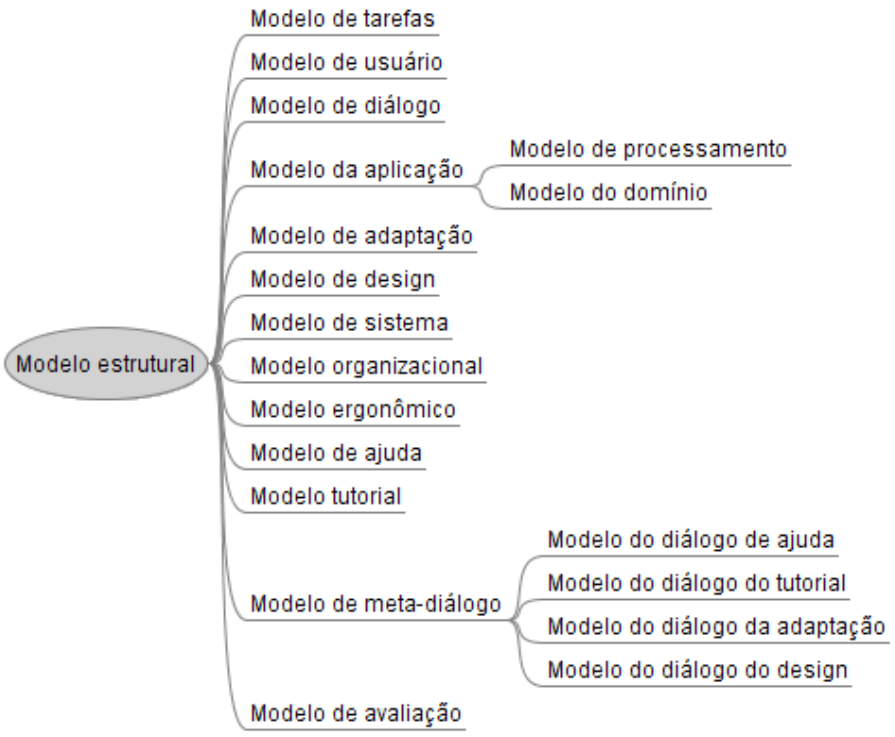


Figura 8: Modelo estrutural

O objetivo do nosso trabalho não é dizer como o mecanismo de adaptação deve ser implementado, isto é, qual o modelo ele deve usar para o gerenciamento das informações das extensões, mas apoiar o designer em sua tomada de decisão sobre qual mecanismo de adaptação ele pode usar em um caso específico.

Precisamos saber como encaixar os componentes dos modelos mostrados até aqui para que eles, juntos, componham a aplicação. A primeira coisa a se pensar é: quem são os usuários? Precisamos saber de características relevantes dos usuários para que o sistema extensível possa fornecer interfaces individualizadas. Tomando como base o modelo do usuário, podemos distinguir as seguintes características (Figura 9).

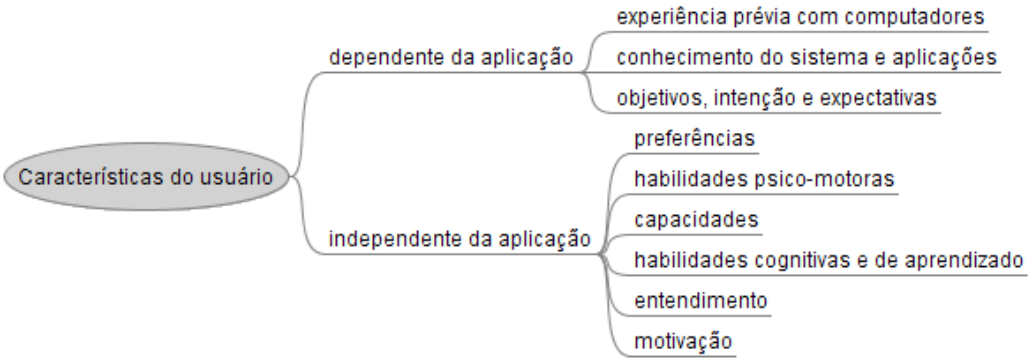


Figura 9: Características do usuário

Como será visto na Seção 2.3.2, Hackos & Redish (1998) propõem utilizar algumas perguntas de análise de usuários, tarefas, e ambiente de trabalho. Estas questões estão relacionadas a modelagem dos usuários e das tarefas descrita por Dieterich e co-autores. No próximo capítulo discutiremos quais perguntas devem ser feitas na análise para que o designer possa tomar as decisões sobre adaptação.

***

Assis (2005) propõe uma arquitetura de propósito geral para adaptação e meta-adaptação de sistemas hipermídia (*Adaptive Hipermedia Systems* - AHSs). Em seu trabalho, ela faz um levantamento das técnicas de adaptações reconhecidas por alguns autores de sistemas hipermídia. A partir dessas técnicas ela utiliza diversas perguntas para caracterizar adaptações em AHSs: “o quê” é adaptado; “em função de quê” a adaptação é feita; “como” ela é feita e “quando” (em que granularidade de tempo) ela ocorre (Assis, 2005: 23). Esta última dimensão é, na verdade, uma resposta a “em função de quê” tempo a adaptação.

O quê é adaptado?

As adaptações em hipermídia adaptativa podem modificar o conteúdo, a navegação e a apresentação (Koch, 2000 *apud* Assis, 2005).

- Adaptação de conteúdo (seleção diferenciada de informação, pela utilização, por exemplo, de conteúdos alternativos e adição ou ocultação de conteúdo);
- Adaptação de navegação (mudanças nas aparências dos elos, nos destinos dos elos, no número de elos ou na ordem em que estes elos são apresentados ao usuário);
- Adaptação de apresentação (design diferente dos elementos de interface tais como: tipo de mídia; ordenação; cores; tipo da fonte; tamanho das imagens).

Com o objetivo de caracterizar mais precisamente o quê pode ser adaptado – ou seja, qual o resultado real da adaptação, qual aspecto da aplicação muda –, Assis faz uma diferenciação entre a adaptação do teor do conteúdo e a adaptação da estrutura do conteúdo, esta última considerada como um tipo de adaptação de navegação. Ela distingue, ainda, um outro tipo de adaptação de navegação: a adaptação da topologia do hiperespaço (alterações nos destinos dos elos ou na quantidade de elos). Em relação à apresentação (interface), ela separa as alterações na aparência do conteúdo das alterações na aparência dos elos. A Tabela 2 a seguir apresenta a classificação proposta.

O quê é adaptado		**Descrição e exemplos**
Conteúdo		O conteúdo propriamente dito. > Exemplos: texto coloquial para leigos versus texto técnico para especialistas; textos simples para novatos versus textos detalhados para usuários experientes.
Navegação	Estrutura do conteúdo	Nós – a maneira pela qual os conceitos são colocados juntos para propósitos de navegação. > Exemplos: inclusão da introdução para novatos versus pular introdução para usuários avançados; inclusão de advertências de segurança da primeira vez que um conteúdo crítico é visitado versus a não inclusão e visitas subseqüentes.
	Topologia do hiperespaço	Elos – alterações nas âncoras e índices modificam os caminhos de navegação. > Exemplos: inclusão de um elo para a "solução" de um problema utilizado como exemplo versus a não inclusão se o problema for utilizado como teste.
Apresentação	Interface do conteúdo	Fragmentos. > Exemplos: utilização de fontes de tamanhos diferentes de acordo com a idade do usuário; emprego de realce para enfatizar certos tipos
	Interface de suporte à	Âncoras. > Exemplos: utilização de menu drop-down versus utilização de

	navegação	lista explícita de âncoras; uso de âncoras textuais versus uso de ícones como âncoras.

Tabela 2: Tabela de descrição e exemplos para a pergunta "O quê é adaptado" (Assis, 2005)

Em função de quê é feita a adaptação?

Esta questão diz respeito aos vários parâmetros que podem ser usados para determinar a adaptação. A Tabela 3 detalha os aspectos básicos que podem ser levados em consideração na adaptação.

Aspecto	Explicação e exemplo
Modelo de domínio	A maneira como o conteúdo é estruturado no DM pode afetar a adaptação. > Exemplo: um conceito (hierarquicamente) composto ser considerado como aprendido somente quando todos os conceitos elementares que o compõem tiverem sido aprendidos.
Perfil do usuário (armazenado no UM)	Preferências e características do usuário; papel desempenhado por ele. > Exemplos: se o usuário é novato ou experiente; se ele prefere textos ou imagens; em que língua ele prefere acessar; se ele é estudante ou professor; se ele se identificou ou está acessando como anônimo; se ele gosta mais de acessar textos inteiros ou resumos; conhecimento dele sobre o domínio; objetivos dele.
Contexto (Ambiente) [1] (Situação de uso)	Sob que condições o sistema é utilizado. > Exemplos: se o usuário está navegando a partir de um computador pessoal ou a partir de um telefone celular; de qual lugar (tais como: rua, bairro, cidade, casa, universidade, escritório) o acesso está sendo feito; qual a banda de acesso.
Histórico de Navegação	Caminho que o usuário percorre enquanto navega. > Exemplo: nós de navegação acessados previamente.
Histórico de Interação	As ações que o usuário executa (interações em nós de navegação) enquanto navega. > Exemplos: quais artigos o usuário só lê e quais imprime; quantas vezes ele vê um vídeo; que imagens ele amplia.
Utilização da Funcionalidade da Aplicação	Como a funcionalidade da aplicação está sendo usada. > Exemplos: resultados de testes; itens comprados previamente.
Granularidade de Tempo	A adaptação é sempre feita em algum instante ou período de tempo. A questão é em que granularidade este tempo é considerado, podendo ir desde uma adaptação pontual até episódica até contínua. Exemplos: adaptação feita no login (customização[2]); adaptação após uma configuração do usuário; revisão e adaptação do sistema a cada passo da navegação.

Tabela 3: Tabela de descrição e exemplos para a pergunta "Em função de quê é feita a adaptação" (Assis, 2005)

[1] Neste caso, "contexto" é empregado no sentido de situação / ambiente de uso e não no sentido mais amplo que engloba, além do dispositivo, o perfil, navegação, localização, etc.

[2] Neologismo largamente utilizado no sentido de "personalizar", "fazer sob encomenda".

Como é feita a adaptação?

Ela trata esta questão levando em conta quatro mecanismos: o modelo de usuário; o modelo de adaptação; o agente de entrada (*input*); e o modelo de atualização (*update*). O modelo de adaptação pode usar regras onde as pré-condições definem o "em função de que" e as pós-condições definem "o que" é alterado/adaptado. A captura e atualização dos dados podem ser automáticas ou manuais, feitas pelo usuário ou pelo autor. A informação sobre o usuário pode ser obtida por observação ou o usuário pode preencher um formulário.

Quando é feita a adaptação?

Algumas características do usuário podem ser determinadas antes da utilização do sistema. A adaptação feita em função de tais características é denominada customização. Neste caso, as alterações ocorrem antes do efetivo uso do sistema pelo usuário e, uma vez iniciada a sua utilização, a aplicação permanece inalterada. Considera-se que a adaptação propriamente dita acontece quando as características influenciam dinamicamente as alterações no sistema.

A granularidade do tempo, ou seja, "em função de que" tempo a adaptação ocorre, pode ser um parâmetro usado para determinar a adaptação.

O trabalho de Assis se concentra em sistemas hipermídia adaptativos. No capitulo 3, revemos as questões por ela propostas para tratar de sistemas extensíveis em geral.

2.1.1. Técnicas de sistemas extensíveis

Uma área de pesquisa relacionada a aplicações extensíveis para pessoas *experts* no domínio, mas que não são profissionais de programação, é a de desenvolvimento por usuários finais (*End-User Development* - EUD). EUD pode ser entendida como uma aplicação do tipo "faça você mesmo". Ou seja, o sistema fornece apoio para que o usuário final, normalmente um não programador, realize tarefas comumente designadas a um programador.

A descrição mais completa sobre o que EUD constitui é dada por Lieberman et al. (2006). Ela pode ser traduzida da seguinte forma:

> EUD pode ser definido como um conjunto de métodos, técnicas e ferramentas que permitem, em certo momento, aos usuários de sistemas de *software* - que estão agindo como desenvolvedores não profissionais de *software* - criar, modificar ou estender um artefato de *software*. (p. 2)

De forma geral, podemos dividir os tipos de EUD de acordo com o usuário final que irá utilizar a aplicação. Podemos ter usuários iniciantes ou experientes. Para auxiliar os usuários iniciantes, normalmente são usadas algumas técnicas como linguagens de programação visual ou programação por exemplos (*Programming by Example – PbE*) ou demonstração (*Programming by Demonstration - PbD*).

O ToonTalk™ (Kahn, 1996) é um programa criado com o intuito de auxiliar crianças na tarefa de programação. A apresentação do sistema é feita através de caracteres animados, incluindo robôs que podem ser treinados por exemplos.

Um programa feito pelo ToonTalk™ é uma seqüência de regras, onde cada regra tem uma cabeça e um corpo. A cabeça é um padrão que pode ser combinado com o argumento, que deve necessariamente ser uma tupla. Na apresentação do programa, a regra é apresentada através do robô, um programa como um time de robôs e uma tupla por uma caixa que pode ter qualquer numero de compartimentos nos quais os objetos podem ser armazenados. Um processo é uma caixa com um time de robôs, trabalhando nela.

A seguir mostramos as "concretizações" de abstrações computacionais do sistema. Para cada abstração computacional o ToonTalk™ fornece um equivalente análogo tangível.

Abstração Computacional	Concretização no ToonTalk™
Computação	Cidade
Ator ou processo ou objeto	Casa
Métodos	Robôs
Pré-condições de método	Conteúdo da mente do robô
Ações de métodos	Ações ensinadas aos robôs
"Tuples" ou mensagens ou vetores	Caixas
Testes de comparação	Escalas (réguas)
Geração de ator	Caminhões carregados
Término de ator	Bombas
Constantes	Números, textos e imagens
Capacidade de transmitir canais	Pombos
Capacidade de receber canais	Ninhos
Armazenagem de programas	Agendas

Tabela 4: Relação entre as abstrações computacionais e as concretizações no ToonTalk™

Na Figura 10 mostramos uma tela do ToonTalk™. Nela, o usuário pegou um pombo e Mário, um robô, está falando sobre o que os pombos fazem.

Figura 10: ToonTalk – programação por exemplos para crianças

Outras técnicas permitem que a interação ocorra de forma visual, em vez de redação, e normalmente é realizada através da manipulação direta. A Figura 11 mostra o exemplo do ambiente de programação Alice, desenvolvido para crianças. Ele utiliza técnicas de programação via menu e *drag-and-drop* de elementos visuais e textuais para facilitar a composição de códigos.

Figura 11: Alice[3] (Pausch et al. 1995): ambiente de programação desenvolvido para crianças

[3] Alice <http;//www.alice.org> - última visita em 05/03/2008

Os usuários experientes, que normalmente são profissionais de áreas diferentes da Computação, porém conhecedores do domínio buscam o auxílio de algumas ferramentas computacionais para a realização de seu trabalho. Eles normalmente possuem uma bagagem de conhecimento computacional, porém não possuem muito tempo para gastar aprendendo a programar. Para estes, são fornecidas funcionalidades como o auxilio a criação de scripts e a escrita de códigos.

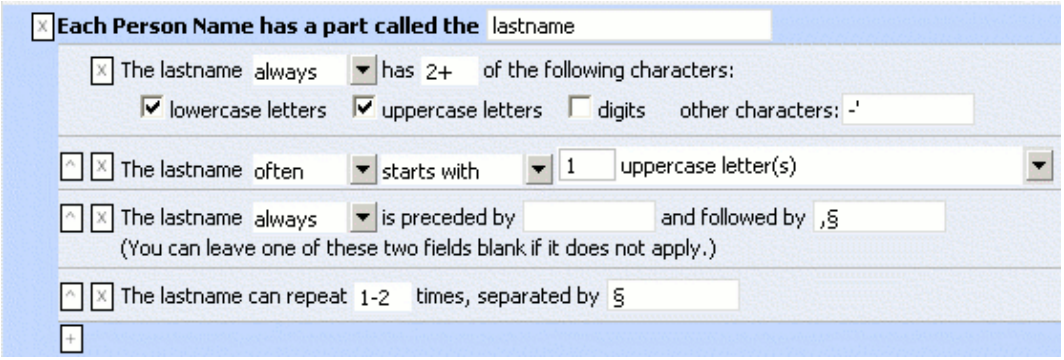
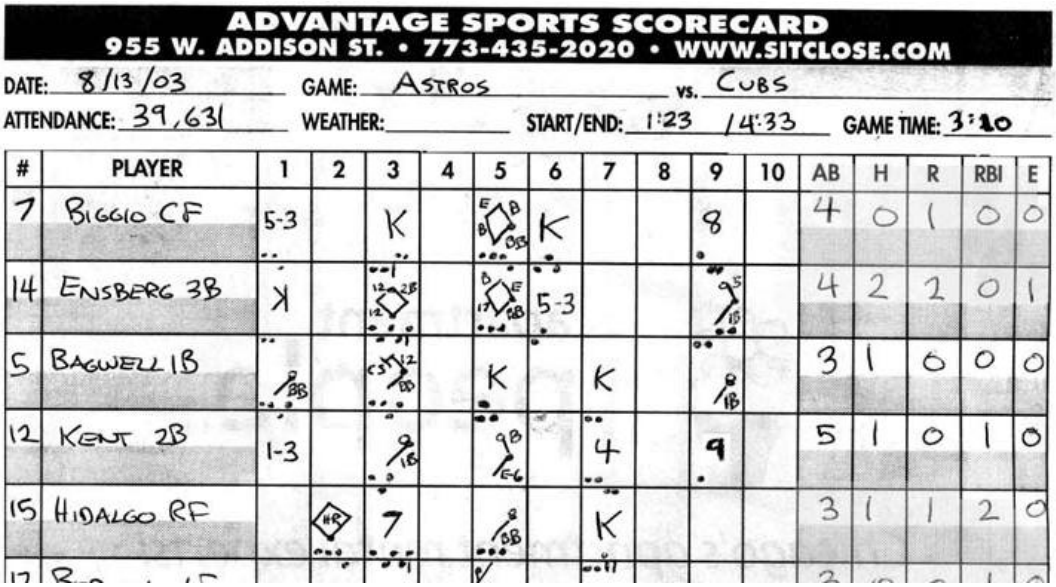


Figura 12: Topes (Scaffid et al., 2007)

O Topes (Scaffidi et al., 2007) (Figura 12) é um sistema desenvolvido para apoiar o usuário final na manipulação de dados semi-estruturados. Cada "*tope*" define como reconhecer um tipo de dado baseado no formato e contexto e como transformar esse tipo de dado de acordo com vários formatos válidos. O usuário final pode criar, compartilhar e aplicar "*topes*", de forma que os usuários possam rapidamente implementar a validação de dados e a formatação de funcionalidades.

Apesar de possivelmente não saberem programar, os usuários finais são aptos a programar. Segundo Nardi (1993), as pessoas usam linguagens formais e sistemas o tempo todo, como os números e o alfabeto. Algumas usam até notações muito mais complexas como o quadro de pontuação de baseball (Figura 13). Segundo a autora, o sucesso das aplicações de programação por usuário final (EUP – *End User Programming*) está na escolha adequada da linguagem de programação (a linguagem de script é uma delas), e esta deve ser preferencialmente orientada à tarefa.

Apesar de este ser um ponto interessante, principalmente sob o ponto de vista da Engenharia Semiótica, não iremos tratar especificamente da linguagem de programação a ser usada nos sistemas. A escolha da linguagem vai além da proposta deste trabalho.



ADVANTAGE SPORTS SCORECARD
955 W. ADDISON ST. • 773-435-2020 • WWW.SITCLOSE.COM

DATE: 8/13/03 GAME: Astros vs. Cubs
ATTENDANCE: 39,631 WEATHER: START/END: 1:23 / 4:33 GAME TIME: 3:10

#	PLAYER	1	2	3	4	5	6	7	8	9	10	AB	H	R	RBI	E
7	Biggio CF	5-3		K			K			8		4	0	1	0	0
14	Ensberg 3B	K					5-3					4	2	1	0	1
5	Bagwell 1B					K		K				3	1	0	0	0
12	Kent 2B	1-3						4		9		5	1	0	1	0
15	Hidalgo RF			7				K				3	1	1	2	0

Figura 13: Exemplo de um quadro de pontuação de Baseball

No decorrer deste trabalho serão mostrados vários exemplos de técnicas, métodos e ferramentas para usuários finais. A seguir classificamos este conjunto e extraímos os pontos principais de cada um, para que possamos relacioná-los às perguntas da análise de usuários e tarefas.

2.1.2. O que leva um usuário a adaptar um sistema?

Após todo o esforço do designer em pensar em quais adaptações podem ser necessárias e úteis para um sistema e implementar tais adaptações, surge outro problema: as adaptações propostas serão usadas pelos usuários? Mackay (1991) atenta para este problema e define um processo de decisão de customização por parte do usuário.

A primeira decisão ocorre pela escolha do uso ou não do software. Vários fatores podem afetar a decisão sobre o uso do sistema. A partir da análise sobre os usuários, podemos descobrir quais são suas expectativas em usá-lo. Um sistema intuitivo e fácil de aprender normalmente promove boas expectativas e grandes chances de ser adotado.

Caso o usuário decida que vale a pena o esforço em tentar aprender o sistema, aparece a decisão sobre a escolha do uso regular ou customizado do sistema. Os usuários precisam de um motivo para customizar. Este motivo pode

vir de quatro fatores: eventos externos, pressão social, mudanças de software e fatores internos.

Os eventos externos são conseqüência de fatores que fazem com que o usuário pense como ele pode organizar o seu tempo para usar o software. A pressão social surge quando outras pessoas compartilham determinados conhecimentos ou quando o próprio usuário vê o uso diferenciado do sistema por outras pessoas e deseja pegar "emprestado" esse uso. As mudanças no software são normalmente falhas, *upgrades* ou mudanças para um novo programa. Os fatores internos partem da própria vontade do usuário em customizar, seja por tentar algo novo, exploração do ambiente, ou mesmo tentar formas alternativas de realizar tarefas mais freqüentes.

Ainda após as mudanças, o usuário pode retornar o sistema para o uso regular. Este processo pode ser visto na Figura 14.

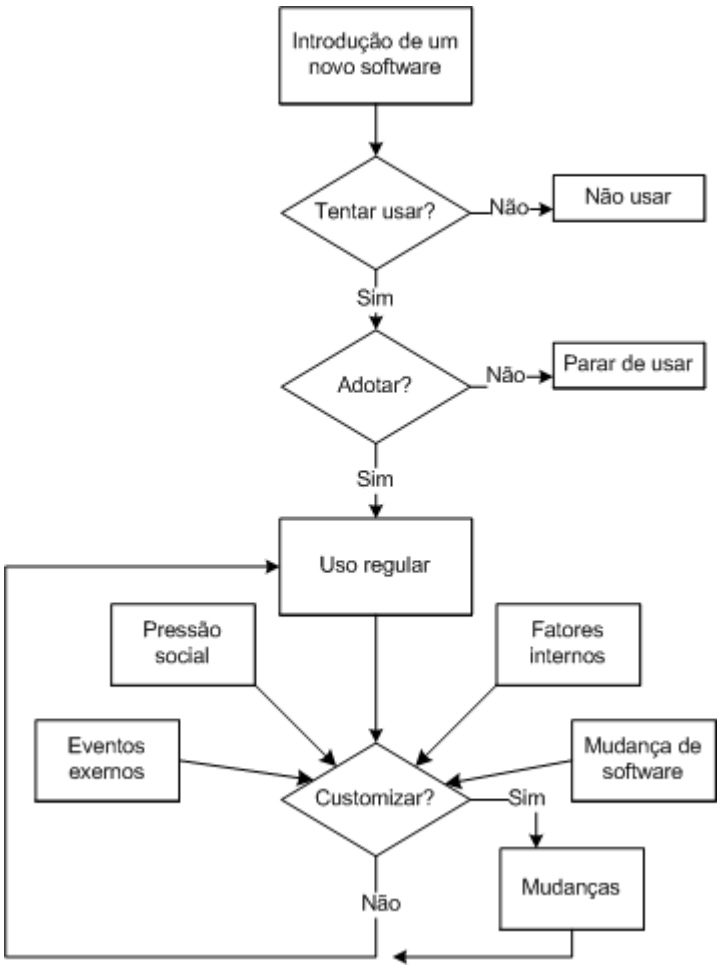


Figura 14: Processo de decisão de customização de software

As adaptações que "valem a pena" são aquelas que aumentam a eficiência do uso do sistema a partir de um conjunto simplificado de comandos e as que permitem que os usuários continuem trabalhando como faziam com o uso regular do sistema, sem que tenham que aprender novos padrões.

Em contrapartida, as funções que se tornam "automáticas", como o uso particular de comandos para funções são mais resistentes às mudanças, assim como as que deixam o ambiente mais prazeroso ou mais interessante.

Os usuários preferem (e querem) customizar padrões de comportamento e não apenas uma lista de funcionalidades. O sistema deve permitir que os usuários aprendam sobre o efeito do uso desses padrões e os modifiquem. Além disto, é natural que os usuários queiram compartilhar esse aprendizado com outras pessoas, e os designers devem ter isto em mente.

O compartilhamento é importante porque quando os usuários estão aprendendo a usar o sistema, eles conhecem pouco sobre ele para que possam tomar decisões efetivas. Por isto, a customização em um tempo posterior ao processo de aprendizagem do sistema é mais efetiva.

Os usuários são muito resistentes à mudanças, por isto, a manutenção de padrões de uso é um fator muito importante para encorajar a customização.

Esses fatores devem ser levados em consideração na tomada de decisão sobre as extensões e estão relacionados principalmente às perguntas "Por que adaptar?" e "Como adaptar?".

2.1.3. Classificações de sistemas extensíveis

A análise da literatura nos mostra que diferentes pesquisadores apresentam formas distintas de classificação para atividades de extensão, apesar de nenhum deles fazer uma referência de base teórica que auxilie o designer a tomar decisões sobre qual mecanismo poderá empregar em seu sistema.

Trigg e co-autores (1987) definem uma classificação com quatro formas de adaptação. Quando o designer do software disponibiliza objetos e comportamentos que podem ser interpretados e usados de forma diferente pelo usuário, encontramos uma adaptação do tipo **flexibilização.** Quando o usuário pode escolher entre comportamentos alternativos previamente disponibilizados pelo designer do software, estamos falando de **parametrização**. A **integração**

ocorre quando o usuário pode conectar novos componentes, externos ou internos, ao software. E alteração do próprio software pela construção de aceleradores, comportamentos especializados e/ou a adição de funcionalidade, os autores dão o nome de ***tailoring***.

O exemplo a seguir mostra o mecanismo de flexibilização existente no iTunes. O usuário pode realizar uma mesma tarefa de três formas diferentes (Figura 15).

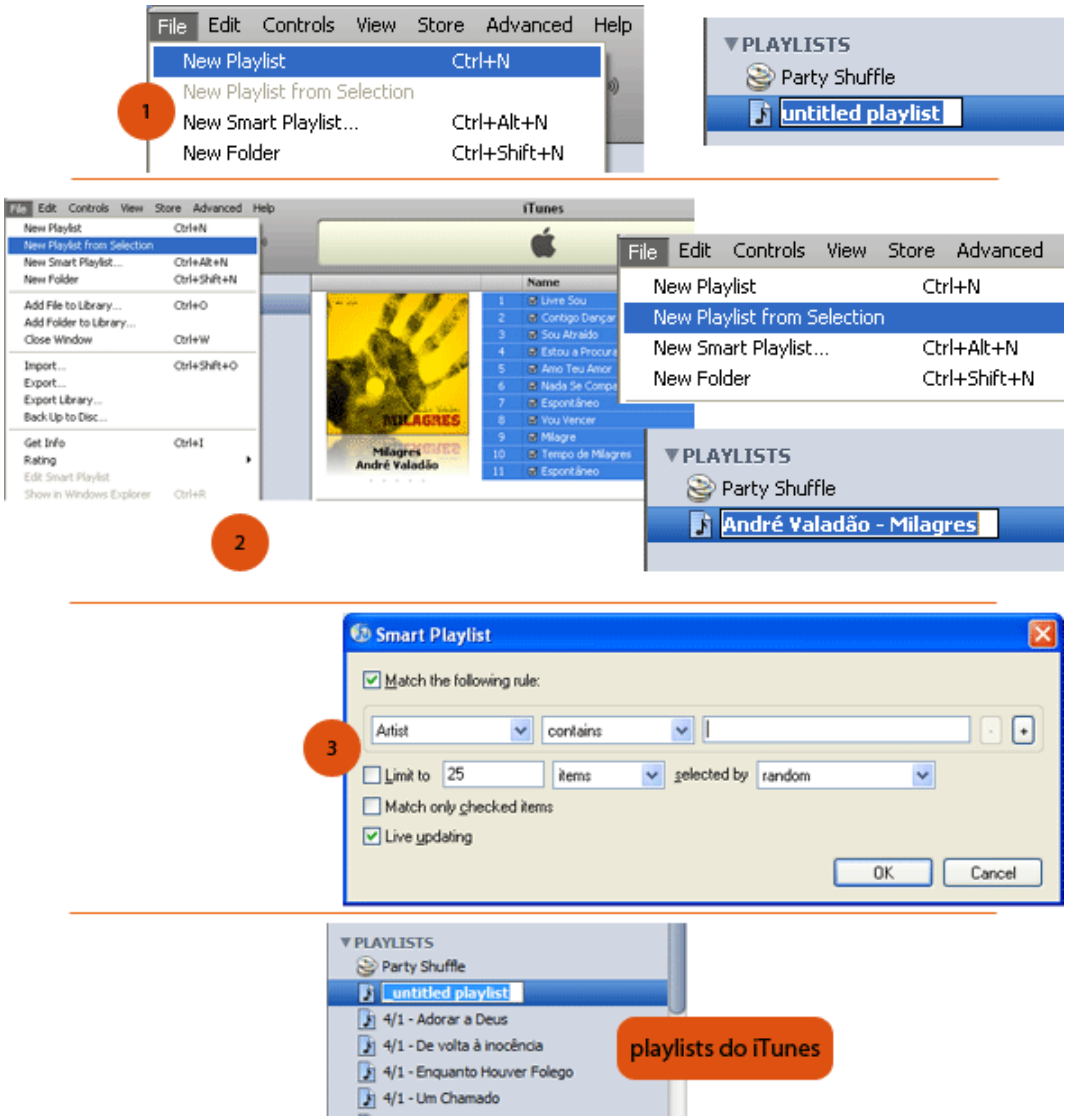


Figura 15: Criação de playlists no iTunes

O iTunes permite a criação de playlists de diversas formas:

1) através do menu File> New Playlist (ou sua variação – atalho ctrl + N)
   > *o usuário irá criar uma lista vazia, para depois arrastar as músicas da biblioteca para dentro da playlist*

2) selecionando as músicas e depois através do menu File> New Playlist from Selection
   > *o sistema cria uma lista com o nome do cantor seguido pelo nome do álbum e uma lista com as músicas selecionadas*
3) através do menu File> New Smart Playlist
   > *o usuário irá selecionar uma opção de busca e o conteúdo da string referente a ela. Ele pode definir outras configurações como o número limite de itens da lista.*

O próximo exemplo retrata a parametrização realizada pelo usuário. A Figura 16 mostra parte da tela de configuração do iTunes.

O usuário acessa a janela de configurações do iTunes (Edit>Preferences ou Ctrl+,) para informar ao sistema que, sempre que um arquivo que esteja fora da biblioteca for executado pelo sistema, uma cópia deste arquivo seja criada na pasta de músicas. O usuário pode escolher qual o caminho desta pasta.

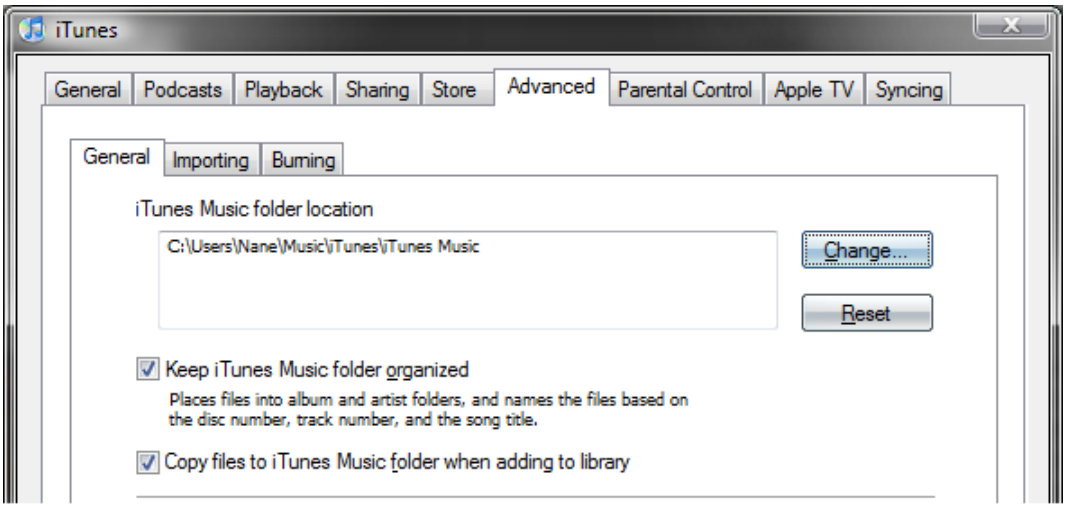


Figura 16: Parte da tela de configuração de preferências no iTunes

O Adobe® Photoshop® CS3 permite que o usuário integre *plugins* ao sistema. Para isto, o arquivo do *plugin* deve estar localizado em uma pasta específica dentro da pasta *plugins* criada na instalação do sistema. Sempre que o sistema é carregado, ele localiza esta pasta para atualizar a lista de *plugins*. O FlickShop[4] é um *plugin* desenvolvido pela empresa de desenvolvimento de software PixelNovel. Este plugin permite que o usuário faça o upload de imagens para o Flickr diretamente pelo Adobe® Photoshop® CS3.

[4] FlickShop <http://www.pixelnovel.com/flickrshop.html> última visita em 06/03/2008.

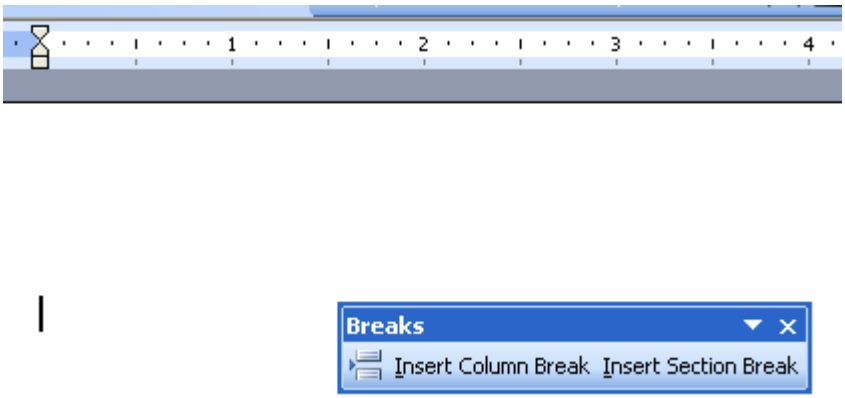


Figura 17: Microsoft® Office® 2003 – criação de uma barra de menu para acesso rápido.

O Microsoft® Office® 2003 permite que o usuário crie uma barra de ferramentas e inclua os comandos a sua escolha. No exemplo (Figura 17) o usuário criou uma toolbar flutuante chamada *Breaks* e incluiu nela três ações: inserir quebra de página, inserir quebra de coluna e inserir quebra de seção.

***

Fischer & Girgensohn, (1990), apresentam como mudanças apoiadas por um sistema que pode ser modificado por usuários finais as seguintes formas:

1. Definição de parâmetros (como em um dialogo de opções);
2. Adição de funcionalidade a objetos existentes;
3. Criação de novos objetos pela modificação dos existentes;
4. Definição de novos objetos a partir do zero.

Como podemos observar a primeira forma definida por esses autores é equivalente à parametrização definida por Trigg. As outras três se encaixam na definição de *tailoring* de Trigg. Elas descrevem o que pode ser adaptado e não exatamente qual mecanismo deve ser empregado para obter tal adaptação.

***

Cypher (1993) propôs outra classificação. A primeira categoria define as **preferências** como uso de alternativas predefinidas disponibilizadas pelo designer para acomodar as várias necessidades dos diferentes tipos de usuários. Além das preferências, Cypher define outras três categorias: as linguagens de script, os gravadores de macro e a programação por demonstração.

As **linguagens de script** são pequenas e simples linguagens de programação, cujo vocabulário é feito sob medida para os objetos e ações de um domínio particular de aplicação, na confecção de extensões. Os **gravadores de**

macro são mecanismos que possibilitam ao usuário gravar a seqüência de ações que ele realiza, podendo ativar, posteriormente, a repetição da seqüência. A **programação por demonstração** faz uso de mecanismos de inferência para realizar a criação de programas generalizados a partir de seqüência de ações dos usuários.

A definição de preferências de Cypher é semelhante à de parametrização defendida por Trigg. Além disto, podemos associar as outras três classificações ao *tailoring* porposto por Trigg.

O Adobe® Photoshop® CS3 apóia a utilização das linguagens de script AppleScript, VBScript e JavaScript™. O usuário pode escolher uma dessas três linguagens para escrever um script e depois importar o script criado através do menu [File > Scripts > Browse...]. A Figura 18 mostra o script Image Processor sendo executado.

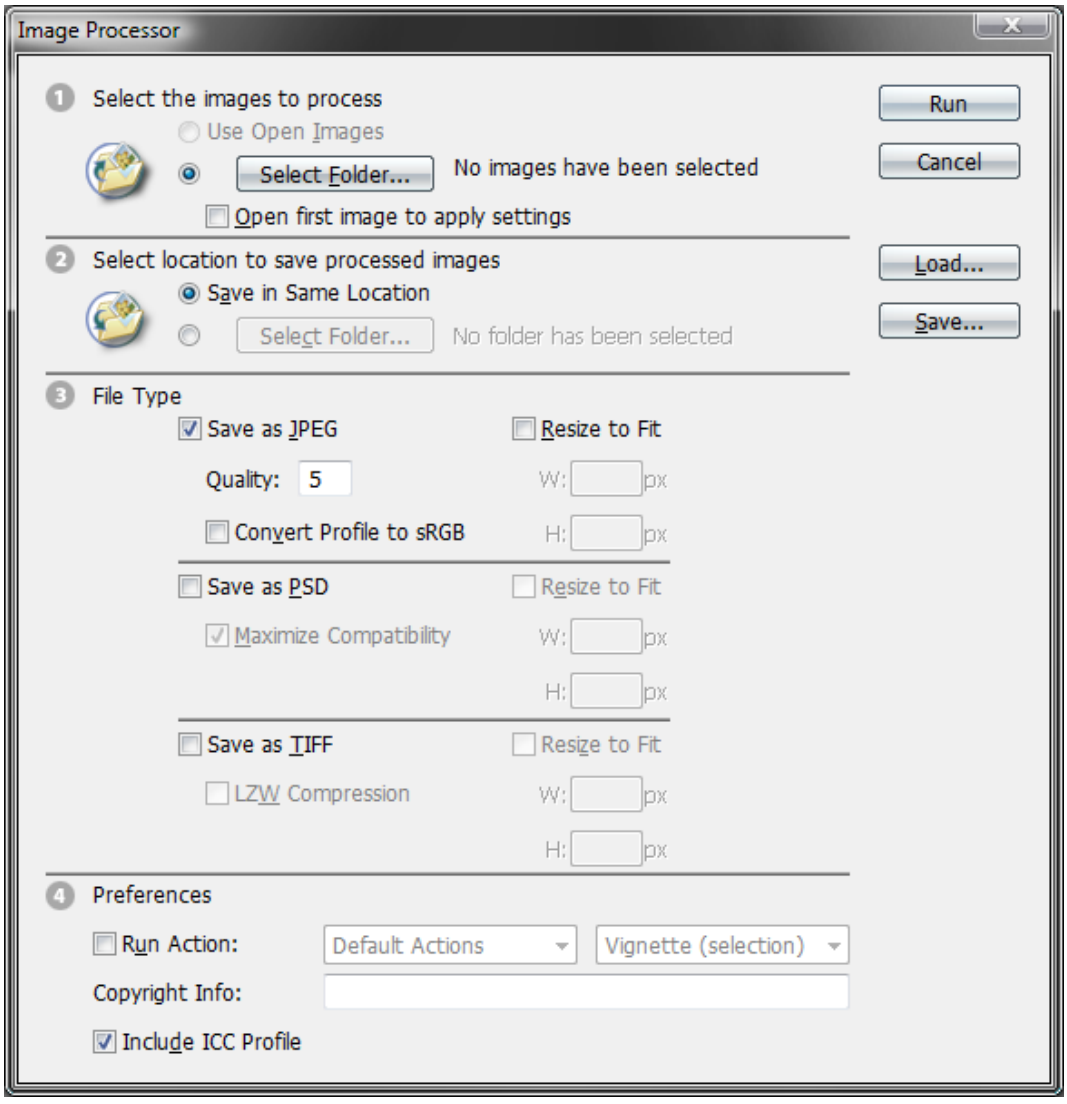


Figura 18: Janela do script Image Processor do Adobe® Photoshop® CS3

Ainda no Adobe® Photoshop® CS3 o usuário pode gravar macros, utilizando *Actions*. Ele deve acionar o comando para começar a gravar a ação, interagir com o sistema de acordo com as etapas da tarefa e depois parar a gravação. O sistema guarda a lista de *Actions* criadas para que o usuário possa executá-las sempre que necessário. A Figura 19 mostra a janela de *Actions*. A ação Teste 1 selecionada demonstra a criação de um novo documento (ação *Make*) com todas as informações relativas a ele, como altura e largura. Depois, o usuário escreveu um texto no documento (ação *Make text layer*).

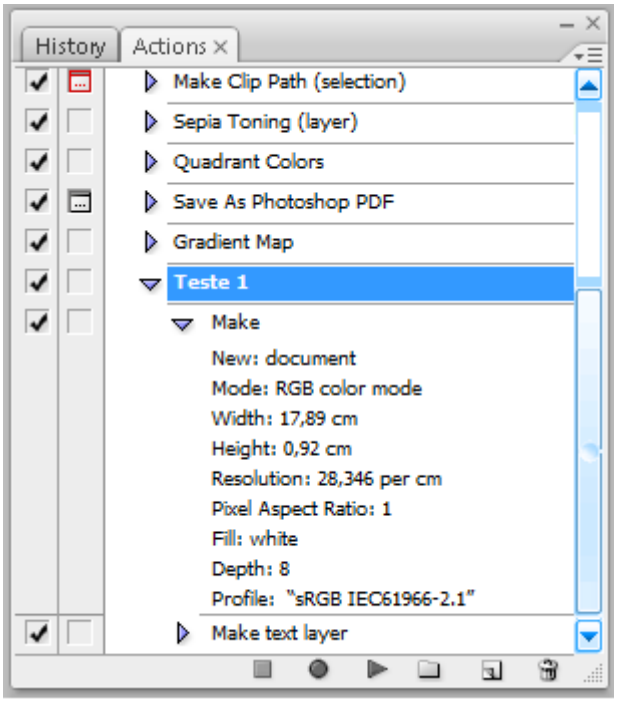


Figura 19: Janela de Actions do Adobe® Photoshop® CS3

O Tinker (Lieberman, 1999) é um ambiente de programação por demonstração. A Figura 20 (a) mostra o Tinker em execução. Ele contém uma janela com uma lista de itens, representando expressões em Lisp e seus valores. Cada item possui duas partes: uma parte de resultado e uma de código.

O exemplo em questão deve demonstrar como mover um bloco qualquer sobre outro. Em (b) vemos o exemplo trivial de posicionamento entre blocos: A pode ser posicionado sobre B. Em (c) vemos o exemplo mais complexo, onde X não pode ser movido diretamente para cima de Y.

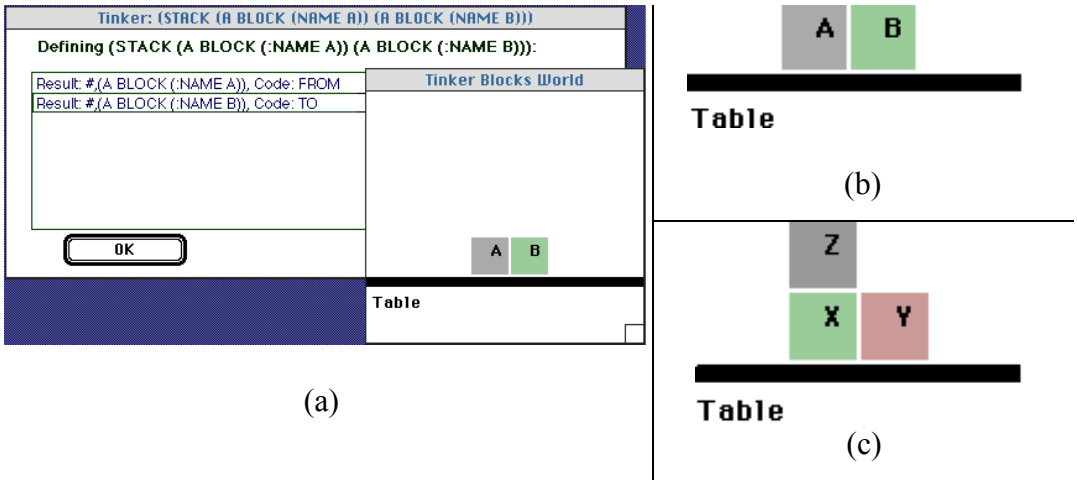


Figura 20: (a) Tinker - programação por demonstração; (b) exemplo trivial de posicionamento entre dois blocos; (c) exemplo de posicionamento de três blocos

***

Mørch (1997) apresenta outra forma de classificação, embasada na teoria da Engenharia Cognitiva. Ele chama de *tailoring* qualquer tarefa de alteração de um software já existente, ou seja, é uma forma generalizada de tratar todos os mecanismos de adaptação. Assim, ele define três possíveis formas de *tailoring*: a customização, a integração e a extensão. Para ele, a modificação da aparência dos objetos de apresentação ou edição dos valores de seus atributos selecionando entre um conjunto de opções de configurações predefinidas é um meio de **customização**. Quando o usuário adiciona novas funcionalidades ao software através da ligação de componentes predefinidos dentro ou através de aplicações sem, no entanto, acessar o texto da implementação, ele está realizando uma tarefa de **integração**. Ele chama de **extensão** a adição de novas funcionalidades às aplicações e aos componentes textuais das aplicações, em "pontos abertos" – isto é, áreas da implementação predefinidas pelo designer que podem apresentar diferentes alternativas de design**.**

Podemos fazer uma relação quase direta com a classificação proposta por Trigg. Mørch não menciona a flexibilização, no entanto podemos associar as definições de Trigg e Mørch da seguinte forma (pensando sempre em Trigg ↔ Mørch): parametrização ↔ customização; integração ↔ integração e *tailoring* ↔ extensão.

***

Silva (2001) classifica os mecanismos de extensão segundo os paradigmas de computação (e conseqüentemente a linguagem) que eles utilizam. Ele define três tipos de paradigmas e cada um deles possui dois ou mais mecanismos de implementação. Na **programação paramétrica**, o designer deve disponibilizar para o usuário final um conjunto básico de módulos e uma forma de configurar estes módulos. Dentro deste paradigma estão as atividades de configuração de preferências, personalização da interface e uso de moldes. Na **configuração de preferências**, o usuário pode habilitar ou desabilitar opções, assim como configurar determinados parâmetros relacionados à interface ou a determinadas funções do sistema. Na **personalização da aparência da interface**, o usuário realiza mudanças nos elementos visuais ou léxicos da interface. O **uso de moldes** permite que os usuários agrupem parâmetros, criem uma denominação para este agrupamento e o usem para modificar o leiaute de um documento.

Na **programação imitativa** o designer deverá disponibilizar para o usuário final mecanismos que permitam tanto a definição de qual a seqüência de ações que ele deseja replicar quanto a ativação de tal seqüência. Dentro deste paradigma estão a gravação de macros e a programação por demonstração ou exemplos. Na **gravação de macros**, o usuário pode gravar uma seqüência de comandos gerados na realização de uma tarefa, atribuir um nome a este conjunto de comandos e associá-lo a um atalho ou item de menu. Já na **programação por demonstração ou exemplos**, o usuário cria "programas generalizados" a partir de um conjunto de exemplos de tarefas realizadas por ele. O sistema, através de indução, determina a intenção do usuário de repetir uma determinada ação.

O terceiro paradigma retratado por Silva é o da **programação descritiva**. Neste paradigma o designer deverá disponibilizar mecanismos que dêem suporte à criação e interpretação de planos de ação do usuário. O autor encaixa dentro deste paradigma dois mecanismos: a aprendizagem por desvelamento e os editores de roteiros (*scripts*) e macros. Na **aprendizagem por desvelamento**, há a integração da linguagem de manipulação direta da interface com a linguagem de extensão textual centrada no domínio da aplicação. Através de técnicas de aprendizagem progressiva, a relação entre as linguagens é apresentada ao usuário. Nos **editores de *scripts***, ocorre a ligação de pequenos programas (ou partes independentes de programas) através de uma linguagem de programação simples e pequena, criando, assim, uma nova funcionalidade para a aplicação.

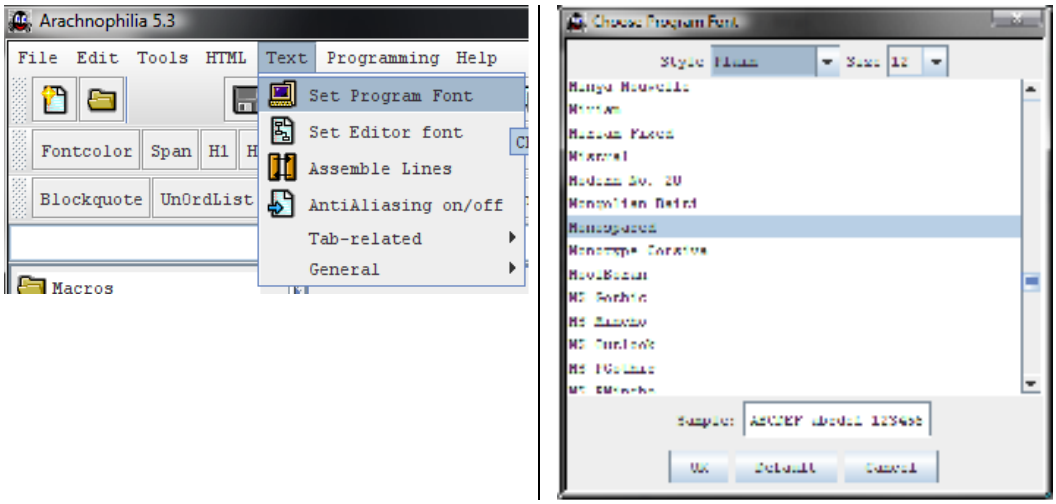


Figura 21: Arachnophilia 5.3 : Configuração da fonte do sistema

O Arachnophilia 5.3 (Figura 21) permite que o usuário mude as configurações de fonte do sistema. Ele pode escolher o estilo e o tamanho. Além desta opção, o usuário também pode escolher uma opção de tema (esquema de cores da interface do sistema), dentre algumas fornecidas pelo designer. Este exemplo retrata o mecanismo de personalização da aparência da interface.

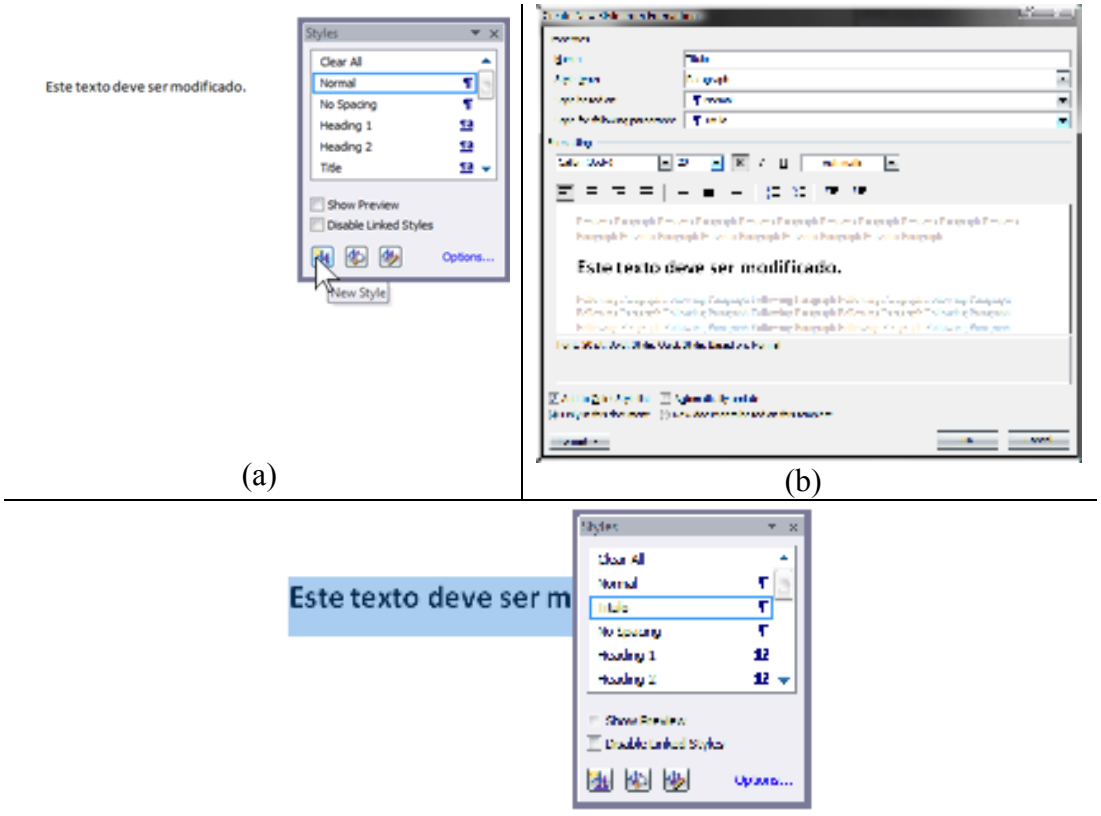


Figura 22: Criação e uso de estilos do Microsoft® Office® 2007

O uso de moldes pode ser exemplificado através da criação e uso de estilos do Microsoft® Office® 2007. A Figura 22 mostra uma das diversas maneiras de criação de um estilo.

Em (a) o usuário já digitou um texto e seleciona a opção [New Style] da caixa de estilos suspensa. Em (b) ele escolhe um nome para o novo estilo (Título) e configura sua formatação. Em (c) ele seleciona o texto e escolhe o estilo [Título] da lista de estilo, modificando o estilo do texto selecionado de Normal para Título.

SchemeChart é um ambiente de programação para a representação e apresentação de gráficos. Ele fornece um meio no qual usuários experientes criam uma variedade de designs, enquanto as críticas, catálogos icônicos, tutoriais acessíveis pelo menu e objetos "*queryable*" (que podem ser questionados) do sistema promovem meios de aprendizado tanto sobre a linguagem Scheme quanto sobre técnicas apropriadas para o designe de gráficos e grafos (Eisenberg, 1994). Este é um exemplo de aprendizagem por desvelamento.

Na Figura 23 o usuário edita uma expressão de exemplo no painel à esquerda, para criar um novo gráfico de barras trapezoidais. Quando esta nova expressão é avaliada, o sinal de alerta crítico (o ponto de exclamação na janela de gráficos) fica piscando várias vezes para indicar que um potencial problema foi detectado pelo novo gráfico criado.

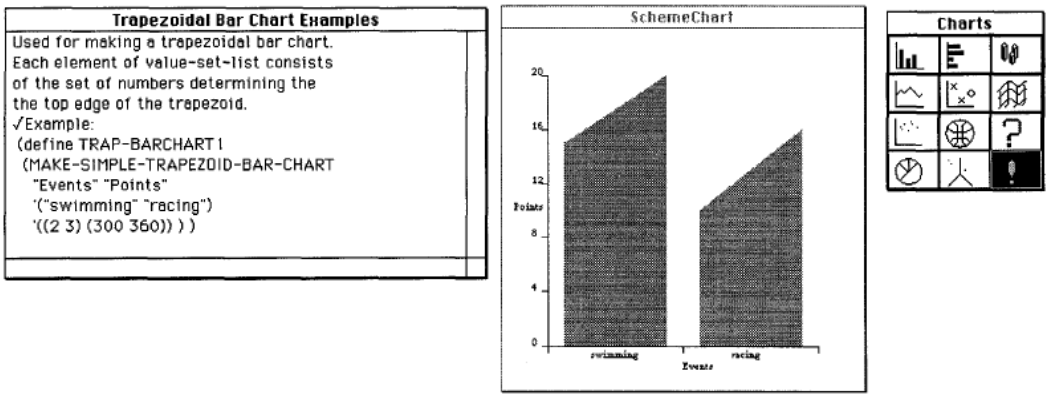


Figura 23: SchemeChart: exemplo de aprendizagem por desvelamento

***

Blackwell (2006) propõe outra classificação, enfatizando os conceitos de EUD sob uma perspectiva mais técnica.

Scripting engloba mecanismos que utilizam uma linguagem de script para automatizar seqüência de ações dos usuários. O maior problema é que os usuários finais podem não saber (ou mesmo não querer) programar. Neste caso, o usuário deve receber apoio com relação a certas técnicas de programação para conseguir compreender e utilizar a linguagem em questão. As linguagens de script são comumente encontradas em processadores de texto, ambiente web e em aplicações dinâmicas de propósito gerais.

As linguagens de **programação visuais** permitem que o usuário especifique os programas através da manipulação gráfica de elementos, ao invés da mais comum forma textual. Alguns pesquisadores alegam que as maiores contribuições dessas linguagens, entre outras coisas seriam:

- que as imagens representam alto nível de abstração, então nenhuma sintaxe é envolvida (Myers, 1992 *apud* Nardi, 1993);
- quando estruturadas apropriadamente, as imagens podem ser entendidas independente da língua falada pelas pessoas (Shu, 1988; Myers, 1989a, 1989b *apud* Nardi, 1993);
- as pessoas podem atribuir significado às imagens de forma mais concisa do que aos textos (Nardi, 1993); e
- as linguagens visuais exploram duas dimensões, o que pode trazer mais informações sobre as estruturas do que as linguagens textuais de única dimensão (Myers, 1989b *apud* Nardi, 1993).

Apesar de alguns sistemas que utilizam linguagens visuais comprovarem o sucesso das mesmas quando comparadas a sistemas concorrentes, nem todas as qualidades citadas anteriormente representam, de fato, um avanço radical para a solução de problemas de EUD.

Por exemplo, Nardi destaca que **sintaxe** representa regras de construção, e a necessidade para essas regras está presente em qualquer notação (Nardi, 1993). O que ocorre, de fato, é que algumas representações utilizam ícones de significado bastante claro, facilitando seu uso. Alguns elementos podem não possuir uma representação gráfica convencionada, como no caso de elementos abstratos, e isto pode dificultar muito a interpretação do usuário. Quando falamos de aplicações extensíveis, estamos a todo o tempo lidando com elementos abstratos, o que dificulta sua representação visual (Barbosa, 1999).

A **reescrita gráfica de sistemas** envolve aplicações que permitem que o usuário final interaja por meio da interface gráfica. O AgentSheets (Repenning, 1994) é um exemplo de sistema EUD. Ele permite que não-programadores criem agentes com comportamentos e missões. Além disto, podem ensinar aos agentes como reagir com as informações e como processá-las. Dessa forma, o usuário pode criar jogos interativos, mundos virtuais, simulações de treinamento etc.

Outro mecanismo adotado por Blackwell são as ***spreadsheets*** (planilhas). Esta é uma das funcionalidades mais usadas, principalmente para a criação de processamento de dados, análise e aplicações interativas. O usuário pode computar os valores de células (as variáveis) através do relacionamento (as funções) entre elas. Nardi aponta suas principais características: expressividade e de fácil aprendizado (Nardi, 1993). O sucesso desta ferramenta está no fato do usuário não precisar saber, e de certa forma entender, os passos necessários para realizar determinada operação. Por exemplo, caso o usuário queira saber a média entre um conjunto de valores usando a função AVERAGE, ele precisa apenas fornecer esses valores (argumentos), sem precisar saber o que a função faz para resolver o problema. Apesar de ser menos flexível do que uma linguagem de programação (com seus diversos tipos e variáveis), as operações básicas fornecidas por esta ferramenta permitem que os usuários realizem suas tarefas.

Assim como outros pesquisadores mencionados, Blackwell enfatiza a **programação por demonstração ou exemplo**. O sistema cria programas a partir das operações dos usuários. Diferentemente da gravação de macros, o sistema não cria os programas à imagem das ações, mas a partir de generalizações do conjunto de operações capturadas.

Na técnica de programação, por exemplo, o sistema realiza determinadas ações a partir da observação do usuário. A partir de uma seqüência de ações do usuário, o sistema infere sua intenção e realiza uma ou mais tarefas para atingir o objetivo do usuário. Neste cenário, o usuário dá ao sistema exemplos concretos e o sistema infere o objetivo, detectando um padrão entre as ações e o seu "conhecimento" sobre o que é possível fazer com os dados recebidos.

O problema desta técnica ocorre quando o sistema está no comando o tempo todo (note que isto pode ser uma finalidade do sistema, implementado de propósito pelo designer). Pode acontecer de o sistema tomar uma decisão, realizar sua decisão e não permitir que (ou não deixar claro como deve ser feito para que)

o usuário volte atrás. Porém, alguns cuidados podem ser tomados, e, de acordo com a finalidade do sistema, este problema pode ser resolvido.

O programa TuneUp Utilities 2007 foi implementado para auxiliar a administração do sistema operacional. Ele possui várias ferramentas que podem ser usadas. Quando o usuário acaba de fazer a instalação e resolve usar uma das ferramentas, antes de abrir a janela principal, o sistema mostra uma janela de ajuda, informando para que serve a ferramenta escolhida e como o usuário deve usá-la. No final da janela, há uma opção para desativar esta ajuda na próxima vez que o usuário escolher esta ferramenta. Todas as outras ferramentas deste sistema possuem uma janela de informação semelhante. A partir da terceira vez que o usuário desativa esta opção de ajuda, o sistema avisa que percebeu que ele desativou duas outras janelas além da atual, e pergunta se ele deseja desativar todos os outros diálogos de ajuda como este.

O mais interessante deste exemplo, além do questionamento sobre o que o usuário deseja que seja feito a partir da inferência do sistema, é que o usuário consegue reativar as janelas de ajuda posteriormente.

A Tabela 5 mostra a comparação entre as classificações proposta pelos autores citadas sob o ponto de vista de procedimentos de extensão. Na próxima seção apresentamos a nossa classificação que, como da Silva (2001), se fundamenta na Engenharia Semiótica, mas, diferente dele, nosso trabalho não enfatiza as definições das linguagens do sistema: a de manipulação direta da interface e a linguagem de extensão textual centrada no domínio da aplicação.

Trigg (1987)	Fischer e Gingersohn (1990)	Cypher (1993)	Mørch (1997)	Silva (2001)		Blackwell (2006)
Flexibilização	n/a	n/a	n/a	n/a	n/a	n/a
Parametriza-ção	Definição de parâmetros (como em um diálogo de opções)	Preferências	Customização	Programação paramétrica	Personalização da aparência da interface; Configuração de preferências	
Integração	n/a	n/a	Integração	n/a		
Tailoring	n/a	n/a	Extensão	Programação paramétrica	Uso de moldes	Spreadsheets
		Gravadores de macros		Programação imitativa	Gravação de macros	n/a
		Programação por demonstração			Programação por demonstração ou exemplo	Programação por demonstração ou exemplo
						Reescrita gráfica de sistemas
		n/a		n/a		Linguagens visuais
		Linguagem de script		Programação descritiva	Roteiros	Scripting
n/a		n/a	n/a		Aprendizagem por desvelamento	n/a

Tabela 5: Tabela comparativa dos diversos procedimentos de extensão

2.2. Engenharia Semiótica

Esta seção apresenta alguns conceitos de Engenharia Semiótica relevantes para este trabalho, promovendo o contexto desta pesquisa e descrevendo como a engenharia semiótica contribui para o desenvolvimento de aplicações extensíveis. A seguir, falaremos um pouco sobre os conceitos de semiótica e de engenharia semiótica. Ao final desta seção falaremos mais sobre a engenharia semiótica e sua relação com o desenvolvimento de aplicações por usuários finais.

A Semiótica é o estudo de signos, dos processos de significação e de como os signos e esses processos ocorrem na comunicação (de Souza, 2005). A engenharia semiótica estende os conceitos da teoria semiótica, adicionando a

perspectiva da construção e design de artefatos. A seguir são apresentados alguns conceitos relevantes para este trabalho.

A semiose e os signos

Em engenharia semiótica estamos a todo tempo referenciando os **signos**. Um signo é qualquer coisa que representa algo para alguém (***representamen***) (Peirce, 1931 - 1958). Uma pessoa faz uma interpretação de um signo através da equivalência com outro signo de sua mente. Esta equivalência é o **interpretante** do primeiro signo e a relação entre os signos que podem ser interpretados é a **semiose**. O interpretante de um signo é um novo signo. Este novo signo poderá ter um novo interpretante que, por sua vez, gerará um novo signo. Este processo pode ocorrer *ad infinitum* e, por isto, dizemos que a **semiose é ilimitada**.

Os signos podem ser codificados em linguagem natural ou através de sistemas de significação, e possuem o objetivo de comunicar atitudes, intenções e conteúdos dos sistemas interativos.

O processo de comunicação

Dois personagens são de fundamental importância em IHC e, particularmente, em Engenharia Semiótica: o designer e o usuário.

Os designers devem comunicar aos usuários todo o significado do artefato criado e, por sua vez, os usuários devem ser capazes de entender e responder à comunicação do designer.

A transferência de mensagens ocorre através da interface do artefato. As mensagens podem ser codificadas através de palavras, símbolos gráficos e ajuda. Esta mensagem designer-para-usuário recebe o nome de **metacomunicação**, pois é uma comunicação cujo conteúdo é a comunicação. Podemos parafraseá-la da seguinte forma:

> Eis o meu entendimento de quem você é, o que eu aprendi que você quer ou precisa fazer, de quais maneiras preferenciais e por quê. Este é o sistema que eu concebi para você e esta é a maneira que você pode ou deve usá-lo para atingir seus objetivos.
>
> Here is my understanding of who you are, what I´ve learned you want or need to do, in which preferred ways, and why. This is the system that I have therefore designer for you, and this is the way you can or should use it in order to fulfill a range of purposes that fall within this vision. (de Souza, 2005, p.25)

Podemos notar, principalmente a partir do teor desta meta-mensagem, que o designer é posto em uma posição tão importante quanto a dos usuários. Apesar de diferir consideravelmente da abordagem centrada no usuário, a meta-comunicação pode ser considerada como um complemento a esta perspectiva. Na abordagem centrada no usuário (UCD – *User Centered Design*) proposta por Norman (1986), os designers tentam definir, da forma mais precisa possível, o que os usuários querem ou precisam. Isto pode ser feito através da análise de tarefas, para que posteriormente possa ser desenvolvido um modelo de design, que é a imagem do sistema. Este modelo é a etapa final e última chance de promover o sucesso da aplicação. Caso o modelo seja definido de acordo com conceitos familiares aos usuários e intuitivos, grandes são as chances dos usuários conseguirem utilizar o sistema de forma agradável para atingir seus objetivos, e poderão se lembrar do que fizeram ou como o sistema deve ser usado.

A engenharia semiótica parte também da análise sobre o que os usuários querem ou precisam, porém, ela não está interessada em construir uma imagem do sistema, mas um meio de comunicação onde o designer comunica ao usuário sua visão de design, e o usuário responde interagindo com o sistema. No momento da interação, o designer não está presente através da interface do sistema, mas esta carrega a visão do designer sobre as intenções do mesmo. Logo, o sistema se comporta como sendo o **preposto do designer**.

A meta-comunicação entre o designer e o usuário ocorre em quatro passos:

- O designer estuda o usuário, suas atividades e seu ambiente;
- O designer expressa, através da tecnologia, sua visão sobre como os usuários, suas atividades e o ambiente podem ou devem ser modificados (porque os usuários desejam que isto ocorra);
- Os usuários desempacotam a mensagem do designer através da interação com o sistema;
- Os usuários finalmente entendem todo o conteúdo da mensagem do designer, e respondem a ela de acordo.

Neste trabalho, obteremos o conhecimento sobre quem é o usuário, quais são suas atividades e qual o seu ambiente através do método de análise de usuários e tarefas para design de interface que é discutido com mais profundidade

no próximo capítulo. Escolhemos este método porque a engenharia semiótica não possui uma metodologia específica para este fim.

Após esta análise, definiremos como o sistema deve ser e quais as possíveis extensões que ele dará suporte. Nesta etapa, estaremos caminhando para o segundo ponto, onde iremos unir o conhecimento sobre os usuários e as tarefas, com o conhecimento técnico a respeito de como a tecnologia poderá apoiar os usuários.

Quando os usuários começarem a interagir e entenderem a mensagem do designer, eles conseguirão usufruir das extensões, respondendo de acordo com o que foi proposto para atingir seus objetivos.

Ao chegar ao quarto ponto, os usuários compreendem completamente a mensagem do designer, independente se estão satisfeitos ou não com o sistema.

2.2.1. EUD sob a perspectiva da Engenharia Semiótica

A metamensagem para sistemas customizáveis e extensíveis pode ser modificada para:

> Eis o meu entendimento de quem você é, o que eu aprendi que você quer ou precisa fazer, de quais maneiras preferenciais e por quê. Este é o sistema que eu concebi para você e esta é a maneira que você pode ou deve usá-lo para atingir seus objetivos. **Porém eu sei que você irá querer modificar minha visão para fazer coisas (de uma forma) que eu não imaginei. Eu posso disponibilizar as mudanças que você gostaria de fazer, mas para isto, você precisa me dizer o que deseja neste pedaço de código.**
>
> Here is my understanding of who you are, what I´ve learned you want or need to do, in which preferred ways, and why. This is the system that I have therefore designer for you, and this is the way you can or should use it in order to fulfill a range of purposes that fall within this vision. **But I know you may want to modify my vision in order to do things (in a way) that I haven´t thought of. I can handle the changes you may wish to do, provided that you can say what you want in this particular code. (de Souza, 2005, p. 191).**

A partir da visão da engenharia semiótica de que um software é na verdade uma mensagem precisamos analisar os processos comunicativos que atuam nas linguagens de interface, programação e explicação. A **linguagem de interface** (*User Interface Language* - UIL) comporta os comandos usados para a troca de mensagem entre usuário e sistema. A linguagem de programação ou **linguagem de extensão** (*End-User Programming Language* - EUPL) possibilita a comunicação das operações metalingüísticas (as que relacionam elementos do

meta-modelo do software) requeridas para a criação das extensões. A **linguagem de explicação** e/ou de documentação (*User Explanation Language* - UEL) tem como objetivo explicar ao usuário os elementos interativos da UIL, a fim de auxiliar no uso do software extensível.

A engenharia semiótica propõe como característica principal destas linguagens o **contínuo semiótico**, a partir dos princípios de Abstração Interpretativa e Contínuo Semiótico (de Souza, 1999) explicados a seguir.

Neste trabalho, nos concentramos em fazer a ponte entre a etapa de análise e a decisão sobre quais estratégias de extensão devem ser apoiadas. A definição da linguagem de extensão, que deverá respeitar o princípio de contínuo semiótico, envolve decisões de design posteriores que estão fora do escopo deste trabalho.

O grande desafio que este trabalho procura resolver é: 1) **quando** podemos estender um sistema, 2) **como ocorrerá** a extensão, e 3) **como explicar** aos usuários o funcionamento das extensões. Cabe aqui uma observação. Chamaremos de extensão qualquer método que possa ser aplicado para que o sistema "ganhe" novas funcionalidades, seja ela por parte do próprio usuário ou por parte do próprio sistema, através de inferência. Dentre esses métodos estão a customização, adaptação, as ferramentas de EUD (*scripting*, linguagens de programação visuais, reescrita gráfica de sistemas, *spreadsheets*, programação por demonstração ou exemplo, programação em linguagem natural, "*help me test*", componentes "*plug-and-play*" e gravação de macros), extensão e personalização.

***

De acordo com de Souza & Barbosa (2006) um sistema de significação é uma linguagem ou um conjunto de linguagens relacionadas que podem ser processadas a partir de três dimensões semióticas: léxica, sintática e semântica. A análise que as autoras fazem vai além dos mecanismos de adaptação, passando a observar o resultado final da adaptação, ou seja, o efeito de uma adaptação de acordo com essas dimensões semióticas apresentadas.

A dimensão léxica é composta pelo vocabulário usado para formar sentenças significativas da linguagem. A sintática diz respeito às combinações que podem ser feitas no vocabulário e às estruturas e símbolos de alto-nível que podem ser criados com tais combinações. A dimensão semântica trata de duas operações na computação lingüística: 1) o estabelecimento do mapeamento

correto entre os itens do vocabulário ou símbolos de alto-nível e as representações ou ações correspondentes ao que eles significam; e 2) a efetivação da operação associada ao significado computado.

Desta forma, podemos ter várias combinações quando estamos tratando de adaptações. As combinações possíveis são mostradas na Figura 24. As três primeiras linhas da tabela preservam o significado do sistema, ou seja, a meta-mensagem original criada pelo designer, enquanto as outras quatro, não. As alterações que preservam o significado normalmente são consideradas customizações, enquanto as que não preservam, são as extensões.

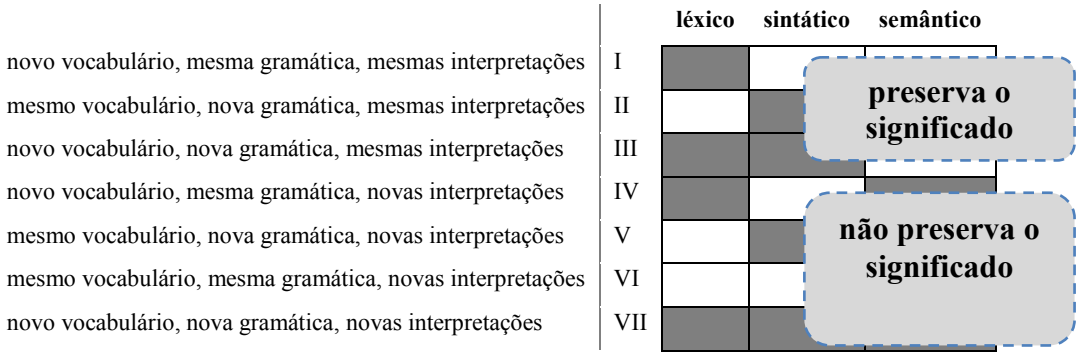


Figura 24: Possibilidades para mudanças nas dimensões da linguagem computacional [extraída de (de Souza & Barbosa, 2006)]

A **combinação do tipo I** trata basicamente das alterações de nomes dos itens do vocabulário ou introdução de sinônimos (criação de apelidos, *alias* – *aliasing*). É necessário que o designer estabeleça restrições com relação a alguns itens. Esses itens são denominados **signos impermeáveis**, que fazem parte da identidade da aplicação. Suponhamos que fosse permitido alterar o nome de um botão de *atualizar* para *cancelar*. Sem alterarmos a função designada para este botão, poderíamos causar um caos na aplicação. Por isto, a alteração dos nomes dos signos ou criação de sinônimos deve ser restrita ao vocabulário léxico da aplicação.

A **combinação do tipo II** promove mudanças nas regras gramaticais. Isto pode ser obtido através da modificação da ordenação de signos e símbolos, como por exemplo, a ordem das barras de tarefas de um editor de texto. Da mesma forma, estaríamos interessados em reordenar seqüências de tarefas.

Seria interessante poder selecionar uma foto do organizador de fotos de um celular para depois pedir para atribuí-la a um contato, ao invés de termos que

selecionar um contato, pedir a opção para atribuir uma foto para só depois escolher a foto da lista.

A **combinação do tipo III** afeta os signos impermeáveis e é conseqüência da remoção de signos não essenciais. Da mesma forma, o usuário pode adicionar outros signos. Esta combinação pode ser exemplificada através da escolha de componentes para serem ou não instalados durante um *wizard* de instalação de *software*. Um problema que podemos ter com esta customização é não sabermos como ativar determinados componentes que não foram ativados na instalação. Por isto, da mesma forma que na primeira combinação, o designer deve selecionar quais signos fazem parte da identidade da aplicação, decidindo quais podem ser ativados ou desativados.

A **combinação do tipo IV** permite atribuir a signos existentes significados diferentes dos que foram atribuídos a eles sem, no entanto, atribuir nova estrutura. Um exemplo desta combinação aparece quando um usuário cria um novo estilo de texto a partir de um outro. Neste caso, um mecanismo de checagem de consistência deve ser incorporado à aplicação para que não haja nenhuma surpresa, caso a perspectiva do usuário não seja a mesma da interpretada pela aplicação. Além de fazer a combinação, o usuário pode atribuir um nome a ela Este tipo de combinação é comumente utilizado em aplicações que não envolvem EUD.

A **combinação do tipo V** permite introduzir extensões de significado com diferentes combinações e/ou estruturas sintáticas. Quando atribuímos um valor *default* para um determinado componente, estamos utilizando este tipo de combinação, através de uma perspectiva de parametrização. Obviamente, o designer deve propor uma maneira para o usuário "voltar atrás" em sua decisão sobre esses valores. Ainda nesta combinação, temos a reordenação de tarefas que resultam em efeitos diferentes.

Imagine que um usuário deseja encontrar uma determinada palavra em um texto. Ele pode fazer isto de duas formas: a) selecionar uma ocorrência de uma palavra e depois ativar a opção de busca (ctrl+f); ou b) ativar a opção de busca (ctrl+f) e depois escrever a palavra. Na forma (a) a palavra já aparece na caixa de diálogo da busca e esta forma permite que o diálogo "guarde" esta informação na próxima vez que o usuário acessar a busca (caso seja pela forma b).

A forma (a) modifica o valor *default* da busca (de vazio "" para a palavra a ser encontrada), enquanto a forma (b) não. Este exemplo, apesar de ser fictício, é uma boa solução para auxiliar tarefas repetitivas.

A **combinação do tipo VI** altera radicalmente o significado dos signos e símbolos. Ela se refere a usar signos da aplicação com significado diferente do proposto pelo designer. Ela deve ser usada cautelosamente ou, na melhor das hipóteses, nunca deve ser usada. Ela pode alterar o significado original do design, potencialmente alterando a identidade da aplicação ou mesmo criar ambigüidades na interpretação dos signos e símbolos.

A **combinação do tipo VII** requer que os usuários tenham conhecimento desejável de lingüística e habilidade em codificação. Normalmente se resume a atividades como criação de scripts e programação de macros. As alterações afetam dentro de qualquer signo encapsulado e pode extrapolar o limite da aplicação. Um exemplo de uso é o caso da gravação de macros. Um usuário experiente pode analisar o código gerado a partir da gravação de uma macro para fazer algumas alterações necessárias. O problema aqui é que é difícil, porém necessário, controlar a preservação dos significados associados aos signos impermeáveis que constituem a identidade da aplicação. As alterações do tipo VII podem, ainda, ser associadas ao conjunto de itens e regras do tipo léxico, sintático e semântico (VIIa) ou a todos os tipos (VIIb).

Como será visto no próximo capítulo, utilizamos uma classificação semelhante à de Souza e Barbosa (2006), na qual encaixamos as técnicas e mecanismos de extensão investigados.

2.3. Como Obter as Informações Necessárias para Definir as Adaptações?

Em qualquer projeto que envolva o design de um sistema, seja ele uma nova proposta de design para um sistema existente ou uma proposta inovadora para um sistema sem precedentes, há a necessidade de estabelecer um conjunto de requisitos iniciais, como o objetivo do projeto, as necessidades dos usuários e suas expectativas, e as restrições de condições sob as quais o sistema irá atuar.

Há diversas formas e metodologias para o levantamento dos requisitos e dados iniciais. Escolhemos, para este trabalho, a análise de usuários e tarefas para

o design de interface, proposto por Hackos & Redish (1998). A metodologia proposta pelas autoras está dividida em três partes: a análise dos usuários, a análise das tarefas e a análise do ambiente.

A partir das análises propostas pelas autoras, conseguimos definir as seguintes características:

- preferências;
- tarefas, objetivos e interesses;
- conhecimento sobre o domínio;
- experiência; e
- background.

2.3.1. Técnicas de coletas de dados

O objetivo deste trabalho não é definir a maneira como o designer fará a coleta de dados. Estamos interessados apenas no resultado da análise, que pode ser realizada utilizando-se diversas técnicas de coleta de dados. A seguir, apresentamos um quadro comparativo das técnicas usadas na identificação de requisitos, de acordo com suas vantagens e desvantagens (Preece et al., 2005).

Técnica	Boa para	Tipo de dados	Vantagens	Desvantagens
Questionários	Responder a questões específicas	Dados qualitativos e quantitativos	Pode atingir várias pessoas com poucos recursos	O design é crucial. O índice de resposta pode ser baixo. As respostas podem não ser o que você deseja.
Entrevistas	Explorar questões	Alguns dados quantitativos, mas mais qualitativos	O entrevistador pode guiar o entrevistado se necessário. Encoraja o contato entre desenvolvedores e usuários.	Requer tempo. Ambientes artificiais podem intimidar o entrevistado.
Grupos de foco e workshops	Coletar vários pontos de vista	Alguns dados quantitativos, mas mais qualitativos	Ressalta áreas de consenso e conflito entre desenvolvedores e usuários.	Possibilidade de dominarem certos tipos de personalidade.
Observação natural	Entender o contexto da atividade do usuário	Qualitativo	Observar o trabalho real oferece percepções que outras técnicas não podem oferecer.	Requer muito tempo. Grandes quantidades de dados.

Estudo de documentação	Aprender sobre procedimentos, regulamentações e padrões	Quantitativo	Não compromete o tempo dos usuários.	O trabalho diário será diferente dos procedimentos documentados.

Tabela 6: Técnicas de coleta de dados para identificação de requisitos (Preece et al., 2005).

2.3.2. Análise de usuários e tarefas

O estudo apresentado nesta seção foi realizado a partir de Hackos & Redish (1998), onde as autoras descrevem o resultado de alguns anos de experiência trabalhando com análise de usuários.

Análise de usuários

A primeira fase da análise envolve um estudo sobre os usuários. Para isto, antes de conversar com eles, é importante que a equipe de trabalho seja selecionada de maneira a contribuir com suas próprias experiências. Desta forma, é interessante que a equipe seja formada por pessoas que tenham contato regularmente com os usuários. Depois, toda a equipe passa por uma etapa de *brainstorm*, para discutir quem serão os usuários, tentando organizar as pessoas escolhidas de acordo com suas características e/ou tarefas. É importante que as características principais dos usuários sejam escolhidas, por isto, a equipe será encarregada de discutir qual a melhor forma para testar e separar tais características.

O intuito desta etapa é confirmar suspeitas sobre os usuários, ou até mesmo definir novas intuições sobre eles. Para este trabalho, estamos interessados no resultado da análise e não na metodologia usada para tal. Desta forma, a maneira como será realizada a análise com os usuários, seja ela através de entrevistas, questionários, visitas ao local de trabalho etc., fica a cargo da equipe de pesquisa e desenvolvimento. Portanto, analisaremos agora apenas os pontos interessantes desta análise. Há três categorias principais de perguntas relacionadas aos usuários:

1. Como eles se definem?
    - emprego, tarefas, ferramentas e modelos mentais
2. Como eles diferem individualmente?
    - características pessoais, físicas e culturais e motivação

3. Como eles usam os produtos no decorrer do tempo e quais são as escolhas que eles fazem de acordo com os níveis de aprendizado que desejam/precisam obter?
    - estágios de uso

Este último ponto é referenciado juntamente com a análise de tarefas. O primeiro ponto da análise fornece insumos para um modelo inicial sobre como os usuários poderão se comportar em relação ao sistema que se deseja (re-) construir.

As características dos usuários ajudam a entender o quanto eles se engajarão no aprendizado do sistema e até mesmo como eles irão lidar com a interface, documentação e treinamento. Os autores citam algumas questões a serem consideradas:

- *título do cargo*: quais são as diferenças (afetam as habilidades e responsabilidades?), o conteúdo, as responsabilidades e as tarefas;
- *cargos*: se são definidos pela organização, pelo treinamento profissional ou pela educação;
- *aprendizado do trabalho*: na escola, por treinamento específico, no dia-a-dia do local de trabalho;
- *comportamento*: como classificam a responsabilidade de cada tarefa, como se dá a relação com os outros colegas de trabalho e pessoas de cargo superior e como lidam com clientes; e
- *workflow*: o quanto sabem sobre as tarefas anteriores ou posteriores às que exerce no *workflow*.

Sobre a função, ou seja, os papés designados para cada usuário de acordo com as tarefas que eles realizam em seus cargos, podemos definir as seguintes perguntas:

1. Quais são as diferenças entre os cargos de acordo com o conteúdo, responsabilidade de cada usuário e as tarefas a serem executadas?
2. Os cargos são definidos pela organização ou de acordo com o treinamento recebido?
3. Como os usuários aprendem o trabalho a ser realizado: treinamento específico ou no dia-a-dia, sozinho ou com colegas de trabalho?

4. Como os usuários classificam a responsabilidade de cada tarefa e como ocorre o relacionamento entre as tarefas dos outros colegas e pessoas de cargo superior?
5. O quanto sabem sobre as tarefas anteriores e posteriores do *workflow*?

Os pontos anteriores não dizem muito sobre o que os usuários já sabem sobre as tarefas que o sistema irá apoiar e até mesmo os diferentes níveis de conhecimento sobre elas. É necessário entender o que eles sabem sobre as tarefas e o quanto de experiência possuem. Desta forma, é preciso saber, entre outras coisas, algumas questões relacionadas às tarefas, como:

- *aprendizado e realização*: como eles aprenderam o que fazem hoje, ocorreu mudança na realização da tarefa, como é o caminho percorrido por um usuário novato (seria interessante ter este usuário para analisar);
- *tarefas x tempo*: mudanças no decorrer do tempo, quanto tempo estão realizando a mesma tarefa;
- *variações*: várias em um dia ou poucas várias vezes ao dia;
- *ensinamento*: ensinam outras pessoas como realizar tarefas iguais ou semelhantes, há algum supervisor; e
- *conhecimento*: quem ou o que é procurado quando ocorre algo inesperado.

Algumas perguntas sobre o aprendizado e realização das tarefas são:

1. Eles gostam de se arriscar e explorar novas formas de fazer o mesmo trabalho? Eles evitam novas experiências, preferindo o que já foi testado e funcionou? Eles aprendem "brincando" com os produtos? Eles querem alguém que os mostre como fazer cada passo enquanto aprendem algo novo?
2. O que eles trazem consigo em suas mentes para executar as tarefas que o trabalho exige?
3. Como os usuários aprenderam a executar as tarefas que realizam no trabalho?
4. Há quanto tempo eles realizam essas tarefas? As tarefas têm mudado ao longo do tempo?

5. Eles executam as tarefas hoje da mesma forma que executavam no passado?
6. Os usuários ensinam outros como realizar as mesmas tarefas? Eles supervisionam o trabalho de alguém?
7. Quem é considerado especialista na empresa? Para quem todos pedem ajuda quando há algo errado?

Essas questões não dizem nada sobre o conhecimento e experiências dos usuários com as ferramentas usadas para realizarem suas tarefas. É importante saber como os usuários se relacionam com sistemas semelhantes aos que se pretende desenvolver e até mesmo descobrir se há outras formas de realizar a mesma tarefa, ou seja, se eles podem utilizar mais de uma ferramenta e como ocorre a escolha da ideal. Portanto, algumas questões interessantes são:

- *ferramentas* atuais: quais são, usam sempre as mesmas (evolução de versões), de mesmos fabricantes;
- *aprendizado*: como ocorreu, onde (escola, treinamento, próprio trabalho, por conta própria);
- *confiança*: gostam e se sentem confortáveis usando, se consideram especialistas, iniciantes;
- *familiaridade com a tecnologia*: sentem-se familiarizados com a tecnologia ou design semelhante ao que se pretende desenvolver, usam computadores, gostam/têm de usar quais sistemas operacionais, quais são os dispositivos utilizados freqüentemente (mouse, teclado etc.); e
- *ferramentas X tarefas*: como se relacionam as ferramentas e as tarefas, como se dá a alteração das tarefas de acordo com a alteração das ferramentas (não há, há alterações significativas etc.).

Sobre o uso de ferramentas, podemos destacar as seguintes perguntas:

1. Que ferramentas os usuários usam hoje para executar suas tarefas? Eles sempre usaram as mesmas ferramentas ou eles estão familiarizados com diferentes versões das ferramentas, às vezes de fabricantes diferentes?

2. Como eles aprenderam a usar essas ferramentas? Na escola? Em treinamento? No trabalho? Por conta própria?
3. Eles estão confortáveis com as ferramentas?
4. Eles se consideram especialistas ou iniciantes?
5. Em que nível as ferramentas que eles usam definem o que eles fazem? Se as ferramentas mudarem, o trabalho deles também irá mudar significativamente?

Segundo a familiaridade com a tecnologia, ou seja, o nível de habilidade com computadores, podemos destacar as seguintes perguntas:

1. Que ferramentas eles sabem usar? Eles são datilógrafos ou lentos ao teclado? Eles se cercam de tecnologia de ponta ou não desejam isto? Eles estão acostumados a preencher formulários em papel ou estão familiarizados com computadores para negócios padrão ou atividades domésticas?
2. Que habilidades técnicas eles trazem para executar seu trabalho? Eles são técnicos experientes que sabem consertar qualquer coisa que vêem? Eles já usaram um mouse? Uma aplicação Windows? Um sistema operacional específico?
3. Qual é sua experiência prévia em usar ferramentas e interfaces similares?
4. Eles estão familiarizados com a tecnologia que é similar ao design pretendido? Eles conhecem PCs ou *mainframes*? Windows ou Mac? Mouse ou teclado?

Quanto mais conhecemos sobre a experiência dos usuários relacionadas a interação com outras ferramentas, mais saberemos como acomodar seu atual conhecimento e experiência no design do novo produto.

Além dos três pontos citados anteriormente, precisamos saber sobre o modelo mental e vocabulário dos usuários. O modelo mental nada mais é do que o entendimento sobre como aplicar o conhecimento e as experiências nas tarefas a serem executadas. As pessoas usam seu modelo mental para relacionar informações sobre o que estão aprendendo com o que já conhecem. Se conseguirmos o máximo de entendimento sobre o modelo mental dos usuários, estaremos mais próximos de uma interface de fácil entendimento e aprendizado.

O importante aqui é prestar atenção em como os usuários descrevem suas tarefas, qual o vocabulário usado por eles para definir termos técnicos, seqüência de ações, funções, e até mesmo erros. Os erros indicam que o modelo mental dos usuários não condiz com o modelo conceitual interpretado pelos designers no momento de concepção do sistema.

A segunda categoria apontada no início desta seção indica que as diferenças entre os usuários também são importantes e interferem no design do sistema. É importante saber se os usuários do sistema possuem alguma característica física que os impeça de usar determinado sistema, ou se eles utilizam o sistema porque gostam ou porque são obrigados, ou casos semelhantes. Deve-se levar em conta se o sistema deve focar um determinado grupo de usuários ou se deve ser desenhado de forma abrangente, incluindo diversos tipos de usuários.

Desta forma, procuramos saber sobre:

- *características pessoais*: como ocorre o aprendizado (leitura, aulas, tutoriais, perguntando a outras pessoas, tentativa e erro), como se dá o uso da tecnologia de maneira geral (preferem usar teclas de atalho e raramente retirar as mãos do teclado, preferem usar o mouse, preferem digitar comandos ao invés de memorizar o local dos mesmos nos menus);
- *diferenças físicas*: incapacidades físicas que restringem o movimento, problemas de visão (daltonismo, cores semelhantes se tornam invisíveis ou indistinguíveis), incapacidades geradas pela idade (distinção de pequenos objetos ou leitura de fontes pequenas);
- *diferenças culturais*: idioma, idade e culturas corporativas; e
- *diferenças motivacionais*: de quem é a idéia do novo design, qual o envolvimento dos usuários na tomada de decisão, os usuários parecem agressivos ou provocativos.

Sobre as características pessoais, destacamos as seguintes perguntas:

1. Quão bem eles lêem? Eles evitam aprender com informações escritas, preferindo perguntar para outras pessoas? Eles são graduados com experiência de usar textos complexos? Eles desejam ler texto quando usam produtos como o que está sendo projetado?

2. Quais são suas características individuais que podem afetar seu comportamento com o software ou com a informação que projetamos?

Para descobrir sobre as diferenças motivacionais, podemos destacar as seguintes perguntas:

1. Que razões eles têm para usar o produto?
2. O que os motiva a fazer seu trabalho? Eles estão fazendo suas tarefas por dinheiro? Estão no ambiente de trabalho para interagir socialmente? Estão tentando mudar de gerência ou profissão?
3. Que valores eles trazem para o trabalho? Eles são aprendizes entusiasmados? Eles esperam que sua interação com a interface seja divertida, não entediante? Eles estão interessados em economizar dinheiro, economizar tempo, se tornar especialistas, ter um trabalho fácil de fazer?

As diferenças culturais, que podem ocorrer dentro de um mesmo país, podem afetar na escolha de determinadas metáforas. Por exemplo, algumas imagens usadas em ícones podem ter interpretações diferentes de acordo com as culturas. No Brasil, a cada cidade ou região temos diferentes usos para uma mesma palavra. Truta pode ser usada para referenciar um peixe da família do salmão. Em algumas regiões do Brasil, ela é usada como gíria, para expressar um amigo, camarada. Em Portugal, ela se refere a uma pessoas malandra, que não tem o que fazer. A escolha dos itens da interface e das metáforas deve estar livre de detalhes culturais, para que não haja problemas na interpretação, a menos que o sistema a ser desenvolvido seja destinado a um grupo de pessoas com características culturais muito próximas.

Sobre essas diferenças, podemos destacar as seguintes perguntas:

1. Que línguas eles se sentem confortáveis em usar?
2. Eles usam sua língua fluentemente ou só falam línguas com as quais você não está familiarizado?
3. Eles têm um vocabulário profissional específico, um vocabulário derivado de suas empresas ou trabalhos, ou um vocabulário de um grupo social relevante para o seu trabalho?

Análise de tarefas

A segunda fase da análise envolve questões sobre as tarefas. Queremos saber como os usuários realizam suas tarefas atualmente, quais os principais problemas que eles encontram, como simplificar algumas tarefas consideradas exaustivas, como relacionar as tarefas realizadas por grupos de pessoas etc. Esta etapa envolve responder as seguintes perguntas, entre outras:

1. O que os usuários fazem? Que metas eles estão tentando atingir? Que tarefas eles fazem hoje para atingir essas metas? Que tarefas o seu produto irá ajudá-los a fazer? Como eles realmente executam as tarefas hoje? Que problemas eles têm executando essas tarefas? Como você pode simplificar o que os usuários fazem de forma que eles possam atingir suas metas mais facilmente? Como as tarefas que uma pessoa faz estão relacionadas com a tarefas que outras fazem para atingir uma porção do trabalho?
2. Como os usuários diferem em termos das tarefas que realizam ou da forma que usam um produto em virtude do tempo em que o estão utilizando, com que freqüência usam, o quão confortável estão – ou seja, qual é a diferença entre iniciantes e especialistas e quais são os outros níveis de uso entre esses dois?
3. Como o trabalho é feito quando várias pessoas estão envolvidas?
4. O que um único indivíduo faz durante o dia, semana ou mês?
5. Que tarefas são realizadas por todas as pessoas que deverão usar seu produto?
6. Qual é a ordem em que os usuários fazem as tarefas?
7. Como uma tarefa grande é decomposta em sub-tarefas?
8. Que passos e decisões os usuários tomam para completar uma tarefa ou parte de uma tarefa?

Para obter as respostas a essas perguntas, é necessário considerar três tópicos: os objetivos dos usuários, identificação de diferentes tipos e níveis de análise de tarefas e como classificar os usuários de acordo com os estágios de uso dos sistemas.

Os autores citam várias formas de análise de tarefas, dentre elas, a análise do *workflow* e do trabalho, a combinação entre essas análises, listas de tarefas (*task lists*), análise de processo, seqüência e hierarquia de tarefas e análise

procedural. Novamente vale ressaltar que o interesse deste trabalho não está na forma de análise escolhida pela equipe de desenvolvimento e sim no resultado da análise.

A primeira etapa desta fase é definir os objetivos dos usuários e, com isto, entender como os usuários passam dos objetivos para as tarefas e destas para as ações. Um objetivo é uma noção sobre o que se quer/é (Norman, 1988). Segundo Norman, as pessoas executam um ciclo de sete passos para atingir um objetivo (Figura 25).

1. *Formulação do objetivo*
2. *Formulação da intenção*
3. *Especificação da ação*
4. *Execução da ação*
5. *Percepção sobre o estado do sistema*
6. *Interpretação do estado do sistema*
7. *Avaliação da intenção*

Figura 25: Ciclo de sete estágios do Norman – como as pessoas se comportam ao tentar atingir objetivos e executar de tarefas

A partir deste ciclo, estamos interessados em saber o que realmente importa para o usuário (baixo custo, confiança, fácil aprendizado, tempo etc.). É importante lembrar que um usuário pode mudar a tarefa, o objetivo ou até mesmo desistir. Além disto, caso o caminho para atingir o objetivo seja frustrante, ele pode acabar desistindo de completar a tarefa. Portanto, devem-se conhecer os limites de tempo e esforço para diferentes tarefas e tecnologias e isto deve ser feito na análise de tarefas.

A seguir, descrevemos o que se espera obter como resultado em cada tipo de análise de tarefa citado anteriormente.

Na análise de *workflow*, deve se levar em conta o processo de trabalho atual para identificar pontos redundantes ou desnecessários. É preciso ter em mente que as pessoas envolvidas no processo podem estar situadas em diferentes locais dentro da empresa, ou até mesmo fora dela. Além disto, o resultado do trabalho de uma pessoa pode ser o ponto de entrada para o trabalho de outra pessoa. Por isto, devem-se analisar tanto as pessoas envolvidas no ponto de entrada do processo,

quanto as pessoas envolvidas no ponto de saída. Esta análise identifica quem faz o quê no processo de trabalho, incluindo o entendimento sobre os objetivos de cada um, especialmente com relação à tecnologia usada ou às possíveis mudanças a serem realizadas.

Na análise do cargo (onde ocorre a identificação de todo o trabalho que uma determinada pessoa realiza ocupando determinado cargo na empresa) podemos **encontrar** novas oportunidades de desenvolvimento para facilitar o trabalho, **entender** funcionalidades específicas para acoplar ao sistema e **aprender** sobre as pressões que essas pessoas sofrem com relação ao trabalho e quais são seus valores. Esta análise requer que o pesquisador observe o usuário (e pergunte o que ele está fazendo) enquanto ele realiza suas tarefas em seu local de trabalho. Ao final da análise, devemos saber:

- a freqüência com que eles realizam cada tarefa;
- como ordenar criticamente cada tarefa de acordo com sua importância;
- quanto tempo o usuário leva para completar a tarefa;
- a dificuldade em realizar determinada tarefa; e
- a divisão de responsabilidade sobre as tarefas (todos da empresa realizam as mesmas tarefas, diferentes pessoas do mesmo cargo etc.).

A lista de tarefas nada mais é do que um inventário de todas as tarefas que os usuários precisarão realizar em toda a extensão do seu sistema. Começando por tarefas "globais" como "escrever uma mensagem", vamos quebrando-as em partes para obter pequenas unidades, como ícones de interface ou itens de menu. É importante lembrar que a lista de tarefas não diz como os usuários devem realizar a tarefa, mas o que eles precisam para realizá-la. O "como" realizar a tarefa é justamente o problema de design que queremos resolver. A lista ideal teria todas as tarefas que todos os tipos de usuários precisariam realizar, e as tarefas seriam nomeadas de acordo com o ponto de vista de cada usuário, e na linguagem de cada usuário.

A análise de processos e seqüência de tarefas é a ordenação de determinadas tarefas da lista, ou seja, a seqüência mostra quais e em qual ordem determinadas tarefas devem ser executadas para alcançar determinado objetivo. Note que nem

sempre usuários diferentes executarão as mesmas tarefas na mesma ordem para atingir determinados objetivos. É importante saber quais são essas diferenças e por quê elas ocorrem. O ideal é que o sistema seja tão flexível ao ponto de acomodar essas diferentes vias que levam a um mesmo lugar. Por outro lado, aqui podemos observar que determinadas vias não são tão eficientes, propondo novos caminhos para elas. Cabe lembrar que as mudanças nas seqüências devem ser feitas sempre com um bom motivo e, caso elas ocorram, deve-se pensar como ajudar os usuários nessa transição.

Na hierarquia de tarefas, as tarefas são decompostas em sub-tarefas e podemos realizar uma análise de acordo com o grau de profundidade almejado para cada tarefa. Na análise procedural, uma tarefa é escolhida para então ser dividida em outras tarefas, sendo que as limitações de uma determinada interface são levadas em conta. Em outras palavras, estamos vendo como os usuários realizam determinadas tarefas usando as ferramentas que eles possuem atualmente, entendendo tanto o processo físico – passos realizados – quanto o mental – decisões tomadas.

Todas essas formas de análise de tarefas ajudam a entender o que os usuários fazem e como eles fazem. A partir do entendimento dos objetivos dos usuários, é possível olhar o trabalho sobre diversas perspectivas e níveis, como foi mencionado anteriormente.

Segundo o padrão de uso da aplicação, ou seja, a freqüência de uso que auxilia no aprendizado do usuário, podemos destacar as seguintes perguntas:

1. O que está sendo desenvolvido está relacionado ao trabalho primário deles ou é algo que eles usam ocasionalmente?
2. Eles querem investir muito tempo aprendendo ou será uma parte menor do que eles fazem?
3. Eles estão interessados apenas em fazer a tarefa uma vez e seguir em frente?

O terceiro ponto mencionado no início deste capítulo relatava o fato dos usuários possuírem diferentes níveis de conhecimento e freqüência de uso de um determinado produto. Os autores classificam os usuários de acordo com quatro níveis: iniciantes, iniciantes avançados, competentes e *experts*. Alguns usuários passam por todas as etapas enquanto outros nunca chegam a ser *experts*. O

importante é promover, através do design, meios para facilitar a transição de um nível ao outro.

De acordo com o nível de habilidade no domínio da aplicação, podemos destacar as seguintes perguntas:

1. O que e quanto eles sabem sobre o assunto que está sendo projetado? Eles já são especialistas neste assunto? Espera-se que eles se tornem especialistas?
2. Que experiências eles tiveram fazendo trabalhos ou tarefas similares? Eles têm feito o mesmo trabalho ou tarefas durante anos? Eles estão mudando para trabalhos parecidos com os antigos? O trabalho ou tarefas sendo projetadas são algo do qual eles nunca nem ouviram falar?
3. O que eles sabem sobre o assunto e as ferramentas que usam hoje ou sobre aquelas que nós devemos apresentar em uma nova interface?

O Ambiente dos usuários

As pessoas são influenciadas pelo local onde trabalham. O espaço físico, os equipamentos disponíveis e a relação com as outras pessoas são fatores que podem alterar a maneira como as pessoas executam suas tarefas ou reagem a novas tecnologias.

Nesta etapa estamos interessados em saber como é o espaço físico, social e cultural dos usuários. Buscamos respostas para as seguintes perguntas:

1. *Físico*: Qual é o tamanho do espaço físico do ambiente de trabalho? Há possibilidade de compartilhamento de material ou cada usuário possui o seu? Há espaço para guardar manuais? Todos os equipamentos podem ser alcançados? Outros equipamentos podem ser adicionados? O ambiente é barulhento? É possível escutar, conversar ou se concentrar? Há luz suficiente para ver a tela do computador e uma documentação? Como é a temperatura, limpeza (poeira, poluição) e umidade? Há algum perigo que possa alterar a maneira como os usuários trabalham?
2. *Social*: Eles precisam realizar as tarefas rapidamente, precisamente? Sofrem alguma pressão para trabalharem de maneira rápida, sem cometer erros? Há recursos que os ajudem a resolver problemas? A

documentação é de fácil entendimento? Podem fazer ligações para obter ajuda? As pessoas que compartilham a informação dividem o mesmo local de trabalho? São separadas por departamentos? Como ocorre o compartilhamento das informações (telefone, e-mail, rede interna)? Qual é a hierarquia social? Eles podem trabalhar em casa? Há muitos trabalhando em casa? Trabalham sozinhos? Como é a relação entre os usuários e os clientes? Interagem com os clientes pessoalmente?

3. *Cultural*: As diferenças culturais influenciam como eles trabalham (rapidez na realização de alguma tarefa, valores, ambiente de trabalho)? Eles pertencem a uma cultura profissional que possui valores particulares de estilos de trabalho? Pertencem a grupos sócio-econômicos que poderiam afetar a experiência com o novo design? Possuem referencias culturais que determinam como obter informações em manuais e ajuda online?

Cada ponto mencionado acima pode refletir em tomadas de decisões diferentes para o design. Alguns poderão ser ignorados, enquanto outros precisarão ter uma atenção especial, levando em conta alternativas de design. A análise do ambiente servirá como uma afirmação das decisões de design tomadas de acordo com a análise dos usuários e das tarefas.