Padrões de Interação e Usabilidade

Eduardo Cesar Valente

Trabalho Final de Mestrado Profissional

Instituto de Computação

Universidade Estadual de Campinas

Padrões de Interação e Usabilidade

Eduardo Cesar Valente

Fevereiro de 2004

Banca Examinadora:

- Prof. Dr. Hans Kurt Edmund Liesenberg (Orientador)
  Instituto de Computação - UNICAMP
- Profª. Drª. Maria de Fátima Ridolvi Ordini Pires da Silva
  Centro de Computação - UNICAMP
- Profª. Drª. Heloisa Vieira da Rocha
  Instituto de Computação - UNICAMP
- Profª. Drª. Maria Cecília Calani Baranauskas - (suplente)
  Instituto de Computação - UNICAMP

Padrões de Interação e Usabilidade

Este exemplar corresponde à redação final do Trabalho Final devidamente corrigida e defendida por Eduardo Cesar Valente e aprovada pela Banca Examinadora

Campinas, 17 de fevereiro de 2004

Prof. Dr. Hans Kurt Edmund Liesenberg
(Orientador)

Trabalho Final apresentado ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Computação na Área de Engenharia de Software

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP

Valente, Eduardo Cesar

V234p Padrões de interação e usabilidade / Eduardo Cesar Valente -- Campinas, [S.P. :s.n.], 2004.

Orientador : Hans Kurt Edmund Liesenberg

Trabalho final (mestrado profissional) - Universidade Estadual de Campinas, Instituto de Computação.

1. Interfaces de usuário (Sistema de computador). 2. Interação homem-máquina. 3. Usuário final. 4. Padrões de projeto. I. Liesenberg, Hans Kurt Edmund. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Resumo

No projeto de uma interface de usuário, cada vez mais a atenção está se voltando ao usuário final e às suas necessidades. Com isto, ganham evidência os princípios de usabilidade, para garantir um produto que corresponda às expectativas do usuário e que traga qualidade à solução apresentada.

Por se tratar de interação entre pessoas e máquinas, a interface de usuário permite interpretações e abordagens diversas e, em algumas situações, conflitantes. Isto se traduz em degradação da qualidade do produto final e em frustração do usuário.

Neste trabalho, serão apresentados os Padrões de Projeto, uma técnica de desenvolvimento de interfaces que vem experimentando uma boa aceitação entre os desenvolvedores de software, por privilegiar a usabilidade da interface e documentar a experiência adquirida de projetos anteriores para reuso. Analisaremos os padrões de projeto sob os princípios de usabilidade, verificando qual o diferencial que a utilização de tais padrões pode trazer a um bom projeto de interface de usuário.

Abstract

On the design of a user-interface, increases the focus on the user and his needs. Because of that, the usability principles gains evidence, to assure a product that matches the user expectations and brings quality for the presented solution.

Because we're dealing with interaction between people and machines, the user-interface allows several, and sometimes, conflicting, interpretations and approaches. This results at a poor quality final product and a user frustracted.

In this work, we'll present the Design Patterns, an interface design technique that has experimented a good acceptance among software engineering comunity, enhancing the interface usability and documenting the experience acquired from earlier projects for reuse. We'll analyse the Design Patterns under usability principles, verifying what is the diferential that the use of this patterns can bring to a user-interface good design.

Conteúdo

Resumo vi
Abstract vii
Conteúdo viii
Índice de Figuras x
Índice de Tabelas xi
Introdução 01
Capítulo 1 - Padrões de Projeto 03
1.1. O que são padrões de projeto? 03
1.2. Padrões na arquitetura convencional 05
1.3. Padrões na computação 06
1.4. Padrões no mundo real 08
1.5. Proto-Padrões e Verificação de Padrões 09
1.6. Os Anti-Padrões 09
1.7. Formato de um Padrão 10
1.7.1. Forma Portland 10
1.7.2. Forma Alexandrina 11
1.7.3. Forma GoF 11
1.8. Classificação de Padrões 12
1.9. Uma Linguagem de Padrões 14
1.10. Uma Aplicação de Padrões : JUnit 15
Capítulo 2 - Interação Humano-Computador 17
2.1. O que é Interface ? 17
2.2. Quem é o usuário ? 18
2.2.1. Fatores Humanos 19
2.2.1.1. Atenção 19
2.2.1.2. Percepção e Reconhecimento 19
2.2.1.3. Memória 20
2.2.1.4. Modelos Mentais 20
2.2.1.5. Frustração dos Usuários 22
2.3. Projeto de Interação 22
2.3.1. Princípios de Projeto 23
2.3.2. Projeto Conceitual e Projeto Físico 24
2.3.3. Metáforas 24
2.3.4. Diretrizes (Guidelines) 25
2.3.4.1. Regras Áureas de Shneiderman 26
2.3.4.2. Diretrizes de Madsen 27
2.3.5. Prototipagem 28
2.4. Modelo Conceitual 29
2.4.1. Estilo de Interação 30
2.4.2. Paradigmas de Interação 31
2.5. Definição de Usabilidade 32
2.5.1. Objetivos de Usabilidade 32
2.5.2. Heurísticas de Usabilidade de Nielsen 32

Capítulo 3 - Padrões de Projeto de Interfaces de Usuário sob a Ótica da Usabilidade ... 34
3.1. O Usuário Como Centro do Projeto ... 35
3.2. Utilização de Padrões para o Projeto da Interface de Usuário ... 36
3.3. A Usabilidade nos Padrões ... 39
3.4. O Diferencial Provido Pelos Padrões ... 41
Capítulo 4 - Conclusões e Trabalhos Futuros ... 43
4.1. Interfaces Afetivas ... 43
4.2. Interfaces Vivas ... 44
4.3. Balas de Prata ... 45
4.4. QWAN ... 46
Referências ... 47
Apêndice A - Exemplos de Padrões de Projeto para IHC ... 49
A.1. The Wizard (O Assistente) ... 49
A.2. The Grid (A Grade) ... 50
A.3. Progress (Progresso) ... 51
A.4. The Shield (O Escudo) ... 52
A.5. The Preview (A Prévia) ... 53
Apêndice B - Exemplos de Anti Padrões ... 55
B.1. Culto da Carga ... 55
B.2. Bode Expiatório ... 55
B.3. Processo à Prova de Idiotas ... 56
Apêndice C - Padrões no Formato Portland ... 57
C.1. Decisão do Usuário ... 57
C.2. Tarefa ... 57
C.3. Janela de Tarefa ... 58
Apêndice D - Padrões na Forma Alexandrina ... 59
A.1. Cozinha de Fazenda ... 59
A.2. Ponto de Ônibus ... 59
A.3. Vista Zen ... 59
Apêndice E - Padrões GoF ... 60

Índice de Figuras

Figura 1. Diagrama exemplificando a utilização do padrão Bridge p/ Chaves Elétricas 08
Figura 2. Arquitetura do JUnit 16
Figura 3. Linha do Tempo do JUnit 16
Figura 4. Modelos Mentais 21
Figura 5. Modelo Execução-Avaliação 34
Figura 6. Os Três Níveis de Processamento 43

Índice de Tabelas

Tabela 1. Aglomerados de Padrões 13
Tabela 2. Pontos de Vista de Padrões 14
Tabela 3. Relação Tarefa-Diretriz 25
Tabela 4. Regras Áureas de Shneiderman 26
Tabela 5. Heurística de Usabilidade x Abordagem Com Padrões 39

Introdução

Daniel LaRusso estava com um problema: precisava fazer com que alguns valentões de sua escola parassem de o importunar. O problema aumentava ao saber que tais valentões faziam parte de uma escola de karatê, com ares nazistas. Tudo isto em uma cidadezinha no sul da Califórnia.

Um velho jardineiro japonês, chamado Sr. Miyagi, tinha a solução para este problema: aprender karatê, mas utilizando a filosofia corretamente ensinada pelos orientais, primariamente como método de defesa e não de ataque.

O método utilizado, que deu origem ao *blockbuster Karate Kid*, de 1984, será a deixa para introduzirmos o leitor na abstração conhecida como *pattern*, ou padrão.

Como típico adolescente ocidental, o personagem interpretado por Ralph Macchio tinha determinados conceitos e práticas que ditavam seus movimentos. O que o mestre Miyagi, interpretado por Pat Morita, fez foi, a partir destes conceitos, identificar um padrão de utilização de músculos e reflexos que prepariam seu pupilo para a arte marcial do karatê, sem que fosse necessária muita filosofia oriental e treinamento extensivo.

A situação que se desenvolve no filme é a de alguém reutilizando os movimentos que normalmente utilizaria para lavar e polir um carro, lixar e pintar madeira, para ações de defesa pessoal. Tirando todo o sentimentalismo que o filme traz, o que o mestre oriental fez foi identificar um padrão de movimentos e utilização de músculos e reflexos, abstrair da situação e perceber que, inserido em outro contexto, tal padrão poderia ser reutilizado de forma surpreendente.

O final do filme todos conhecem: Daniel se torna campeão de karatê e o Sr. Miyagi tem sua coleção de carros perfeitamente limpa e encerada e suas cercas impecáveis. Padrão de história com final feliz...

A abstração de determinadas situações e técnicas com fins de reutilização de recursos é prática antiga. A identificação de padrões sempre teve espaço nas mais variadas áreas do conhecimento humano, mas foi na arquitetura e, posteriormente, na computação, que tais padrões se tornaram foco de estudos mais formais.

Quando se fala que determinado criminoso segue um padrão (no jargão criminalístico M.O., ou *modus operandi*), isto é utilizado com ferramenta na solução de um crime. Quando se diz, em agricultura, que teremos o fenômeno El Niño neste ano, significa que um padrão climático irá predominar e, com ele, toda uma série de fatores que influenciarão na safra anual.

Basta observarmos um pouco e veremos que a busca por padrões é comum em toda ciência. Desde padrões climáticos até padrões sociais, a busca por tal identificação auxilia na reutilização de um conceito e na transmissão de experiências na área.

Na língua portuguesa, o termo padrão é utilizado para traduzir dois termos em Inglês que, apesar de assemelhados, têm conotação completamente diferente: *pattern* e *standard*.

Pattern é traduzido como padrão, assim como *Standard*. A diferença entre eles é que, enquanto *Standard* pode também ser traduzido como norma, a opção em Português para *Pattern* não seria outra a não ser padrão mesmo. A diferença entre elas pode ser verificada nas traduções de *standard behaviour* (comportamento padrão) e *behaviour pattern* (padrão de comportamento), coisas bem diferentes, mas com tradução aproximada.

O termo padrão, no texto que se segue, representa "algo que serve de modelo à feitura de outro" e não "aquilo que serve de norma para avaliação de algo".

Outro termo que pode causar confusão na sua tradução é *design*, já que há uma certa tendência em se traduzir para o termo desenho, como sinônimo de *layout*. A palavra em Português que será usada neste texto é projeto, representando "atividade de concepção em um determinado nível de abstração" e não "disposição visual". Tal confusão se torna mais evidente quando falamos de *web designer*, que para a maioria das pessoas é o profissional que sabe fazer um *website* agradável visualmente e não aquele que planeja como será a interação entre o usuário e o *website*.

A intenção deste trabalho é analisar como a utilização de padrões de projeto afeta o processo de desenvolvimento de interfaces de usuário, se existe melhoria ou degradação. Para isto, o texto foi dividido em quatro partes.

A primeira parte busca compreender qual é o conceito de padrões, com foco em computação, suas classificações e utilizações e, em particular, de padrões relacionados com interação homem-computador; explicar como descrever padrões, a partir de alguns exemplos da linguagem comumente utilizada.

Na seqüência, serão abordadas questões relativas às interações humano-computador: o projeto da interação, os conceitos de usuário e interface e, ainda, a conceituação do termo usabilidade de um artefato e as técnicas atualmente utilizadas para a verificação de metas de usabilidade. A intenção é dar uma panorâmica sobre como é o processo de desenvolvimento de interfaces e sobre a aplicação de princípios de usabilidade para garantir a qualidade de interfaces de usuário.

A terceira parte fará a análise sobre o quanto a idéia de aplicar padrões no desenvolvimento de interfaces de usuário pode auxiliar e agilizar tal processo. A partir de heurísticas de usabilidade, serão verificados diversos fatores que promovem a melhoria (ou a degradação) quando se utiliza padrões no desenvolvimento ou na manutenção de uma interface para um produto.

Por fim, a última parte trará a conclusão sobre quais são os prós e contras da utilização de padrões no desenvolvimento de interfaces de usuário, sugerindo ainda as melhores práticas para que tal uso venha a ser o mais efetivo possível, dentro dos paradigmas de qualidade de *software*. Será feita aqui uma avaliação dos rumos que estas técnicas estão tomando e elaborada uma previsão sobre o que pode ser feito.

Capítulo 1
Padrões de Projeto

Na arquitetura civil e posteriormente, na engenharia de *software*, padrões começaram a ser identificados e formalizados de modo que, atualmente, o uso de padrões está bem difundido na comunidade de desenvolvimento de sistemas e o catálogo de padrões encontra-se em plena expansão.

Mas... formalmente falando... o que são padrões? Existe uma definição que possa ser aplicada à imensa gama de soluções encontradas, de tal maneira que poderíamos saber se determinado padrão pode ou não ser útil em um determinado contexto, e se tal solução pode ser reutilizada por profissionais com menos experiência do que a de quem elaborou tal padrão?

A resposta é sim. No decorrer deste capítulo, os padrões serão definidos, classificados e detalhados. A intenção é mostrar que a utilização de padrões é realmente uma técnica que traz melhoria a um projeto, acelerando seu desenvolvimento e aumentando sua qualidade.

1.1. O que são padrões de projeto?

Imaginemos uma situação onde uma equipe de desenvolvimento depara-se com um problema em um projeto, onde a solução parece distante. O que se verifica é uma certa frustração nos usuários quando, após entrarem com a identificação do cliente, precisam esperar um tempo aparentemente longo demais até que todas as dívidas pendentes aparecessem para pagamento, com seus juros devidamente calculados.

A longa espera é, na verdade, um período variável de três a dez segundos, tempo para que o banco de dados fosse consultado e os cálculos fossem feitos. O tempo seria um pouco mais longo nos momentos de maior movimento, mas não era tão demorado assim, se pensarmos na quantidade de informação a ser manipulada.

Mesmo assim, a reclamação dos usuários persiste... As opções? Trazer algo menos que todas as dívidas pendentes do cliente está descartado. Todo o trabalho de otimização - *tuning* - das consultas ao banco de dados está feito. É extremamente custoso buscar toda aquela informação no meio de tantos clientes... Seria necessário considerar um servidor mais potente e até uma rede que permitisse um tráfego maior de informações.

A solução encontrada é, ao mesmo tempo, simples, eficiente e barata: logo após a entrada do código do cliente, exiba uma tela de "Aguarde, por favor". Ou algo semelhante...

Não é necessário modificar nenhum outro código, nenhum *hardware* adicionado, nada incrementado além da tela simples de aguarde. O resultado? Os usuários perguntavam: "Uau! Como ficou rápido!" ou ainda "Agora sim, está bom!".

Apesar de o tempo de espera continuar o mesmo, a sensação entre os usuários era de que o sistema estava muito mais rápido. O que acontecera?

A resposta será dada a seguir, na forma de descrição de um padrão de projeto:

Padrão: "Aguarde Por Favor"

Problema: Os usuários tendem a impacientar-se quando o tempo de espera por uma requisição é grande. O problema tende a se agravar quando há alguém por perto cobrando um resultado e a expectativa é de retorno instantâneo. Se a situação for de pressão, como por exemplo, um caixa com uma longa fila de clientes, o usuário desiste do pedido anterior e refaz o pedido, tentando clicar no botão repetidas vezes, como se isto bastasse para ser atendido mais rapidamente.

Solução: Dê um retorno ao usuário, na forma de uma tela bem evidente ou ainda de mensagens visuais do tipo 'Aguarde por Favor', barras de progresso, animações, etc. Ao mesmo tempo, bloqueie os elementos que possam gerar mais eventos de requisição.

Contexto: Isto é mais indicado para procedimentos intensivos em termos de busca de dados ou para procedimentos demorados. Para os intensivos, a tela de aguarde dá mais resultado. Para os demorados, uma barra de progresso é de mais valia.

Sabendo que a ocorrência deste tipo de problema é razoavelmente comum, a intenção era, de alguma forma, documentar esta boa solução para que pudesse ser reutilizada em problemas semelhantes que surgissem posteriormente.

Isto é, em suma, um padrão: Representar uma solução para um problema dentro de um contexto de forma a poder reutilizá-la posteriormente. No exemplo, poderíamos ainda adicionar mais cláusulas, como por exemplo, situações onde este padrão não se aplicaria, padrões semelhantes, etc.

Richard Gabriel [Gabriel, 1996] tem a seguinte definição de padrões: "*Cada padrão é uma regra de três partes, que expressa uma relação entre um certo contexto, um certo sistema de forças recorrentes dentro deste contexto, e uma certa configuração de software que possibilita tais forças a se resolverem"*

No livro *Padrões de Projeto* [GoF, 1995], de Erich Gamma: *"Um padrão de projeto sistematicamente nomeia, motiva e explica uma solução de projeto geral, que trata um problema recorrente de projeto em sistemas orientados a objetos. Ele descreve o problema, a solução, quando aplicar a solução e suas conseqüências. Também dá sugestões e exemplos de implementação. A solução é um arranjo genérico de objetos e classes que resolve o problema. A solução é customizada e implementada para resolver o problema em um contexto particular."*

A definição acima trata padrões no contexto de programação orientada a objetos mas, com uma pequena adaptação, pode ser aplicada a qualquer solução no mundo real.

Podemos definir padrões como sendo uma solução para um problema comumente encontrado que, devidamente abstraída e contextualizada, poderá solucionar um problema específico conceitualmente similar ao descrito no padrão.

A estrutura de um padrão, na sua forma mais básica, consiste de três elementos: problema, solução e contexto. Dentro das diversas áreas onde os padrões de projeto são utilizados, podemos ter mais seções, como veremos adiante; para o momento, nossos padrões terão somente estes três elementos.

A teoria de aprendizado do conectivismo de Thorndike[1] sugere que a transferência de aprendizado depende da existência de elementos idênticos na situação nova como na original. Um padrão, neste contexto, é uma ótima técnica de transferência de aprendizado por fazer com que, por meio da abstração da situação, os elementos sejam idênticos. Com isto, os padrões de projeto ganham uma característica desejável e interessante: eles são reutilizáveis e abstratos o suficiente para se aterem ao conceito, não ao elemento físico.

1.2. Padrões na arquitetura convencional

O primeiro a utilizar o termo padrão como é utilizado atualmente foi Christopher Alexander, um arquiteto que publicou vários livros, sendo os dois mais importantes *A Pattern Language: Towns, Buildings, Constructions* [Alexander, 1977], e *The Timeless Way of Building* [Alexander, 1979].

Nestes livros, onde o tema principal é arquitetura e urbanismo, Alexander propõe a criação de estruturas que são boas para as pessoas e influem positivamente nelas, melhorando seu conforto e qualidade de vida. Padrões são propostos como idéias atemporais de projeto. Para concretizar esta atemporalidade, Alexander propõe um paradigma de arquitetura baseado em três conceitos: A Qualidade, O Portal e O Caminho.

A Qualidade ("A Qualidade Sem Um Nome") - ou ainda QWAN - *Quality Without A Name* - é a essência de todas as coisas viventes e úteis que dá a elas qualidades como: liberdade, integralidade, completude, conforto, harmonia, habitabilidade, durabilidade, abertura, resiliência[2], variabilidade e adaptabilidade. É o que faz nos sentirmos "vivos" e "saciados", nos dá satisfação e melhora a condição humana.

O Portal - é o mecanismo que nos possibilita alcançar A Qualidade. Ele se manifesta como uma liguagem viva de padrões que nos permite criar projetos de diversas formas que satisfaçam diferentes necessidades. É o "Éter" universal de padrões e seus relacionamentos que permeiam um determinado domínio. O Portal é o duto para A Qualidade.

O Caminho - por meio do Caminho, padrões advindos do Portal são aplicados usando a técnica de diferenciação de espaços em uma seqüência ordenada de crescimentos, evoluindo progressivamente de uma arquitetura inicial, até um projeto "vivo" que possui A Qualidade. Seguindo O Caminho, alguém pode atravessar O Portal e alcançar A Qualidade.

Percebe-se que a descrição de padrões em arquitetura é um tanto fora do convencional e daí, do livro *The Timeless Way of Building* [Alexander, 1979], vem a primeira definição do que é um padrão:

---

[1] Edward L. Thorndike, Animal Intelligence; Transaction Pub; 1911, Reprint edition (January 2000)

[2] resiliência: do Lat.*resilientia*, *resilire*, recusar, voltar atrás; s. f., Mecân., capacidade de resistência ao choque de um material

"Um padrão é uma regra em três partes, que expressa uma relação entre um certo contexto, um problema e uma solução.

Como um elemento no mundo, cada padrão é uma relação entre um determinado contexto, um determinado sistema de forças que ocorrem repetidamente dentro deste contexto, e uma certa configuração espacial que possibilita às forças se resolverem.

Como um elemento de linguagem, um padrão é uma instrução, que mostra como esta configuração espacial pode ser utilizada, muitas e muitas vezes, para resolver o tal sistema de forças, onde quer que o contexto evidencie relevância.

O padrão é, em resumo, ao mesmo tempo uma entidade, que acontece no mundo, e a regra que nos diz como criar tal entidade, e quando nós devemos criá-la. Ele é tanto um processo quanto uma entidade; ao mesmo tempo a descrição de uma entidade que vive, e uma descrição do processo que gerou tal entidade.".

Alexander define [Alexander, 1979] o termo **geratividade**. Padrões gerativos (ou geradores) são aqueles que não são simplesmente uma descrição de algo recorrente, mas uma ferramenta para reproduzir tal recorrência.

A idéia dos padrões na arquitetura convencional é muito voltada para o que podemos chamar emulação de vida. É a idéia de que algo que se constrói deve evoluir e adaptar-se às mudanças que o ambiente, ou o contexto, impõe.

Ainda, em *The Timeless Way of Building* [Alexander, 1979]: "*Os padrões em nossas mentes são, mais ou menos, imagens mentais dos padrões do mundo: eles são representações abstratas de regras morfológicas que definem padrões no mundo. Entretanto, num quesito eles são diferentes. Os padrões no mundo meramente existem. Mas os mesmos padrões em nossas mentes são dinâmicos. Eles têm força. Eles são gerativos. Eles nos dizem o que fazer; como podemos e como devemos fazer; e nos contam também, sob certas circunstâncias, que nós devemos criá-los. Cada padrão é uma regra que descreve o que deve ser feito para gerar a entidade por ele definida.".*

1.3. Padrões na computação

Os padrões de projeto na computação já nasceram orientados a objeto. Talvez pela idéia da dualidade abstração/entidade, o paradigma mais próximo e onde seria mais simples de se identificar "vida" era realmente o de orientação a objetos.

O grande impulso para a utilização de padrões de projeto na computação foi dado com o livro *Padrões de Projeto* [GoF, 1995], de Erich Gamma, Richard Helm, Ralph Johnson e John Vlissides, em 1995. Neste livro se encontra o primeiro catálogo de padrões de projeto, num total de 23 padrões, divididos em padrões de criação, padrões estruturais e padrões comportamentais. Tal publicação é de tamanha importância para a comunidade de engenharia de *software* que o grupo ficou conhecido como a Gangue dos Quatro (em Inglês, *Gang of Four*, ou simplesmente GoF).

A noção de padrões foi ligeiramente modificada, para condizer com o universo mais abstrato do *software*. No livro, Erich Gamma et al. citam quatro diferenças fundamentais para se entender o conceito de padrões de projeto para sistemas de *software*:

"1. Os homens têm construído edificações há milhares de anos, e nessa área existem muitos exemplos clássicos em que se basear. Em contrapartida, estamos construindo sistemas de software há pouco tempo e poucos sistemas são considerados clássicos.

2. Alexander dá uma ordenação segundo a qual seus padrões devem ser usados; nós não.

3. Os padrões de Alexander enfatizam os problemas de que tratam, enquanto nossos padrões de projeto descrevem as soluções com mais detalhes.

4. Alexander afirma que o uso dos seus padrões gera edificações completas. Não afirmamos que os nossos padrões geram programas completos."

Apesar de sua compreensão ser muito facilitada pelo conceito de orientação a objetos, não é somente nesta área que os padrões de projeto podem ser identificados e classificados. Como exemplo, há um padrão em uma estrutura de banco de dados razoavelmente comum, que iremos chamar de Contrato-E-Parcela. A idéia é que temos duas tabelas: contrato e parcela, onde na tabela contrato se encontram o código do cliente que comprou, o valor total da compra, etc. enquanto que na tabela parcela se encontra a informação pertinente ao parcelamento, como vencimentos, valores, datas de pagamentos, etc.

Esta estrutura é tão simples como potencialmente danosa para o desempenho em um banco de dados, quando são manipulados milhões de registros. A situação comum que reflete a queda do desempenho é o cálculo do saldo devedor. Como a informação sobre os pagamentos está em uma tabela e a identificação sobre o cliente está em outra temos que, no momento da definição da estratégia de busca, buscar todas as dívidas do mesmo, inclusive as pagas. Quando um cliente tem um histórico de muitas compras, a diferença de tempo no cálculo aumenta bastante, gerando uma degradação sensível no desempenho geral do banco de dados.

A solução é incluir alguma informação na tabela contrato, sobre quantas parcelas ainda estão em aberto. Esta solução simples, que pode ser descrita em forma de padrão, melhora o desempenho sensivelmente e reduz a influência do tamanho do histórico de compras do cliente.

Tal padrão é baseado estritamente nos mecanismos de tratamento de consultas SQL[3] a banco de dados relacionais e não tem absolutamente a necessidade de se referir às tabelas como objetos e classes. Em várias aplicações nos sistemas de *software*, encontramos problemas recorrentes que podem ser solucionados através de padrões de projeto.

Mesmo com todas as diferenças apresentadas, os padrões na computação ainda têm fortemente embasada a idéia de que são ainda uma regra em três partes. Pela própria abstração inerente ao projeto de *software*, os padrões raramente bastam por si só. Mas, como técnica de descrição de um problema e sua respectiva solução, os padrões de projeto são provavelmente uma das mais poderosas técnicas de auxílio à criação de sistemas eficientes, estáveis e de rápido desenvolvimento.

[3] SQL - Structured Query Language : linguagem estruturada de consulta a bancos de dados

1.4. Padrões no mundo real

Como já discorrido anteriormente no texto, é relativamente freqüente encontrarmos padrões no mundo real, desde os padrões climáticos aos de comportamento. No entanto, o uso que será feito aqui de padrões no mundo real será o de eliminarmos um pouco a abstração de um padrão, para que seja mais fácil a sua compreensão e aumente o poder comunicativo da linguagem de padrões.

Padrões de projeto são utilizados para discutirmos estruturas maiores que módulos, procedimentos e objetos. Para isto, temos como elemento crucial de uma linguagem a imagem mental associada a um determinado símbolo ou abstração. É necessário que exemplos concretos de um padrão sejam evidenciados, tanto para a melhor comunicação entre uma equipe, como para o aprendizado de novos desenvolvedores.

Dentro desta filosofia, temos a clonagem como exemplo do padrão *Prototype* e a presidência de um país como exemplo do padrão *Singleton*. Temos os interruptores e disjuntores como exemplo do padrão *Bridge* e ainda as operações aritméticas como exemplos do padrão *Composite*. Alguns exemplos, como o da Figura 1, podem ser encontrados em [Duell, 1997]

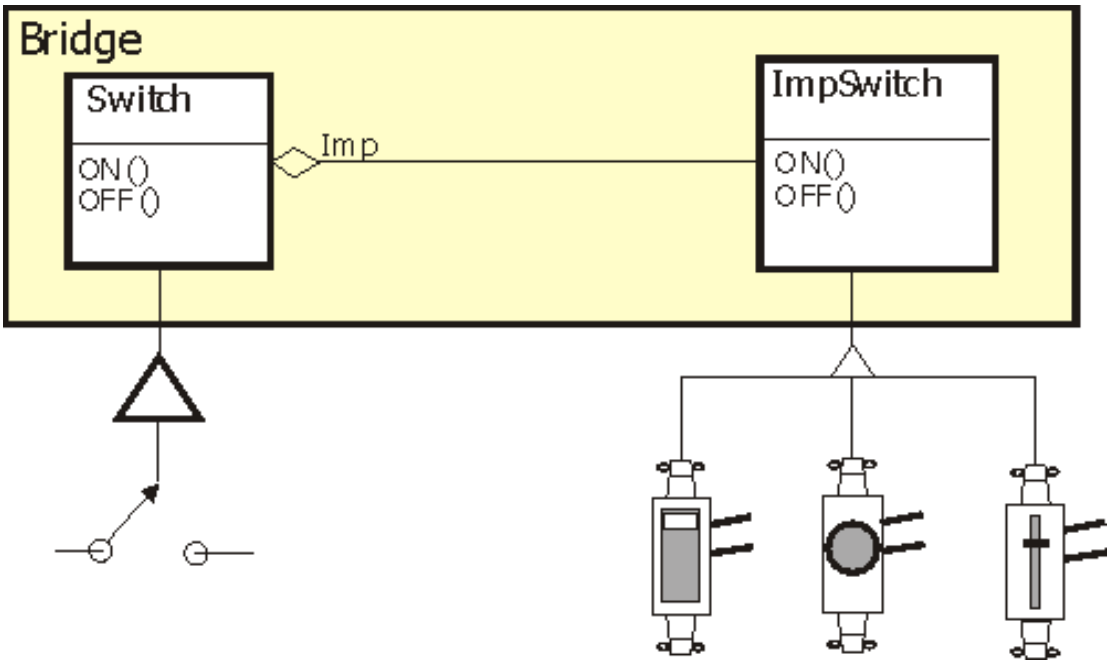


Figura 1. Diagrama exemplificando a utilização do padrão Bridge para Chaves Elétricas

Toda linguagem só é eficiente como meio de comunicação, quando todos os participantes conseguem se comunicar eficientemente. Para isto, há a necessidade de exemplos fora do domínio computacional.

1.5. Proto-Padrões e Verificação de Padrões

Brad Appleton [Appleton, 2000] lembra que " *(...) nem toda solução, algoritmo, boa prática, máxima ou heurística constitui um padrão. (...) É necessário que o fenômeno seja recorrente (preferivelmente em pelo menos três sistemas existentes - normalmente chamada de Regra de Três). Alguns crêem que não é apropriado chamar algo de padrão até que seja submetido a um certo grau de escrutínio ou revisão por outros.*"

Quando um padrão não é ainda reconhecido como tal, é chamado de proto-padrão. Descrições breves destes proto-padrões são chamadas de *patlets*.

Para verificar se uma solução pode ser documentada como padrão, primeiramente deve ser verificada sua recorrência. Sem ela, não há a menor possibilidade de se tratar de um padrão.

James Coplien [Coplien, 1996] define que bons padrões fazem o seguinte:

- **Resolvem um problema:** padrões capturam soluções, não somente princípios abstratos ou estratégias;
- **São um conceito aprovado:** padrões capturam soluções com um registro trilhável, não teorias ou especulações;
- **A solução não é óbvia:** muitas técnicas de solução de problemas (como métodos e paradigmas de projeto de *software*) tentam derivar soluções de princípios básicos. Os melhores padrões geram uma solução para um problema indiretamente, abstraindo as particularidades e generalizando a solução - uma abordagem necessária para os problemas de projeto mais difíceis;
- **Descrevem um relacionamento:** padrões não são simples descritores de módulos, mas descrevem mais profundamente estruturas e mecanismos;
- **Têm um componente humano significativo:** todo *software* deve servir ao conforto humano ou qualidade de vida; os melhores padrões explicitamente apelam à estética e utilidade.

Como Appleton [Appleton, 2000] diz: "*Um padrão é onde a teoria e a prática se encontram para reforçar e complementar uma à outra, mostrando que a estrutura que ele descreve é útil, utilizável e utilizada*".

1.6. Os Anti-Padrões

Se um padrão é considerado uma "boa prática", então um anti-padrão representa uma "lição aprendida".

"*Enquanto é razoável assumirmos que a principal razão para se escrever software é prover soluções para problemas específicos, também é argumentável que estas soluções freqüentemente nos levam a um estado pior do que quando começamos*", enuncia o site www.antipatterns.com. Uma foto de um livro *Construa Sua Própria Bomba Atômica em Sete Passos Fáceis* ilustra o conceito...

Inicialmente proposto por Andrew Koenig [Koenig, 1995] na edição de novembro de 1995 do *C++ Report*, há duas noções de anti-padrões:

1. Aqueles que, descrevendo uma solução ruim para um problema, resultam em uma situação indesejada.
2. Aqueles que descrevem como sair de uma situação indesejada e como proceder para alcançar uma situação desejada (boa situação).

É possível encontrar na internet vários anti-padrões, alguns até bem engraçados como, o "Bode Expiatório", o "Culto da Carga" e "Processo à Prova de Idiotas", que detalharemos no Apêndice B.

Na realidade, os anti-padrões são úteis se aderirmos à idéia de que "aprendemos mais com nossos erros do que com nossos acertos". Como elemento de todo anti-padrão, há a seção Lições Aprendidas, onde se expõe como todo o processo culminou em uma situação indesejada.

Os anti-padrões devem ser utilizados tanto para identificar as alternativas a serem escolhidas para o projeto que fatalmente resultarão em desastre (lembrando que é um padrão e, portanto, reproduzível), como para encontrar uma maneira de sair da situação indesejada para uma retomada de rumo.

1.7. Formato de um Padrão

Para descrever um padrão, alguma formalidade é necessária, mas somente o suficiente para que outras pessoas identifiquem rapidamente que se trata da descrição de um padrão.

Os formatos podem variar ligeiramente, dependendo da categoria de padrões a que se referem, mas acabam se enquadrando em dois formatos principais: o usado nos padrões de C. Alexander, chamados de Forma Alexandrina [Alexander, 1979], e os descritos da mesma maneira que no livro da *Gang of Four*, chamados de Forma GoF [GoF, 1995].

É preciso ter em mente que padrões são escritos em linguagem natural, não em código. Dependendo do tipo de padrão, pode-se encontrar alguma representação gráfica, mas também ela não é necessária em todas as descrições de padrão. Os formatos variam desde a Forma Portland, que é puramente narrativa, até a Forma GoF, muito mais estruturada, com descrições em UML[4], inclusive.

Vamos dar uma breve explanação das estruturas de três formatos diferentes: O Portland [Gabriel, 1996], que é muito próximo do Alexandrino, o próprio Alexandrino, e o GoF, mais orientado a objetos.

Exemplos da forma Portland se encontram no Apêndice C, da forma Alexandrina no Apêndice D e da forma GoF, no Apêndice E.

1.7.1. Forma Portland

Contém três seções: Documento de Linguagem, Parágrafos do Padrão e Sumário.

- **Documento de Linguagem**: contém um sistema de padrões que operam em conjunto;
- **Parágrafos do Padrão**: segue uma estrutura bem natural de descrição de padrão:
  - "Deparando-se com este problema..."
  - "O problema existe por causa disto e as forças a serem resolvidas são..."

[4] UML - Unified Modelling Language : Linguagem Gráfica de Modelagem Orientada a Objetos

Portanto...
- "Aja aproximadamente da seguinte maneira..."
- "Agora você está preparado para tratar dos seguintes problemas.."

- **Sumário** : dentro da linguagem, existem grupos de padrões com idéias similares; no sumário são explicitadas as entrelinhas da criação do padrão.

1.7.2. Forma Alexandrina

A maior parte dos padrões Alexandrinos são descritas da seguinte maneira:

SE você se encontra em um **CONTEXTO**
por exemplo, **EXEMPLO**
com o **PROBLEMA**
submetido a determinadas **FORÇAS**
ENTÃO por algumas **RAZÕES**
aplique **REGRAS E/OU FORMATOS DE PROJETO**
para construir a **SOLUÇÃO**
convergindo para um **NOVO CONTEXTO** e **OUTROS PADRÕES**

É razoavelmente informal, mas com alguma estrutura.

- **Nome**: pode se usar uma única palavra ou uma descrição curta, desde que o nome seja significativo;
- **Problema**: as metas e objetivos que se deseja alcançar dentro do sistema de forças e contexto;
- **Contexto**: as pré-condições para que haja a recorrência do problema;
- **Forças**: as restrições e limitações que conflitam com os objetivos que desejamos atingir;
- **Solução**: relações estáticas e regras dinâmicas que ensinam como alcançar a meta desejada;
- **Exemplos**: uma ou mais aplicações ilustrando como utilizar o padrão em termos práticos;
- **Contexto resultante**: o estado da configuração do sistema após a aplicação do padrão;
- **Motivação (Rationale)** : uma leitura das entrelinhas da utilização do padrão, explanando como o problema é resolvido e como as forças são orquestradas para se obter harmonia;
- **Padrões relacionados**: padrões que fazem parte da mesma linguagem deste padrão;
- **Usos conhecidos**: descreve ocorrências conhecidas do uso do padrão, auxiliando na validação do mesmo.

1.7.3. Forma GoF

É mais extenso, obedecendo a um gabarito com os seguintes itens:

- **Nome e classificação do padrão**: o nome expressa a essência do padrão de forma sucinta. Um bom nome é vital porque ele se tornará parte do vocabulário de projeto. A classificação do padrão será descrita na próxima seção;
- **Intenção e objetivo**: declaração curta sobre o que o padrão faz e quais problemas trata;

- **Também conhecido como**: os apelidos e sinônimos do padrão, se existirem;
- **Motivação**: um cenário que ilustra o problema e como o padrão o soluciona;
- **Aplicabilidade**: é o universo de contextos onde o padrão pode ser aplicado;
- **Estrutura**: representação gráfica do padrão, normalmente em UML, se o padrão for de classe e objetos;
- **Participantes**: as classes e/ou objetos que participam do padrão e suas responsabilidades;
- **Colaborações**: como os participantes colaboram para executar suas responsabilidades;
- **Conseqüências**: quais os custos e benefícios da utilização do padrão;
- **Implementação**: dicas, sugestões, técnicas e armadilhas conhecidas quando da implementação do padrão, incluindo considerações específicas de linguagem;
- **Exemplo de código**: fragmentos ou blocos de código para ilustrar a implementação;
- **Usos conhecidos**: exemplos do padrão encontrados em sistemas reais;
- **Padrões relacionados**: bem parecido com a idéia de linguagem de padrões do formato Alexandrino. Quais os padrões mais intimamente relacionados com este.

A utilização deste ou daquele formato deve ser considerada no momento da análise global do que o sistema faz. Para projetos sem a participação de usuários (pessoas no sentido físico), o padrão pode ser descrido com mais formalidade. Para projetos com forte ligação com a interação humana, um formato menos rígido normalmente dá melhor resultado.

Veremos mais à frente que, para a área de Interação Humano Computador, precisamos de algumas adaptações nestes formatos, chegando a algo próximo à forma Alexandrina.

1.8. Classificação de Padrões

Christopher Alexander escreve em *The Timeless Way of Building* [Alexander, 1979]: "*Cada padrão, então, depende tanto dos padrões menores, contidos nele, como dos padrões maiores, nos quais ele está contido*".

Padrões tratam basicamente de relacionamento, de interação, entre os padrões e entre os componentes de cada padrão. Existe claramente a necessidade de se estabelecer uma organização entre os padrões, de modo que possamos classificá-los e, assim, estabelecer uma maior eficiência na sua utilização.

Para facilitar a navegação entre os padrões, necessitamos de quatro conceitos básicos: Relações, Aglomerados, Níveis de Abstração e Pontos de Vista:

- **Relações:** relacionamentos entre padrões ajudam a identificar padrões que são próximos ao padrão que está sendo utilizado ou analisado;
- **Aglomerados:** (do inglês *Clusters*), agrupam padrões que pertencem a uma área temática;
- **Níveis de Abstração:** possibilitam descrever conceitos de maneira consistente com o nível de detalhamento da discussão;

- **Pontos de Vista:** auxiliam a selecionar o vocabulário que é relevante a uma equipe em particular (por exemplo, a equipe de redes).

A versão inicial do ESP - *Enterprise Solution Patterns Using Microsoft .NET* [Microsoft, 2003], uma coleção de padrões da Microsoft para a plataforma .NET, identifica os cinco aglomerados exibidos na Tabela 1

Tabela 1

Aglomerado	Problema
Apresentação Web	Como criar aplicações Web dinamicamente ?
Distribuição	Como dividir uma aplicação em camadas e depois distribuí-las em uma infraestrutura de *hardware* multi-camada ?
Sistemas Distribuídos	Como se comunicar com objetos que residem em processos ou computadores diferentes ?
Performance e Confiabilidade	Como criar uma infraestrutura de sistema que atenda requisitos operacionais críticos ?
Serviços	Como acessar serviços providos por outras aplicações? Como expor as funcionalidades da sua aplicação como serviços a outras aplicações ?

Quando se classifica padrões pelo seu nível de abstração, usualmente se divide em três níveis: Padrões Arquiteturais, Padrões de Projeto e Padrões de Implementação:

- **Padrões Arquiteturais:** expressam um esquema ou organização estrutural fundamental para sistemas. Fornecem um conjunto de subsistemas predefinidos, especificando suas responsabilidades e incluindo regras e diretrizes para organizar os relacionamentos entre eles;
- **Padrões de Projeto:** fornecem um esquema para refinar os subsistemas e componentes do sistema, ou o relacionamento entre eles. Descrevem estruturas recorrentes comuns de comunicação entre componentes que solucionam um problema geral de projeto dentro de um contexto particular;
- **Padrões de Implementação:** também chamados de **Idiomas**, são padrões de um nível de abstração mais baixo, específico para uma linguagem de programação ou plataforma. Um idioma descreve como implementar determinados componentes ou o relacionamento entre eles usando as construções de determinada linguagem.

Embora os níveis de abstração ajudem a endereçar diferentes grupos de usuários, eles não refletem o fato de que uma solução de *software* compreende muito mais do que componentes de código. Uma visão holística de um sistema inclui *software* personalizado, *software* específico a uma determinada plataforma, infra-estrutura de *hardware* e a distribuição do *software* pelo *hardware*.

Dois pontos de vista do mesmo sistema descrevem a mesma solução, mas não descrevem hierarquias, apenas maneiras diferentes de ver a mesma coisa.

Nos *Enterprise Solution Patterns* [Microsoft, 2003], são descritos quatro pontos de vista, explicitados na Tabela 2:

Tabela 2

Ponto de Vista	Descrição
Banco de Dados	Descreve a camada persistente da aplicação. Este ponto de vista foca em coisas como esquemas físicos e lógicos, tabelas, relações e transações.
Aplicação	Foca no aspecto executável da solução. Inclui coisas como modelos de domínio, diagramas de classe, codificações e processos.
Distribuição	Explicitamente mapeia a aplicação com relação à infra-estrutura (por exemplo, processos nos processadores).
Infra-Estrutura	Incorpora todo o *hardware* e equipamentos de rede que são necessários para executar a solução.

1.9. Uma Linguagem de Padrões

Ao utilizarmos padrões, devemos ter sempre em mente certos conceitos sobre eles:

- **Padrões devem ser orgânicos** - nas palavras de Alexander [Alexander, 1979]*: "um perfeito equilíbrio entre as necessidades das partes e as necessidades do todo*";
- **Padrões devem ser "vivos"** - a idéia aqui é de que padrões devem ter embutidos em si uma certa consciência da sua existência, do seu papel no sistema;
- **Padrões devem ser sinérgicos** - ou seja, o valor de vários padrões agrupados coerentemente deve ser maior que a soma dos valores de cada padrão em separado;
- **Padrões devem ser gerativos** - não simplesmente indicam o fenômeno, mas permitem gerá-lo novamente.

Traduzindo: para obter um resultado realmente efetivo da utilização de padrões, deve-se utilizar uma coleção deles, não apenas um.

Uma coleção de padrões forma um vocabulário para compreender e comunicar idéias. Quando esta coleção é descrita de tal maneira que se revela uma coesão, onde estruturas e relacionamentos inerentes aos seus componentes trabalham em conjunto, visando um objetivo comum, esta coleção é chamada de Linguagem.

James Coplien [Coplien, 1996] define uma linguagem de padrões como "*uma coleção estruturada de padrões que se complementam uns aos outros para transformar necessidades e restrições em uma arquitetura*".

A idéia por trás disto tudo é dada por Alexander, quando fala de um "ecossistema de padrões", dando a noção de algo vivente. Desta forma, em *The Timeless Way of Building* [Alexander, 1979]: "*uma linguagem de padrões dá a cada pessoa que a utiliza, o poder de criar uma infinita variedade de novas e únicas construções, do mesmo modo que uma linguagem natural dá o poder de criar uma infinita variedade de sentenças.*"

Em outra parte do livro, ele diz que "*a estrutura da linguagem é criada pela rede de conexões entre os padrões individuais e a linguagem vive, ou não, como uma totalidade, até o grau em que estes padrões formam um todo*". "*Cada padrão então, depende tanto dos padrões menores que ele contém, como dos padrões onde ele está contido*" .

Como saber se uma linguagem de padrões é boa? Para facilitar a compreensão dos conceitos acima, Alexander diz: "*A linguagem é boa, capaz de fazer algo por completo, quando ela é morfologicamente e funcionalmente completa*".

Para James Coplien [Coplien, 1996], "*boas linguagens de padrões guiam o projetista em direção a arquiteturas úteis e para longe de arquiteturas cuja analogia literária seria o jargão ou escrita pobre. Boas arquiteturas são duráveis, funcionais e esteticamente agradáveis, e uma boa combinação de padrões podem equilibrar as forças em um sistema para alcançar estas três metas. Uma boa linguagem de padrões dá aos projetistas liberdade de se expressar e de compor a solução para as necessidades particulares do contexto onde os padrões são aplicados*".

1.10. Uma Aplicação de Padrões : JUnit

Criado para automatizar os testes unitários de Java, auxiliando a divulgação do XP - *eXtreme Programming* - a mais famosa das metodologias de desenvolvimento, o JUnit [Beck, 2002] tem com criadores Kent Beck (do XP) e Erich Gamma (do livro *Padrões de Projeto*).

O caminho para chegar ao resultado atual é um exemplo da boa utilização de um conjunto de padrões de projeto. A idéia foi partir do nada e aplicando padrões, um após o outro, chegar à arquitetura do sistema.

A seguir, listaremos os problemas encontrados e os padrões que, aplicados, solucionariam o problema da automatização de testes unitários.

Como criar um caso de teste? - os casos de teste são compreendidos como objetos, e para isto criaram a classe *TestCase* [Beck, 1996], partindo do padrão *Command* [GoF, 1995];

Onde executar o teste? - os testes têm um padrão, que se resume em preparar o teste, executar e apresentar o resultado. Isto pode ser obtido utilizando o padrão *Template Method* [GoF, 1995];

Como relatar o resultado do teste? - se não soubermos se um procedimento passou ou não em um teste, como avaliar a qualidade? Para guardar os resultados dos teste e exibir um relatório com as falhas e os sucessos, utilizou-se o padrão *Collecting Parameter* [Beck, 1996];

Como evitar a explosão de subclasses? - a arquitetura deve ser estável e genérica o suficiente para que não se necessite criar uma classe nova a cada novo caso de teste implementado. Verificou-se que os casos de teste deveriam ter um comportamento acoplável, obtido através dos padrões *Pluggable Selector* [Beck, 1996] e *Adapter* [GoF, 1995];

Como testar um caso de teste ou uma suíte de casos na mesma arquitetura? - o padrão *Composite* [GoF, 1995] permite considerar o teste como somente um ou como uma suíte que, por sua vez, é composta de um teste ou uma suíte, e assim por diante, recursivamente;

A arquitetura resultante, de grande valor para os testes em Java encontra-se na Figura 2.

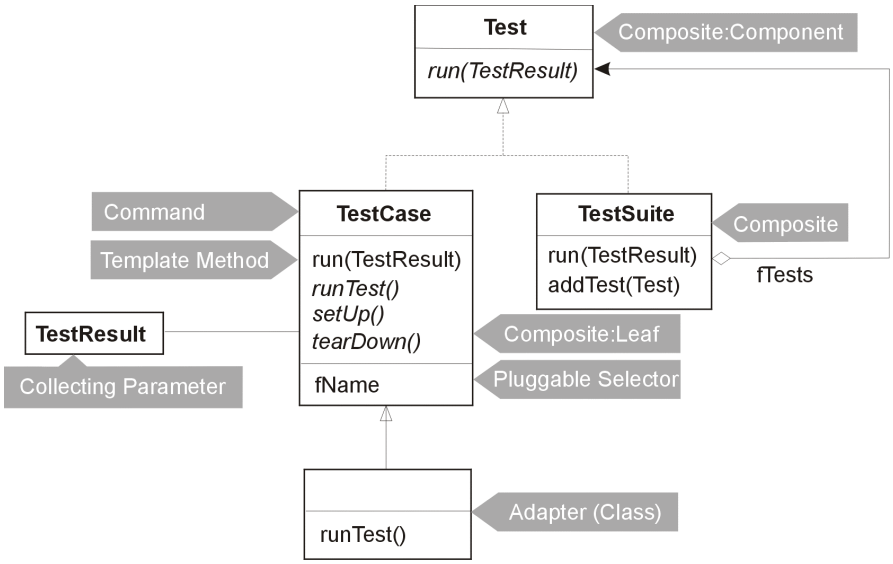


Figura 2. Arquitetura do JUnit

A linha de tempo da arquitetura do sistema JUnit pode ser visto na Figura 3:

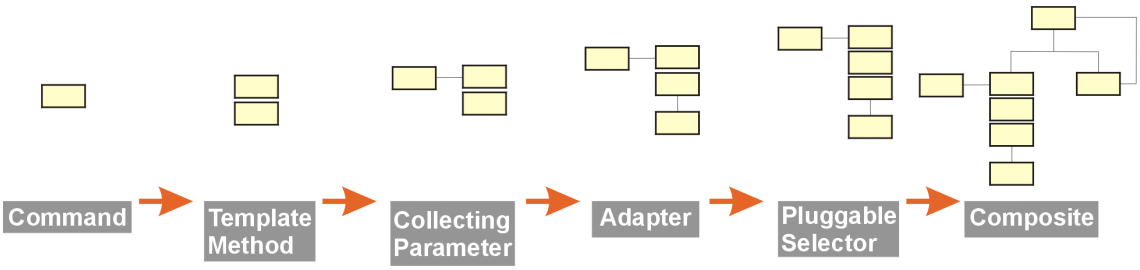


Figura 3. Linha do Tempo do JUnit

Com isto, se consegue uma alta densidade de padrões, que facilita o uso, mas dificulta a mudança. Na visão dos autores do JUnit, quanto mais densa é a estrutura, mais madura é a arquitetura. Visão compartilhada com Alexander, quando diz que [Alexander, 1979] "*é possível projetar edificações juntando padrões de uma maneira bastante casual. Uma edificação projetada desta forma é uma montagem de padrões. Ela não é densa, ela não é profunda. Mas também é possível juntar padrões de tal maneira que muitos se sobreponham no mesmo espaço físico: a edificação fica muito densa; tem muitos significados capturados em um pequeno espaço; e, através desta densidade, se torna profunda*".

Capítulo 2
Interação Humano-Computador

A interação humano-computador - IHC - (em Inglês: HCI - *Human-Computer Interaction*), é uma área multidisciplinar, envolvendo disciplinas como Ciência da Computação, Psicologia, Ergonomia e Sociologia para tentar abranger o conceito de um ser humano interagindo com um computador.

Numa definição mais formal, "é *a disciplina preocupada com o projeto, avaliação e implementação de sistemas computacionais interativos para o uso humano e com o estudo dos principais fenômenos ao redor deles*" [SIGCHI, 1992]

Uma vez que as máquinas mais simples de hoje já podem operar em ambientes gráficos e que o poder de processamento continua a aumentar, há uma crescente demanda por melhores paradigmas de projetos de interface de usuário.

O expressivo aumento no uso de computadores por pessoas cada vez menos capacitadas para tal exige um desenvolvimento de interfaces cada vez mais fáceis de serem compreendidas e cada vez mais robustas e confiáveis.

Há ainda outro grupo de pessoas que utilizam o computador cada vez mais cotidianamente, numa abrangência cada vez maior de tarefas, tanto profissionais como pessoais.

Atualmente, o uso do computador tem mudado seu foco. O computador é considerado nos dias de hoje como um facilitador para a comunicação entre pessoas. A diversidade de informação com que é possível manipular e comunicar, traz novas e impensadas audiências para os programas.

Hoje, ninguém mais adquire uma máquina que não possa estar conectada à internet. O uso cada vez mais intensivo de mecanismos de comunicação leva a situações onde, às vezes, a impressora é um acessório supérfluo, mas os dispositivos de comunicação (modem, placa de rede) são cada vez mais tratados como *commodities* do que como uma janela para o futuro.

2.1. O que é Interface ?

Interface é aquilo que serve de conexão entre dois modelos, entre duas visões: o modelo mental do usuário em relação ao sistema e o modelo de programa, construído pelos engenheiros de *software* [Norman, 2002]. Toda interface tem dois lados, uma para cada modelo conectado a ela.

Interfaces entre humanos e máquinas vêm sendo desenvolvidas desde que a primeira máquina foi criada, mas somente a partir da década de 80, as interfaces de usuário começaram a ser desenvolvidas visando o usuário final, e não profissionais altamente especializados.

A evolução do conceito de interface desde a década de 50 pode ser resumida assim:

1950 - Painéis com chaves seletoras - a interface se encontrava em nível de *hardware* e só era compreendida por engenheiros.

1960-1970 - Interface para Programação - com o advento de linguagens de programação como COBOL e FORTRAN, a interface migra para o *software*, mas é textual e é necessário entender o código de programação para operar a mesma.

1970-1980 - Terminais - a interface aparece ao usuário como comandos de interação. Ainda é textual.

1980 - GUI (*Graphic User Interface* - Interface Gráfica de Usuário), Multimídia - a interface atinge o nível de diálogo e inicia-se o abandono ao modo puramente textual.

1990 - *Groupware*, Sistemas Distribuídos - a interface abrange mais de uma máquina, permitindo que equipes de usuários trabalhem cooperativamente.

2000 - A interface começa a se tornar pervasiva, com dispositivos móveis, telas interativas, tecnologia embarcada, *middleware*.

Neste trabalho, iremos focar nas interfaces de usuário, deixando de lado as interfaces entre dois computadores ou dois usuários.

2.2. Quem é o usuário ?

O usuário, a princípio, é toda pessoa que, de alguma forma, é afetada pelo funcionamento de um determinado produto. O foco deste trabalho será nos usuários que terão contato mais próximo com o produto, interagindo com ele por meio da interface.

Para isto, é necessário responder a algumas perguntas básicas [Preece, 2002]:

- Quem será o usuário, ou seja, quem utilizará o produto ?
- Quais são os objetivos e necessidades destes usuários, quais as tarefas que ele deseja executar?
- Qual será o ambiente (contexto) onde a interação terá lugar?

Quando se implementa uma interface de usuário, é preciso entender que o usuário é uma mescla de várias pessoas, cada uma com necessidades, restrições e preferências individuais. O usuário, então, se torna uma abstração, cujas interações deverão ser projetadas visando alguns objetivos fundamentais da experiência do usuário.

Para isto, devemos ter em mente que:

- usuário nem sempre informa o que precisa, mas o que acha que precisa;
- em certas atividades os usuários são bons e em outras, nem tanto;
- usuário deve ser envolvido no processo de projeto da interface;
- os métodos desenvolvidos devem focar o uso do sistema por parte do usuário em um particular contexto.

2.2.1. Fatores Humanos

Quando falamos de usuário, a engenharia cognitiva traz alguns aspectos que auxiliam no processo de compreender as necessidades do usuário em foco, entre eles, a atenção, a percepção e reconhecimento e a memória.

2.2.1.1. Atenção

É a seleção de tarefas para se concentrar dentre várias ao nosso redor, em um dado momento. A atenção pode ser classificada [Preece, 2002] em atenção focalizada e atenção dividida.

Atenção Focalizada - é a habilidade de concentrar em um evento específico dentre vários

Atenção Dividida - quando é necessário atender a mais de uma demanda ao mesmo tempo

Num projeto de interface, deve haver um balanceamento entre o que necessita de atenção focalizada e o que necessita de atenção dividida. Na maior parte das tarefas, é desejável a atenção focalizada, mas com a consciência de que podem existir outros eventos que necessitem da atenção e intervenção do usuário.

Para dirigir a atenção do usuário para a tarefa mais relevante, devemos planejar a arquitetura da interface, tornando-a ao mesmo tempo simples e significativa.

2.2.1.2. Percepção e Reconhecimento

O usuário que vai interagir com a interface sempre traz uma bagagem, de experiências anteriores com aquisição de informação. Portanto, é importante que a interface tenha a informação facilmente perceptível, sem que seja necessário o uso de manuais e treinamentos extensivos.

Esta percepção da informação é realizada através dos sinais transmitidos pela interface ao usuário, como símbolos gráficos, textos, cores, etc. A informação deve ser entendida sem ambigüidades, pois a compreensão equivocada pode gerar tanto resultados errados como frustração por parte do usuário.

A engenharia cognitiva apresenta duas abordagens para o projeto de interfaces, a abordagem construtivista e a abordagem ecológica.

Abordagem Construtivista - a percepção é entendida como resultado da experiência anterior do usuário e da expectativa do mesmo sobre o que deveria aparecer. A idéia central é de que o usuário filtra as informações do mundo através de transformações, acréscimos, distorções. Há inclusive um processo de descarte de informação.

Abordagem Ecológica - a percepção é detectada e não construída. O conceito central passa a ser o de *affordance*, que será explicado mais à frente no texto. O comportamento de um produto percebido pelo usuário é somente o que o produto permite que seja visualizado.

2.2.1.3. Memória

A memória é nossa capacidade de armazenar e recuperar informação e a eficiência em utilizá-la para determinada tarefa.

A memória humana, diferente da de um computador, não é um simples registro de informação, mas é um repositório onde a informação recebida foi filtrada, processada, codificada e relacionada com outras informações dentro de um contexto. A memória é um mecanismo de informação sensorial, ou seja, a via de entrada de informação é através dos nossos sentidos.

A memória pode ser dividida em dois tipos: de curta e de longa duração.

Memória de Curta Duração - só se retém uma interpretação imediata dos eventos, sem muito processamento.

Memória de Longa Duração - é a informação que, após um processo que está longe de ser explicado, é armazenada e pode ser acessada às vezes por toda a vida da pessoa.

Um dos mais proeminentes trabalhos sobre psicologia cognitiva, que trata os tipos de memória, é o de Donald Broadbent[5].

Por ser muito mais complexa que uma memória de computador, nossa capacidade de identificar coisas é muito maior do que a de lembrar coisas, o que explica porque é mais fácil reconhecermos feições de pessoas do que lembrarmos os seus nomes.

Para o usuário, reconhecer é mais efetivo que recordar. Há inclusive teorias que dizem que a memória de curta duração só tem capacidade para guardar sete blocos de informação, ou *chunks*, com variação de dois *chunks* para mais ou para menos[6]. A noção é de que devemos reduzir o número de componentes que devam ser manipulados a cada interação.

Conforme aumenta o número de componentes e conforme o reconhecimento vai ganhando importância na tarefa, a utilização de símbolos gráficos (imagens, ícones, cores) vai ganhando mais destaque e a necessidade de organizar os componentes em grupos que façam sentido para o usuário se torna mais relevante.

Com estas características, percebe-se porque as interfaces devem ser significativas, preferivelmente gráficas e adaptadas às características culturais e contextuais do usuário.

Uma exceção a esta regra está nas interfaces em que seja requerido do usuário especificar e recuperar informação textual. Há uma sobrecarga cognitiva quando misturamos informações textuais, verbais e visuais. Consulta a sistemas bibliográficos é um bom exemplo disto.

2.2.1.4. Modelos Mentais

Cognição envolve muitos processos incluindo atenção, memória, percepção e aprendizado. Toda a interação realizada com o mundo exterior (ambiente, artefatos tecnológicos, outras pessoas, etc) produz e aperfeiçoa modelos mentais.

---

[5] Broadbent, D (1958). Perception and Comunication. Elsevier Science Ltd.

[6] Miller, G. A. (1956). The magical number seven plus or minus two: Some limitations on our capacity for processing information. Psychological Review, 63, 81-97

Joel Spolsky [Spolsky, 2002] descreve modelos mentais assim: *"(...) quando um usuário se senta para utilizar um programa, ele não vem como uma página em branco. Ele tem algumas expectativas sobre como o programa irá funcionar. Se ele já utilizou um programa similar anteriormente, ele achará que este programa funcionará de maneira parecida. Se ele já usou qualquer tipo de programa antes, a expectativa é de que este programa siga algumas convenções. Ele tem algumas opiniões inteligentes de como uma interface para este tipo de tarefa deve ser. Isto é chamado o Modelo do Usuário: é sua compreensão mental de como o programa deve ser."*

Continuando, *"O programa também tem seu modelo mental, com a diferença de que este está codificado em bits e roda fielmente em uma CPU. Este é chamado Modelo do Programa, e esta é 'A Lei'. (...) Se um Modelo de Programa corresponde ao Modelo de Usuário, temos uma boa interface."*

Norman [Norman, 2002] define a imagem declarada pelo produto (com janelas, manuais e acessórios), como sendo a "imagem do sistema". Para ele, os projetistas só se comunicam com o usuário através da imagem do produto (Figura 4).

Com isso, é possível entender porque, se seu modelo do programa não é trivial, ele provavelmente não corresponderá ao modelo do usuário.

Os modelos mentais, tanto do programa quanto do usuário, são incompletos e imprecisos. Ao projetar interfaces, devemos entender que a ambientação a uma interface é feita através de analogias, onde modelos do usuário passam por um processo de associação e adaptação aos modelos do programa. Se a interface não for projetada com foco nisto, corre-se o risco de gerar frustração no usuário e, pior, a eficiência do programa pode estar comprometida.

Figura 4. Modelos Mentais

O projeto da interface pode afetar enormemente quão bem os usuários podem entender, aprender e lembrar como realizar suas tarefas.

Atualmente, não se conhece suficientemente bem a construção e uso dos modelos mentais. A elicitação de tais modelos é uma área de pesquisa muito frutífera no projeto de interfaces de usuário. Se houver uma técnica que possibilite formalizar modelos mentais, a qualidade dos programas advindos da aplicação desta técnica deverá ser sensivelmente aumentada.

2.2.1.5. Frustração dos Usuários

Quando os modelos mentais do programa e do usuário não combinam, verificamos uma frustração por parte do usuário. Tal fenômeno é delatado pelas queixas dos usuários com relação ao programa:

- "Este programa trava demais !"
- "Não era isto que eu queria que ele fizesse..."
- "Eu achava que este programa faria isto por mim, mas não..."
- "O que eu tenho que fazer ?", ou sua variante, "Como este programa funciona mesmo?"
- "Este programa é chato (feio, desagradável, etc..)"
- "Ok, eu errei... Mas o que você quer me dizer com esta mensagem de erro?"
- "Se eu clicar no OK é sinal de que eu quero ou de que eu não quero?"

Os objetivos da experiência do usuário [Preece, 2002] são simples:

- a experiência deve trazer **satisfação**;
- a experiência deve ser **agradável**;
- a qualquer momento, o usuário deve ter **ajuda** para a tarefa que deseja executar;
- o programa deve **motivar** o seu uso;
- o programa deve ser **esteticamente agradável**;
- a experiência de utilização do programa deve ser **recompensadora**.

Podemos listar outros objetivos de experiência do usuário, mas estes podem ser considerados os principais. Todo projetista de interfaces deve ter em mente que, toda vez que algum destes objetivos não é preenchido, corre-se o risco de causar frustração nos usuários.

2.3. Projeto de Interação

O projeto de interação é um processo onde o objetivo é desenvolver um espaço de trabalho, através de uma interface, onde o modelo mental do usuário alvo seja correspondido pelo modelo do programa. A interface gerada a partir deste projeto é basicamente uma resposta às necessidades do usuário dentro do contexto abordado.

Por ser um processo, o projeto de interação necessita de um plano de desenvolvimento e, pela premissa de que os modelos mentais de usuário nunca são totalmente compreendidos, passível de exploração de alternativas e refinamentos.

Um bom processo de projeto de interação deve ter três características fundamentais:

- deve ser focalizado no usuário desde o inicio do projeto até a avaliação do artefato;
- deve identificar, documentar e definir metas de usabilidade relativas a experiências prévias de usuários específicos;
- deve ser iterativo, pois "projetistas nunca fazem a ' coisa certa' da primeira vez".

2.3.1. Princípios de Projeto

Em um bom projeto de interação, os princípios a seguir, elaborador por Donald Norman [Norman, 2002], quando seguidos, elevam a qualidade do produto final:

Visibilidade - torne o relevante visível e o que precisa ser feito, óbvio. Em outras palavras, não deixe para o usuário a descoberta das funcionalidades.

Realimentação (*Feedback*) - informe ao usuário sobre o reconhecimento de uma ação e novo estado resultante da ação que ele acaba de executar. Isto pode ser feito através de cores, texto, sons, ou uma combinação destes.

Restrições - nem tudo é permitido ao usuário e nem tudo que é permitido ele deve poder executar. As restrições, como paredes em uma sala, servem para que o usuário perceba até onde pode ir. Servem também para evitar que erros sejam cometidos.

As restrições podem ser divididas em três tipos: físicas, culturais e lógicas.

Restrições físicas são componentes de uma interface que não permitem que, fisicamente, uma tarefa seja executada de outra maneira diferente da qual foi projetada.

Exs.: Chave na fechadura, cartão de banco no caixa automático, etc.

Restrições culturais definem o que é universal e o que é culturalmente específico.

Ex.: A nossa Cruz Vermelha, nos países islâmicos é o Crescente Vermelho.

Restrições lógicas servem para explorar o modelo mental do usuário, através do senso comum, sobre como as coisas funcionam.

Exs.: Relacionar, em um corredor com várias portas, as portas e as chaves através de números.

Mapeamento - é a relação entre a distribuição dos controles pela interface e seus respectivos resultados. O resultado do acionamento de um controle deve, na medida do possível, não ser surpresa para o usuário.

Consistência - traz o conceito de quanto mais parecido forem dois elementos, mais semelhantes serão as tarefas que eles executam. A quebra da consistência aumenta o tempo de aprendizagem e normalmente induz a erros.

A consistência pode ser tanto interna como externa. A consistência interna refere-se ao projeto de operações similares dentro de uma aplicação. Quanto mais complexa a interface, mais difícil é obter consistência. Já a consistência externa envolve uma correspondência entre elementos da interface e do mundo externo.

Affordance - é um termo sem tradução boa no Português, mas que refere-se a uma característica de um elemento que torna óbvia a sua utilização. É o poder de sugestão de uso do elemento.

Por exemplo, um botão sugere que devemos pressioná-lo. Um resultado diferente do pressionar neste botão pode gerar frustração mas, se o resultado for mesmo este, não há necessidade de treinamento para operar o controle.

Em interfaces, Norman [Norman, 2002] diz que não tem sentido tratarmos *affordances* como as do mundo real. No caso, podemos entendê-las como *affordances* "percebidas". São convenções ação-efeito utilizadas no projeto de interface que devem ser aprendidas. Um exemplo disto é a barra de rolagem das janelas.

Muito do *affordance* de interfaces é obtido através de metáforas.

2.3.2. Projeto Conceitual e Projeto Físico

O projeto de interação pode ser sub-dividido em dois: projeto conceitual e projeto físico.

O projeto conceitual mostra como traduzir os requisitos do sistema para a arquitetura do mesmo. Metáforas, protótipos, diretrizes e outras técnicas são empregadas para permitir a tradução mais eficiente dos requisitos em funcionalidades.

O projeto físico existe em um nível menor de abstração. Nesta etapa são definidas as telas, menus, ícones e demais elementos da interface.

2.3.3. Metáforas

Metáforas são parte integrante do nosso pensamento e linguagem, o ser humano é metafórico por natureza. Trata-se de aplicar o significado original de uma coisa em outra para facilitar a compreensão.

Exemplos: gastar dinheiro, derrubar um argumento, trânsito engarrafado, partir o coração...

Para Erickson [Erickson, 1990]: *"Uma metáfora é uma rede invisível de termos e associações que auxiliam o modo de falar e pensar sobre um conceito. (...) Metáforas funcionam como modelos naturais, possibilitando-nos pegar nosso conhecimento de experiências e objetos familiares e concretos e usá-los para dar estrutura a um conceito mais abstrato. (...) Uma metáfora pode fazer diferença mesmo quando não há gráficos ou textos associados à interface".*

Nas interfaces, as metáforas funcionam como um modelo, uma realização de uma abstração, como, por exemplo, o gerenciador de arquivos ou a área de trabalho do *MS-Windows*. As metáforas na interface agem como catalisadores entre o modelo do usuário e o modelo do programa.

Hoje em dia, usuários mais jovens não têm mais contato com a antiga máquina de escrever, mas o processador de textos trabalha exatamente com esta metáfora. O ambiente em que o usuário opera é o de um teclado com uma folha de papel diante de si.

Como parte integrante da interface, a escolha da metáfora correta para uma situação é de tal importância que uma escolha errada pode fazer com que o usuário, devido a uma distância grande entre seu modelo mental e o que a metáfora sugere, abandone o programa.

Erickson [Erickson, 1990] indica como identificar metáforas adequadas para uma interface, em um processo baseado em três etapas:

Inicialmente, descobrir quais as metáforas que já estão implícitas na própria definição do problema. Se você não conhecer bem, em detalhes, o foco do sistema, você não estará habilitado para encontrar uma boa analogia com o mundo real que auxilie o uso da interface.

Após isto, identificar as áreas onde a compreensão da funcionalidade é mais difícil para o usuário. Isto pode se tornar a parte mais difícil do processo, já que às vezes ainda não temos heurísticas para identificar tais áreas com precisão. Sempre devemos lembrar que metáforas não são "balas de prata", que só por si só não darão conta de toda a funcionalidade...

Na terceira etapa, gera-se o máximo de metáforas possíveis que suportem o modelo requerido. A seguir, submete-se tais metáforas por um processo de avaliação:

- A metáfora está suficientemente compreendida pelo público-alvo? O público-alvo poderá mudar no decorrer do ciclo de vida do programa? A metáfora é realmente fácil e não somente comum?
- A metáfora é rica? Ela provê abundância de estruturas para trabalhar?
- A metáfora sugere funcionalidade que não está incluída no produto? Você incluiu "pistas" na interface que sugerem o contrário?
- A metáfora falha em indicar a funcionalidade presente no produto? Quais "pistas" foram incluídas para contrabalançar isto?
- O uso de várias metáforas está confundindo a identificação das funcionalidades do produto? Você pode reduzir o número de metáforas utilizadas? Alguma metáfora colide com outra?
- A metáfora é representada claramente na interface? Os usuários são capazes de identificar tal metáfora?
- A metáfora possibilita o uso por diferentes níveis de usuários? Pelo menos, ela pode ser utilizada pelo público-alvo?
- Houve negociação suficiente do significado da metáfora para assegurar a compreensão por vários níveis de usuário para, assim, evitar os problemas de interpretação?

Veremos mais à frente que as diretrizes de Madsen [Madsen, 1994] também permitem identificar metáforas adequadas para um projeto de interface.

2.3.4. Diretrizes (*Guidelines*)

Diretrizes são consensos de informações e recomendações desenvolvidos para auxiliar na tomada de decisões em um projeto. Diretrizes não são protocolos fixos que devem ser obrigatoriamente seguidos mas pretendem apresentar intervenções que são habitualmente recomendadas baseadas mas melhores evidências disponíveis. Podem ser gerais ou específicas a determinado contexto.

O uso de diretrizes no projeto de interfaces auxilia o projetista a falar a língua do usuário, reduzir a carga cognitiva, desenvolver sistemas robustos e manter consistência.

A tabela 3 traz alguns exemplos da relação tarefa-diretriz:

Tabela 3

Tarefa	Definição	Diretriz
comparar	examinar dois ou mais objeto de modo a descobrir semelhanças e diferenças entre eles	para valores quantitativos, texto é superior a gráfico

discriminar	descobrir as diferenças entre dois ou mais objetos	para interfaces gráficas, gráfico é superior
reconhecer	identificar um objeto ou classe de objetos	para objetos abstratos, texto é superior a gráficos
pesquisar	pesquisar um objeto complexo fazendo uma varredura entre todos os seus elementos	para valores quantitativos, apresente os itens em uma lista ou tabela
selecionar	escolher um ou mais objetos em um conjunto deles	para todos os tipos de objeto, use menus textuais com atalhos
integrar	sintetizar vários objetos em uma única funcionalidade	para objetos gráficos, exiba os objetos e permita manipulação direta
julgar	formar uma opinião a partir de uma análise cuidadosa das evidências	para todos os tipos de objetos, esta é uma tarefa humana
decidir	chegar a uma solução cujo resultado não é completamente conhecido	para todos os tipos de objeto, apresente simultâneamente pistas adequadas para evitar confusão. Obviamente, é uma tarefa para humanos

2.3.4.1. Regras Áureas de Shneiderman

Ben Shneiderman propôs uma coleção de princípios em seu livro *Designing the User Interface* [Shneiderman, 1998] e chamou-os de regras áureas, ou *golden rules*. São derivados heuristicamente da sua experiência e aplicáveis na maioria dos sistemas interativos:

A tabela 4 traz as oito regras áureas:

Tabela 4

Diretriz	Explicação
Lute pela consistência	Seqüências consistentes de ações devem ser requeridas em situações similares; terminologia idêntica deve ser usada em *prompts*, menus e telas de ajuda; e comandos consistentes devem ser utilizados em todo lugar.
Permita que usuários freqüentes utilizem atalhos	Conforme a freqüência de uso aumenta, os usuários desejam reduzir o número de interações e aumentar a velocidade da interação. Abreviações, teclas de função, comandos ocultos, e facilidades de macro são muito úteis ao usuário experiente.
Ofereça respostas informativas	Para toda ação do usuário, deve haver alguma resposta do sistema. Para ações freqüentes e simples, a resposta deve ser modesta, enquanto que para ações infreqüentes e importantes, a resposta deve ser mais substancial.
Projete diálogos fechados	Seqüências de ações devem ser organizadas em grupos com começo, meio e fim. A resposta informativa no término de um grupo de ações dá ao operador a satisfação de acompanhamento, um senso de conforto, o sinal para deixar de lado os planos de contingência e opções em suas mentes, e uma indicação de que o caminho está livre para preparar o próximo grupo de ações.

Ofereça um tratamento de erros simples	Na medida do possível, projete o sistema de modo que o usuário não cometa erros graves. Se um erro ocorre, o sistema deve ser capaz de detectar o erro e oferecer mecanismos simples e compreensíveis para tratar o erro.
Permita uma fácil reversão de ações	Esta característica diminui a ansiedade, já que o usuário sabe que ações impróprias podem ser desfeitas; encoraja a exploração de opções não familiares. As unidades de reversibilidade podem ser uma simples ação, uma entrada de dados, ou um grupo completo de ações.
Suporte a obtenção de controle	Usuários experientes desejam fortemente sentir que eles estão no comando do sistema e que o sistema responde às suas ações. Projete o sistema para fazer dos usuários os iniciadores das ações em vez de simplesmente responderem a elas.
Reduza a utilização de memória curta duração	A limitação humana de processar em memória de curta duração requer que o que for exibido seja simples, páginas múltiplas sejam consolidadas, movimento na tela seja reduzido, e tempo suficiente de treinamento seja alocado para comandos e seqüências de ações.

2.3.4.2. Diretrizes de Madsen

As diretrizes de Madsen [Madsen, 1994], elaboradas em 1994, têm por objetivo gerar, avaliar e desenvolver metáforas em três fases: a geração, a avaliação e o desenvolvimento de metáforas.

Fase 1 - Geração

A. Construa sobre metáforas existentes;

B. Use artefatos e ferramentas familiares como metáforas;

C. Focalize em metáforas ou estruturas já existentes no problema ou domínio da tarefa.

A principal preocupação na identificação de metáforas é prover os benefícios da familiaridade. A metáfora pode conter inconsistências, já que se trata de uma abstração. Se a metáfora funciona, o usuário pode aprender com esta incongruências, como por exemplo, as janelas flutuantes na metáfora do Ambiente de Trabalho (*Desktop*).

A interface deve auxiliar o usuário a transpor suas intenções em ações na própria interface. Neste caso, o teste a ser feito é confrontá-la com a ação para a qual a metáfora está sendo identificada.

Fase 2 - Avaliação

A. Verifique o mapeamento entre a metáfora e a ação;

B. Tenha sempre o usuário em mente;

C. Assegure-se de que a metáfora esteja compreendida.

As metáforas devem ser avaliadas em ação, já que metáforas podem não funcionar. Um meio de avaliar metáforas é aplicar testes de usabilidade. O desempenho de uma metáfora é melhor na medida em que facilita a ação mapeada.

Uma metáfora boa para um grupo de usuários pode não funcionar para outro grupo. É importante e necessário ter o público-alvo sempre focalizado.

Fase 3 - Desenvolvimento

A. Identifique os conceitos ativados;

B. Adapte apropriadamente a metáfora;

C. Verifique as hipóteses subliminares.

Metáforas úteis contêm conceitos chave indutores de ações que são apropriadas dentro do contexto da metáfora. Se possível, um conceito deve ser ativado somente por uma metáfora e uma metáfora deve ativar somente um conceito.

A metáfora evidencia o modelo mental que se utiliza. Portanto, como parte integrante de uma metáfora, vêm hipóteses sobre como funciona e o que é tal metáfora. A etapa C pode ser entendida como a fase de testes da metáfora implementada.

2.3.5. Prototipagem

Quando falamos de projeto de interação, o conceito de protótipo difere um pouco do tradicional. Normalmente, falamos em protótipo quando temos um modelo em menor escala do objeto a ser criado (exs. carro, edifício, ponte...).

No projeto de interação, um protótipo funciona como uma avaliação da metáfora escolhida. Ele permite que o modelo de programa seja confrontado com o modelo mental do usuário para um maior refinamento dos modelos. Sua construção deve ser rápida e adaptada à sua função no projeto, ao seu objetivo de avaliação e à sua técnica de construção.

Um protótipo é, em suma, uma técnica interativa onde o usuário é constantemente envolvido para testar as diferentes possibilidades de projeto da interface. Como para a maioria dos usuários, a interface é o próprio sistema, pode-se optar por trabalhar com um protótipo que será descartado ou um que evoluirá até ser o próprio sistema.

O protótipo pode ser classificado de diferentes formas como, por exemplo, pela sua função no projeto, pelo seu objetivo de avaliação ou ainda pela sua técnica de construção:

Função no Projeto

- **Contrastante** - confronta os modelos mentais do projetista e do usuário, determinando viabilidade e atendimento aos requisitos do usuário;
- **Autêntico** - foca em um aspecto específico da interface ou da funcionalidade, ajudando na compreensão dos problemas envolvidos. Na maior parte das vezes é provisório e funcional;
- **Funcional** - é um tipo de protótipo mais utilizado pelo projetista sem o contato com o usuário, para sanar dúvidas relativas à construção do sistema;
- **Piloto** - trabalha com as funcionalidades mais básicas, mas no ambiente onde estará o sistema. É evolutivo e tende a tornar-se o sistema posteriormente.

Objetivo de Avaliação

- **Exploratório** - esclarece requisitos do usuário ou determina a solução mais adequada dentro de um grande número de opções;
- **Experimental** - trabalha aspectos mais técnicos do desenvolvimento, fornecendo resultados experimentais para a tomada de decisões de projeto;
- **Evolutivo** - trata-se de um processo de reengenharia contínuo, onde as soluções são avaliadas em conjunto com os usuários e, se necessário, reprojetadas.

Técnicas de Construção

- **Completo** - toda a funcionalidade está contida, mas sem o desempenho do sistema definitivo;
- **Horizontal** - prototipa apenas uma camada do sistema (ex. esquema de telas);
- **Vertical** - é um protótipo completo de uma visão do sistema (ex.: cadastro do cliente).

Um protótipo pode ser um esquema de telas, um *storyboard*, uma apresentação, uma simulação em vídeo ou ainda um *software* com funcionalidade limitada. Depende do propósito do protótipo e do tipo de usuário que avaliará o mesmo.

A elaboração de um protótipo obedece basicamente às mesmas regras utilizadas para a identificação da metáfora e, aproximadamente, às mesmas fases. Adicionalmente às fases de geração, avaliação e desenvolvimento, há a fase de verificação de maturação do protótipo onde são, ou não, identificados novos requisitos, ou estes são identificados em pequeno número e não têm importância crucial no projeto.

2.4. Modelo Conceitual

Modelos conceituais são representações mentais de como os elementos de interação funcionam e como os controles da interface afetam o sistema. É no projeto da interface que ocorre uma adaptação evolutiva entre o modelo mental do usuário, com todas suas pré-concepções, e o modelo do programa, com as suas funcionalidades.

A interface implementada deverá comunicar o modelo, isto é, todas as metáforas escolhidas, dentro do paradigma de interação escolhido, deverá deixar claro para o usuário o que o programa poderá e deverá fazer. Neste processo, deve-se considerar alternativas e interagir muito com o usuário, através de protótipos e outras técnicas.

Na elaboração do modelo conceitual deve-se considerar três perspectivas:

Escolha do modo de interação - se será baseado em atividades (comandos, manipulação, exploração, etc) ou se será estrututurado ao redor de objetos do mundo real;

Escolha do paradigma de interação - WIMP, computação ubíqua, vestível, , etc...

Escolha da metáfora - seguindo as diretrizes já apresentadas.

- Exemplos de modelos baseados em atividades: Dando instruções; Comandos e menus; Conversacional; Manipulando e Navegando.

- Exemplos de modelos baseados em objetos: *e-Books*; *Desktop* - metáfora da mesa de escritório; Planilhas de cálculo.

2.4.1. Estilo de Interação

Estilo de interação é a forma como os usuários interagem com sistema computacionais [Preece, 2002]. Podemos classificar o estilo de interação de várias formas, onde as principais são linguagem natural, linguagem de comando, menus, WIMP, preenchimento de formulário e manipulação direta.

- **Linguagem Natural** - trata-se de interagir com o sistema utilizando-se da sua própria linguagem, ou seja, utilizando a mesma comunicação que dois serem humanos utilizariam. É utilizada principalmente em sistemas que interajam com usuários com pouco ou nenhum treinamento. Sistemas de consulta ou base de conhecimentos são exemplos onde é possível utilizar este estilo de interação. A interface pode ser textual ou ainda vocal (onde há reconhecimento de voz);
- **Linguagem de Comando**[7] - o usuário se comunica com o sistema através de comandos específicos (teclas de atalho, identificador de comando, acompanhadas ou não de parâmetros). Tende a privilegiar o usuário experiente, pois exige treinamento, mas este estilo é muito utilizado por prover uma eficiência muito grande, quando comparado com outros estilos;
- **Menus**[8] - é um conjunto de opções apresentado na tela, normalmente categorizando hierarquicamente as funcionalidades que exibe. Privilegia muito o reconhecimento e tende a ser auto-explicativo, não sendo necessária a relembrança. O mais comum de ser encontrado é o *drop-down*, que, ao ser selecionado, exibe as opções de uma categoria ou ainda o caminho para sub-categorias;
- **Preenchimento de Formulário**[7] - utilizado para entrada de informação no sistema. Muito popular na internet, utiliza a metáfora do formulário impresso, com campos para entrada de informação. Particularmente útil quando se trabalha repetidamente com os mesmos tipos de dados. A correção e prevenção de erros deve ser fator fundamental em um projeto utilizando este estilo de interação;
- **WIMP** - é o acrônimo em Inglês para Janelas, Ícones, Menus e Apontadores *(Windows, Icons, Menus and Pointers*). Exige interface gráfica e funciona através de seleção de componentes interativos da interface denominados *widgets*. Mais que um estilo de interação, é um arcabouço (*framework)* de interface gráfica. É o estilo predominante das plataformas gráficas *MS-Windows*, *Linux* e *Mac*;
- **Manipulação Direta** - permite ao usuário agir diretamente sobre os elementos representados na tela, sem comandos. As metáforas são fortemente utilizadas aqui, onde o cursor representa uma extensão da mão do operador e os objetos exibidos são considerados "reais" no espaço do sistema (pode-se tocar neles!). O usuário clica, arrasta, solta, etc. É o principal estilo utilizado na realidade virtual;

---

[7] Preece, J.; Rogers, Y.; Sharp, E.; Benyon, D.; Holland, S.; Carey, T. (1994) Human-Computer Interaction. Addison-Wesley.

[8] Paap, K.R. e Roske-Hofstrand, R.J (1988). "Design of Menus" In Helander (ed.) Handbook of Human-Computer Interaction. Amsterdam: North-Holland.

- **Agentes** - identificam necessidades do usuário e realizam tarefas em segundo plano, de maneira colaborativa.

Para escolher um estilo de interação, este deve estar de acordo com os requisitos e necessidades dos usuários, com as restrições de orçamento e com a disponibilidade de tecnologia. A escolha correta de metáforas e a prototipagem permite selecionar com razoável grau de certeza o melhor estilo a ser implementado.

2.4.2. Paradigmas de Interação

Paradigma é uma palavra grega - *parádeigma* - que significa modelo e é composto de primitivas que regulam metáforas. Simplificando, uma metáfora é realizada através de um paradigma e um paradigma só é compreendido se estiver associado a uma metáfora. Os paradigmas de interação são formas de inspiração para modelos conceituais e representam uma maneira particular de interação com um ambiente.

Nos primeiros computadores, que não tinham teclados nem monitores, a interação era feita por controles como painéis, cartões perfurados, indicadores luminosos. A interação só era possível a um especialista. À medida em que o poder de computação foi aumentando, surgiram os monitores de vídeo, os teclados, o mouse e os dispositivos de rede, como exemplos de *hardware*, e as linguagens de programação, as interfaces gráficas e os diversos aplicativos, como exemplo de *software*, que permitiram utilizar novos paradigmas para novos desafios de interação que surgiram.

Além de alguns paradigmas já citados, como o WIMP e a linha de comando, alguns novos paradigmas estão sendo pesquisados atualmente como prováveis estilos de interação homem-computador para novos ambientes computacionais.

Computação Ubíqua/Pervasiva[9] (*ubiquous/pervasive computing*) - é o paradigma onde o computador é onipresente e praticamente desaparece. Nada de realmente novo é gerado, mas tudo se torna mais rápido e fácil, ampliando o mundo perceptível ao invés de criar um novo. O paradigma se propõe a uma interação transparente entre o mundo físico e o virtual. O computador acompanha o usuário a qualquer lugar. Há uma integração de tecnologia onde a geladeira acessa a internet, o celular tira fotos, o carro informa sua localização, a casa é inteligente e sabe que seu dono chegou.

Computação Vestível[10] (*wearable computing*) - a tecnologia está integrada ao vestuário e objetos de uso. Pode-se dizer que seja o paradigma "*cyborg*", onde o próprio usuário está integrado aos componentes tecnológicos que "veste". Por exemplo, um óculos pode prover informação de um banco de dados e um sapato pode calcular a distância percorrida.

[9] M. Weiser, " The Computer of the 21st Century," Scientific American, vol. 265, no. 3,Sept. 1991, pp. 66–75 (reprinted in thisissue, see pp. 19–25).
[10] Mann,S. "Wearable Computing as means for personal empowerment"; Keynote Address for The First International Conference on Wearable Computing, ICWC-98, May 12-13, Fairfax VA, 1998

Bits Tangíveis[11], Realidade Aumentada[12] (*tangible bits, augmented reality*) - nos bits tangíveis, há uma tentativa de acoplar bits e átomos, transformando superfícies físicas (paredes, portas, janelas) em interfaces ativas entre o mundo físico e o virtual. Tenta utilizar outros sentidos humanos para melhorar a percepção (som, luz, fluxo de ar). Realidade aumentada é uma variação da realidade virtual onde o usuário vê o mundo real, com objetos virtuais sobrepostos ou compostos com os reais, fazendo o usuário achar que ambos coexistem no mesmo mundo.

Computação Transparente, Ambientes Atentos[13] (*transparent computing, attentive environments*) - são ambientes que analisam o usuário e contexto de forma transparente. O controle do usuário passa de uma forma explícita para uma forma implícita. O computador antecipa as ações do usuário, por saber quem é o usuário ou por avaliar o contexto.

2.5. Definição de Usabilidade

A usabilidade é definida como "*a capacidade que um sistema interativo oferece a seu usuário, em um determinado contexto de operação, para a realização de tarefas de maneira eficaz, eficiente e agradável*" [ISO9241, 1993].

A usabilidade tornou-se uma disciplina e uma área da engenharia, preocupada com o desenvolvimento de sistemas ao mesmo tempo úteis, eficientes e agradáveis. Para projetarmos sistemas assim, devemos focar nos objetivos de usabilidade e avaliar os sistemas utilizando as heurísticas de usabilidade de Nielsen [Nielsen, 1990].

2.5.1. Objetivos de Usabilidade

Os objetivos de usabilidade foram descritos por Jennifer Preece [Preece, 2002] e são:

- **Uso efetivo (*effectiveness*)** - faz o que tem que fazer? auxilia a realizar o desejado?
- **Uso eficiente (*efficiency*)** - quão rápido pode ser feito? quantas etapas requer?
- **Uso seguro (*safety*)** - protege o usuário de danos e frustrações. em situações potencialmente danosas, aumenta o nível de alerta; previne o erro e se, mesmo assim, ele ocorrer, permite recuperação.
- **Ter boa utilidade (*utility*)** - provê ferramentas poderosas; similar ao uso efetivo.
- **Fácil de aprender (*learnability*)** - quão fácil é aprender a utilizar o sistema ?
- **Fácil de recordar como se usa (*memorability*)** - quão fácil é recordar como o sistema funciona após ter aprendido uma vez ?

2.5.2. Heurísticas de Usabilidade de Nielsen

Similares aos princípios de projeto, provêem suporte para uma avaliação heurística da interface [Nielsen, 1990]:

---

[11] Ishii, H. and Ullmer, B. "Tangible Bits: Towards Seamless Interfaces between People, Bits and Atoms"; Published in the Proceedings of CHI ' 97, March 22-27, 1997

[12] Wellner, P., Mackay, W., and Gold, R. "Computer Augmented Environments: Back to the Real World." Commun. ACM, Vol. 36, No. 7, July 1993

[13] Norman, D. A. "The invisible computer". Cambridge, MA: MIT Press, 1998.

- **Visibilidade do estado do sistema** - o sistema deve sempre manter os usuários informados sobre o que está acontecendo, através de um retorno de informação (realimentação) em um tempo razoável.
- **Compatibilidade do sistema com o mundo real** - o sistema deve falar a língua do usuário, com palavras, frases e conceitos familiares ao mesmo, ao invés de termos orientados ao sistema em si. Siga convenções do mundo real, exibindo informação numa ordem natural e lógica.
- **Controle do usuário e liberdade** - usuários frequentemente escolhem opções do sistema por engano e necessitam de uma "saída de emergência" claramente identificada para sair do estado indesejado sem passar por um diálogo extenso. Suporte "desfazer" e "refazer" para motivar exploração.
- **Consistência e padrões** - usuário não devem precisar se preocupar quando palavras, situações ou ações diferentes signifiquem a mesma coisa. Siga as convenções da plataforma.
- **Prevenção de erros** - melhor que boas mensagens de erro é um projeto cuidadoso que previna o problema de ocorrer.
- **Reconhecimento ao invés de recordação** - faça objetos, ações e opções visíveis. O usuário não deve ser obrigado a recordar informação de uma parte de um diálogo em outra. Instruções para o uso do sistema devem estar visíveis e facilmente recuperáveis, quando apropriado.
- **Flexibilidade e eficiência de uso** - aceleradores (atalhos) - não notados pelo usuário novato - freqüentemente aceleram a interação para o usuário avançado de tal modo que o sistema possa ser utilizado tanto por usuários experientes como por inexperientes. Possibilite aos usuários agilizar ações freqüentes.
- **Estética e projeto minimalista** - diálogos não devem conter informação irrelevante ou raramente necessária. Toda unidade supérflua de informação em um diálogo compete com as unidades relevantes de informação e diminui sua visibilidade relativa.
- **Ajudar os usuários a reconhecer, diagnosticar e corrigir erros** - mensagens de erro devem ser expressadas em linguagem comum, não codificada, indicar o problema com precisão e sugerir uma solução.
- **Ajuda e documentação** - mesmo sabendo que é melhor que o sistema possa ser utilizado sem documentação, é necessário prover ajuda e documentação. Esta informação deve ser de busca fácil, focada na tarefa do usuário, listar passos concretos para a solução e não ser muito grande.

Esta avaliação heurística permite, através de um método rápido, barato e simples, avaliar uma interface de usuário.

Utilizaremos esta avaliação heurística no capítulo 3, para verificar como os padrões podem prover melhora ou degradação no projeto de uma interface de usuário.

Capítulo 3

Padrões de Projeto de Interfaces de Usuário sob a Ótica da Usabilidade

Depois de compreender o que é um padrão e como se desenvolve o projeto de interfaces de usuário, é hora de colocarmos os dois conceitos frente a frente, para descobrir o que podemos obter de proveitoso da utilização de padrões no projeto.

O modelo de interação entre o usuário e o computador é dado por Norman [Norman, 2002], que refere-se a ele como modelo de "Execução-Avaliação". Toda tarefa de um usuário passa por uma seqüência de ações que são posteriormente avaliadas pela modificação resultante da tarefa executada. A figura 5 exibe o modelo.

Figura 5. Modelo Execução-Avaliação

A meta principal no desenvolvimento de interfaces é aproximar o modelo mental do usuário do modelo do programa, construído pelos projetistas, a fim de que a usabilidade seja fator preponderante no projeto. Esta opinião é senso comum entre especialistas em IHC, como Welie e Trætteberg [Welie, 2000], Mahemoff e Johnston [Mahemoff, 1998], Hussey [Hussey, 1999] e Borchers [Borchers, 2000].

Há uma certa distância entre a Engenharia de *Software* e a Interação Humano-Computador pois, enquanto a Engenharia de *Software* requer regras precisas e procedimentos formais, a Interação Humano-Computador trabalha com conceitos mais abstratos, como diretrizes e princípios de projeto. No entanto, tal distância é danosa no momento em que a falha na compreensão dos conceitos de um lado e de outro resulta em uma interface de usuário onde não há a preocupação com a usabilidade, ou ainda, o produto não é eficiente nas tarefas as quais se propõe a apoiar.

A Lei de Murphy para a Computação é: "O computador faz tudo o que você manda, mas nem tudo o que você quer". Isto evidencia falhas na compreensão do modelo mental do usuário que, é sabido, é um modelo incompleto, vago, dinâmico, às vezes ambíguo e de difícil compreensão. Adaptar este modelo ao modelo do programa que é preciso, estático, formal e completo pode ser uma tarefa frustrante e ainda ineficiente. A proposta de termos o usuário como centro do projeto é de não adaptarmos o usuário ao programa, mas o programa ao usuário.

3.1. O Usuário Como Centro do Projeto

Lorraine Johnston [Johnston, 2001] narra um pouco da história da arquitetura civil, onde antigamente o papel do arquiteto era exatamente o de mestre-de-obras (sua origem do grego). Com o advento da Revolução Industrial, o arquiteto, que era responsável tanto pelo projeto como pela construção das edificações, começou a se responsabilizar mais pelo projeto e deixou a coordenação da construção para um novo tipo de profissional: o engenheiro. As construções começaram a se tornar impessoais, onde era mais importante a expansão industrial do que a qualidade de vida. As pessoas que se adaptassem a casas menores, poluição, barulho. Ou se adaptavam ou não faziam mais parte do jogo.

Já no século vinte, houve uma modificação no papel dos arquitetos. Havia a noção de que as construções deveriam ter uma certa consciência social e uma missão de serviço à sociedade. Ao novo arquiteto era requerido conhecer as necessidades socias e apresentar soluções viáveis a elas.

A computação não é tão antiga assim, mas sua evolução traduz séculos em décadas. Numa analogia com a arquitetura civil, estamos hoje mudando o paradigma de que qualidade de um produto é um produto sem erros e que eficiência é entendida como rapidez e precisão. No entanto, este é ainda o conceito preponderante: de que o usuário é um mero detalhe.

Como escreve Matijn van Welie [Welie, 2000]: "*Quando se toma a perspectiva do usuário, se torna importante pôr ênfase na argumentação de ' por quê' e ' como' a usabilidade é melhorada. Sem este ' rationale' , é impossível dizer quando e como a solução é ' boa' ou ' aceita' . Certas soluções no projeto de interfaces de usuário resolvem problemas que projetistas e o pessoal do marketing têm, mas não necessariamente problemas que os usuários têm. Em geral, banners e splash screens são soluções aceitas, mas dificilmente por razões de usabilidade.*"

Num projeto de interface de usuário, devemos entender que, para a maioria dos usuários, a interface é, na realidade, o próprio programa. Pouco interessa a ele se o programa foi desenvolvido em C, Pascal, ou se é cliente-servidor, monousuário; o que o usuário deseja é que o programa esteja ali pronto para suprir suas necessidades. Qualquer coisa diferente disto resulta em frustração.

A tarefa de um projeto de IHC é basicamente, criar um ambiente onde, com a devida abstração, o usuário "viva". Neste ambiente, tanto faz se em duas ou três dimensões, há uma interação direta entre o programa e o usuário, interação esta provida pela interface. Esta tarefa está em evidente constraste com o projeto do restante do programa, onde a única pessoa que pode chegar a viver ali é o engenheiro de *software*, ou ainda o programador. Para o usuário final, não há (e nem deve haver) consciência da existência desta parte do programa.

A melhor maneira de se tomar a perspectiva do usuário é fazer com que o usuário colabore no desenvolvimento do sistema. O notável não-sucesso dos padrões na arquitetura civil se explica em parte pelo fato de que a disseminação destes tirava um pouco do poder dos arquitetos e incorporava ao projeto da construção a opinião de não-profissionais, transferindo a estes uma parte do poder. Os trabalhos de Christopher Alexander acabaram sendo mais utilizados por pessoas querendo redecorar suas casas ou jardins, essencialmente os "amadores".

Ao centrarmos o projeto no usuário final, concedemos a ele poder que este não tinha antes: de influir qualitativamente no processo de projetar interfaces de usuário, como um amador projetando um jardim ou a melhor maneira de se colocar uma cerca. Com a colaboração ativa dos usuários no processo, ganhamos um importante aliado para a melhoria da qualidade do produto final.

Como obtermos colaboração produtiva do usuário dentro da formalidade requerida dentro do projeto? Os padrões de projeto podem ser uma técnica muito interessante...

3.2. Utilização de Padrões para o Projeto da Interface de Usuário

A intenção original de Alexander, quando introduziu o conceito de padrões [Alexander, 1977], foi capturar a essência de soluções bem sucedidas para problemas recorrentes na arquitetura convencional. Estes padrões descrevem aspectos do ambiente físico onde pessoas vivem e trabalham.

O desenvolvimento de interfaces de usuário depara-se, dentro de um nível de abstração, com a mesma situação: pessoas irão, de certa maneira, viver e trabalhar no ambiente criado pela interface e esta interface se torna o próprio mundo onde a interação irá ocorrer. Não é físico, é virtual, mas a idéia é a mesma.

Enquanto que nas demais partes de um programa, o projetista pode variar seu modelo para maximizar desempenho, segurança, etc., nas interfaces a base é a experiência do usuário. Nisto, podemos estabelecer alguns aspectos que um padrão de projeto de interface deve ter:

- **centrado no usuário**: ele é quem tem a palavra final se a interface é boa ou não e é ele quem vai utilizar ou abandonar o sistema projetado com base, na maior parte dos casos, apenas em sua interface;
- **compreensível ao usuário**: devem endereçar tanto o público especializado (engenheiros de *software*, analistas, programadores) como o público leigo (o usuário por excelência); para isto, devem ser escritos em linguagem comum, com diagramas que sejam compreensíveis para alguém sem treinamento específico;
- **incentivar colaboração entre o projetista e o usuário**: através de uma linguagem comum, os usuários podem expressar suas próprias preferências de projeto, mesmo dependentes do mestre-de-obras, que é o projetista;

- **formalidade adequada**: devem ser concisos, para que fique evidente que não se trata de interação pessoa-pessoa, mas de interação pessoa-computador; as expectativas são bem diferentes;
- **trazer evidência empírica de seu sucesso**: padrões que não permitam medição de sua aplicabilidade, devem ser evitados;
- **observar o aspecto temporal**: de acordo com Jan Borchers [Borchers, 2000], *"os artefatos que nós, projetistas de interfaces de usuário, criamos , mudam substancialmente através do tempo, obedecendo as tarefas que apoiam. (...) nós projetamos interfaces em uma dimensão temporal tanto quanto em duas ou três dimensões espaciais"*. Padrões devem evidenciar o processo, não somente um momento no tempo;
- **ser seguro**: nas palavras de Andrew Hussey [Hussey, 1999], "*um sistema é seguro quando o risco de um problema acontecer e gerar danos é aceitavelmente baixo*". Padrões devem conduzir a sistemas seguros, tanto do ponto de vista do usuário, como do ponto de vista do projetista;
- **prover usabilidade**: deve prover melhora nos indicadores de usabilidade, como facilidade de aprendizagem, satisfação do usuário, completude de tarefas, eficiência, etc. De acordo com van Welie [Welie, 2000], "*se um padrão de interface de usuário não melhora pelo menos um indicador de usabilidade, ele não é um padrão de interface de usuário*".

O formato proposto por Matijn van Welie [Welie, 2000], mais próximo da Forma Alexandrina do que da Forma GoF, contém os seguintes elementos:

- **Problema**: relacionados com a utilização do sistema e relevantes aos usuários. Em contraste com padrões da engenharia de *software*, problemas em padrões de interfaces de usuário não focam em aspectos construcionais do sistema, como classes de objetos, mas orientados às tarefas que deverão ser realizadas pelos usuários;
- **Princípio de Usabilidade**: padrões de interação normalmente utilizam pelo menos um "princípio", no qual a solução é baseada. Não se conhece o conjunto completo de princípios, mas podemos utilizar a lista proposta por Norman [Norman, 2002]:
  - Visibilidade, Affordance, Mapeamento Natural, Restrições, Modelos Conceituais, Realimentação, Segurança, Flexibilidade;
- **Contexto**: focado no usuário, o contexto é descrito em termos de tarefas, usuários e ambientes nos quais o padrão é aplicável;
- **Forças**: descreve todos os fatores que influenciam o projeto, direta ou indiretamente; podem ser forças de tarefas, usuário ou contexto;
- **Solução**: a solução deve ser concisa e não impôr novos problemas. Por esta própria concisão, às vezes a solução depende de outros padrões relacionados para ser viável. Estes padrões devem constar do elemento Padrões Relacionados;
- ***Rationale* (Motivação)**: descreve como o padrão funciona, porque funciona e porque é bom. A diferença entre a solução e o *rationale* é que enquanto a solução descreve a estrutura visível e o comportamento do sistema, o *rationale* provê uma visão da estrutura interna e os mecanismos-chave que agem debaixo da superfície do sistema. O *rationale*

discute o impacto do padrão na usabilidade, ao descrever quais aspectos da usabilidade são melhorados e quais são piorados. Dentre tais aspectos, estão:
- Desempenho, Facilidade de Aprendizado, Recordação, Satisfação, Completude de Tarefas, Erros;

- **Exemplos**: mostram como o padrão pode ser utilizado com sucesso em um sistema; pode ser uma figura ou um texto explicando um uso particular do padrão. São preferíveis exemplos reais, para que a validade do padrão seja reforçada;
- **Usos Conhecidos**: exemplos de produtos que já utilizam a solução;
- **Padrões Relacionados**: os padrões que, junto com este, compõem a solução;
- **Contra-Exemplos**: é um exemplo onde o padrão deveria ter sido aplicado e não foi ou foi usado onde não deveria ter sido. Funciona como um anti-padrão e serve como motivação extra para o uso do padrão.

Ao escrever padrões de interface de usuário, deve-se ter sempre em mente que o ponto de vista tomado é sempre o do usuário e não o do projetista. Quando diretrizes são reescritas no formato dos padrões, é tentador cair neste erro. O exemplo de van Welie [Welie, 2000] de escrever padrões sobre "Como Utilizar Controles com Abas" mostra como o problema não deve ser tratado partindo da solução sem compreender o problema propriamente dito:

"(...) Por exemplo, se a visão é incorreta, alguém pode escrever padrões do tipo ' Como Utilizar Controles com Abas' . Isto é muito tentador de fazer, especialmente quando estamos reescrevendo diretrizes no formato de padrões. Entretanto, tais visões tomam a perspectiva do projetista e não a do usuário. Além do mais, o conhecimento de projeto sobre ' Como Utilizar Controles com Abas' depende do contexto em que é aplicado, dos usuários e suas tarefas. Em outras palavras, olha para o problema partindo da solução sem conhecer o problema."

A sugestão é criar o padrão obedecendo a quatro etapas:

I - primeiramente **formule** o padrão, abstraindo aspectos particulares que são irrelevantes e focando no núcleo da solução;

II - **classifique** o padrão, para precisar o que ele realmente supre;

III - **selecione** padrões para uso a fim de compreender como o padrão melhor se encaixa no problema que o projetista está tendo;

IV - finalmente, **aplique** o padrão e verifique se o que é proposto pelo padrão realmente auxilia no fragmento de programa que está sendo implementado.

Seguindo estas linhas, é possível obter uma linguagem de padrões que possa suprir se não todas, pelo menos as dificuldades mais comuns com que nós nos deparamos em projetos de interface de usuário.

3.3. A Usabilidade nos Padrões

Durante o processo de criação de um *software*, partimos de um modelo mais abstrato, de alto nível, para uma instância específica, restrita a determinada plataforma, nível de *expertise* do usuário, tecnologia e paradigma de interação. Em qualquer momento, falhas de projeto podem ser inseridas ou corrigidas, afetando a qualidade geral do produto.

Quanto mais cedo uma falha for detectada, menos cara vai ser a correção. Conforme a entrega do produto se aproxima, o custo de correção, para mantermos o nível de qualidade proposto, tende a ser maior.

O grande problema com projetos de interface no que se refere à usabilidade é que normalmente esta só é verificada nas últimas fases do projeto, quando o produto está quase pronto e se torna possível disponibilizar uma versão (alfa, beta, ...) de teste para o usuário final. Erros encontrados neste momento quase sempre são caros demais a ponto de comprometer o cronograma do projeto como um todo.

Os padrões de projeto, por prover um alto nível de abstração, permitem tratar a usabilidade em fases anteriores do projeto, diminuindo o custo e aumentando sensivelmente a qualidade do mesmo.

Na tabela 5, é feita uma análise sobre como os padrões de interface podem melhorar ou piorar um projeto de interface de usuário, baseando-se na avaliação heurística da interface [Nielsen, 1990]. A coluna à direita indica se padrões podem melhorar (+ e ++) ,piorar (- e --) ou ainda, não afetar sensivelmente a qualidade do produto final (N/A). Os padrões citados aqui estão detalhados no Apêndice A:

Tabela 5

Heurística de Usabilidade	Abordagem por Padrões	
Visibilidade do estado do sistema	Por observar o aspecto temporal, um padrão de projeto permite um rápido reconhecimento do estado do sistema. Padrões como *Progress (Barra de Progresso)*, *Hinting (Pistas)*, *Contextual Menus (Menus Contextuais)* são compreensíveis ao usuário, informando o que está acontecendo, o que pode acontecer dentro de um período de tempo e, dependendo da tarefa, como sair da execução da tarefa antes do seu término.	**+**
Compatibilidade do sistema com o mundo real	A interface, a princípio, define seu próprio ambiente e não necessita ser completamente compatível com o mundo real. Às vezes, a metáfora aplicada não necessita ter uma relação um-para-um com a realidade, desde que a tarefa desejada seja suficientemente compreendida pelo usuário. Exemplo disto é o ícone de disquete, para salvar um arquivo, em um processador de texto. Raramente grava-se em disquetes, mas a metáfora provê a associação entre os modelos e dispensa maiores compatibilidades. Padrões, por seu caráter empírico, suprem a compatibilidade somente onde usuários já precisaram de tal compatibilidade. Além disto, padrões não dependem do paradigma de interação e, ao tratar interfaces textuais, a compatibilidade com o mundo real é evidentemente prejudicada. Apesar disto, temos padrões que	**N/A**

	tratam exatamente deste princípio, como *o Like in the real world (Como no mundo real)* e *The Preview (A Prévia)*. Em uma visão mais abrangente, padrões não agregam valor neste quesito.	
Controle do usuário e liberdade	Transferindo poder ao usuário ainda na fase de projeto, este tem maior peso de opinião antes do produto estar pronto. Isto se traduz em um produto com maior qualidade e é sensível ao usuário o conceito de que o produto fornece a ele maior controle. Para exemplificar melhor, pode-se utilizar o padrão *The Stranger (O Estranho)*, que trata o problema onde "cada usuário é diferente e prefere fazer coisas a seu modo". Os padrões fornecem, na verdade, uma liberdade assistida, onde o usuário sente-se livre para personalizar sua interface, mas somente nos elementos que não possam comprometer a eficiência e completude da tarefa à qual se propõe.	**+**
Consistência e padrões	Provendo um meio padronizado de projetar, garante-se a consistência tanto no projeto da interação como na comunicação com o cliente. É importante frisar que, ao prover um vocábulário comum entre o usuário final e o projetista, aumentando a colaboração durante todo o ciclo de vida do produto, a consistência se torna natural. O usuário, ao finalmente ter o produto em mãos, sabe o que esperar e que não esperar. A consistência aumenta na mesma medida em que a frustração na entrega diminui.	**++**
Prevenção de erros	Padrões são essencialmente empíricos. A conclusão imediata desta afirmação é que erros comuns do passado devem estar corrigidos. A criação de um padrão é reativa. Cria-se um padrão de projeto para fornecer uma solução contextualizada a um determinado problema. Problemas recorrentes que permitem soluções igualmente recorrentes é a essência do conceito de padrões, portanto, este princípio de usabilidade é tratado com rara eficiência pelos padrões. Um exemplo disto é o padrão *The Shield (O Escudo)*, que protege o usuário de realizar algo irreversível. Ele requer que o usuário cometa dois erros em vez de um só, para causar danos irreversíveis.	**++**
Reconhecimento ao invés de recordação	Apesar de não ter seu foco no mundo real (mas no usuário), os padrões são empíricos e compreensíveis ao usuário. Quando o usuário se vê frente a frente com um *Wizard (Assistente)*, recordação não é necessário. O usuário reconhece que está diante de uma tarefa composta de vários passos. Do mesmo modo, o padrão *The Preview (A Prévia)* não exige do usuário recordar o conteúdo de determinado arquivo, exibindo uma prévia do conteúdo sem que seja necessário abrir arquivo por arquivo.	**+**
Flexibilidade e eficiência de uso	Por não se basearem em minúcias da solução, permitem flexibilidade por deixarem ao gosto do usuário como a interface deve parecer. Isto traz eficiência, se pensarmos que uma boa interface, projetada com padrões voltados para a flexibilidade, como	**+**

	o *The Stranger (O Estranho)* e *The Grid (A Grade)* irá fazer com que o produto abranja uma diversidade maior de usuários. Conforme se torna especialista no produto, o usuário vai refinando sua interação e otimizando a usabilidade da interface.	
Estética e projeto minimalista	Padrões são minimalistas por excelência. Os detalhes de implementação não são tratados por eles, mas o conceito por trás da interação. A reutilização provida pelos padrões faz com que as minúcias de um problema não sejam explicitadas pelo padrão, e tornam, assim, a solução mais simples e abrangente. Uma solução simples e abrangente é por si só minimalista, tanto no seu projeto, como na estética da interface. Há uma espécie de repulsa a um comportamento anti-minimalista ao utilizarmos padrões, pois o contexto o impede: tendo o usuário como foco, queremos que a tarefa seja cumprida da maneira mais eficiente possível e isto impede de nos valermos de sons, efeitos visuais, passos desnecessários.	**++**
Ajudar os usuários a reconhecer, diagnosticar e corrigir erros	Norman [Norman, 2002] define dois tipos de defeitos: os erros (mistakes) e os deslizes (slips). Erros são deliberados, conscientes, enquanto que deslizes ocorrem por descuido ou falta de informação. Comunicando-se através de vocabulário comum, os padrões conduzem o usuário por caminhos já comprovadamente seguros. Cabe lembrar que o usuário participa do projeto, portanto vários defeitos são sanados logo na fase de projeto e as deficiências do usuário são detectadas e minimizadas. Padrões como *Warning (Aviso)* auxiliam o usuário a reconhecer, diagnosticar e, se mesmo assim o erro ocorrer, desfazer o mesmo, voltando a um estado anterior sem erros.	**++**
Ajuda e documentação	Padrões são uma documentação em si, trazendo uma linguagem de fácil acesso tanto ao usuário final como ao projetista. O formalismo sem exageros proposto pelos padrões documenta o produto, criando uma interface mais orgânica, em que a ajuda se traduz em colaboração, visibilidade, compatibilidade e eficiência. Quanto melhor a interface, menos ajuda (explícita) o usuário requer.	**++**

3.4. O Diferencial Provido Pelos Padrões

Padrões não são o único formato de comunicar experiência no projeto de interfaces de usuário. Podemos, para isto, utilizar normas de interface, guias de estilo, diretrizes, regras áureas, mas há um diferencial provido pelos padrões de projeto de interface que se traduz em uma qualidade e uma abrangência maior do que o obtido com outros formatos.

Guias de estilo, como as elaboradas pela *Apple*, *IBM*, *Microsoft*, são eficientes em assegurar consistência em uma plataforma específica. No entanto, quando se tenta utilizar o mesmo guia em outra plataforma (ou *toolkit*), apesar de se ter em mãos um formato concreto e construtivo, ele é muito enraizado à plataforma de origem. Os padrões sobrepujam os guias de estilo por prover independência de plataforma.

Diretrizes e regras áureas, como as de Ben Shneiderman [Shneiderman, 1998], são úteis para categorizar deficiências de um projeto de interface e localizar qual a regra que foi violada, mas falham em não apresentar exemplos concretos de sua utilização e ainda não são construtivos (não ajudam os projetistas em novos sistemas). São de difícil interpretação e sua simplicidade causa um efeito reverso: não podem ser utilizadas por não-especialistas.

Normas de interface, como as da ISO [ISO9241, 1993], conseguem estabelecer regras rígidas sobre um estilo de interface estabelecido, mas se o paradigma de interação mudar, elas devem ser reescritas. Como não provêem exemplos concretos e não são acessíveis a um público leigo (entenda-se, o usuário), as normas não são centradas no usuário e com isto falham em vários princípios básicos de usabilidade. A vantagem de se utilizar uma norma é principalmente quando se deseja certificar um produto, mas como o foco deste trabalho é o usuário, os padrões revelam-se como um formato mais eficiente de consistência.

Padrões não são de caráter tão geral como diretrizes, nem tão específicos como guias de estilo e normas de interface, mas exibem um diferencial na medida que as maiores forças que influenciam o projeto de interface, como o usuário, o contexto, a tarefa e o *rationale*, são tratadas, capturadas e documentadas pelos padrões.

Outro diferencial importante é que, por proverem uma linguagem compreensível tanto para o especialista como para o leigo, os padrões são uma ferramenta útil para educar tanto o projetista novato, como o usuário final. A colaboração e reutilização que esta compreensão proporciona traz embutida uma melhora implícita na qualidade do produto final.

Finalmente, e talvez o diferencial mais importante, é o aspecto atemporal dos padrões de projeto. Documentar requer esforço e encontrar soluções boas e construtivas requer mais esforço ainda. Padrões de projeto, ao abstrair as particularidades, capturar a essência da solução e serem razoavelmente invariantes através do tempo, permitem que o custo do processo de armazenamento de experiência se dilua através do tempo.

Padrões de projeto de interface são, enfim, soluções que, em sua maior parte, estão em um nível de abstração maior que o dos próprios paradigmas de interação. Isto significa que as mesmas forças que atuam nos paradigmas de interação continuarão atuando nos paradigmas futuros, fazendo com que a utilização de padrões de projeto funcione como um caminho seguro no desenvolvimento de novos sistemas, interfaces e paradigmas.

Capítulo 4
Conclusões e Trabalhos Futuros

O que mais fascina em padrões de projeto é a capacidade de armazenamento de experiência em uma linguagem acessível, onde tanto o projetista experiente, quanto o novato e o usuário final, todos participam do processo.

No caso de interfaces de usuário, os padrões trazem benefícios à usabilidade de tal maneira que é difícil não utilizá-los. Só o fato de trazerem consistência e prevenção de erros já os qualifica como uma ótima ferramenta no projeto de interfaces de usuário. Suas raízes na arquitetura civil também embasam a noção de espaço habitável, que as metáforas acabam transmitindo.

Mas o grande trunfo dos padrões é que eles propõem um projeto de interfaces de usuário com características que suprem não só o desenvolvimento atual, mas também os paradigmas de interação vindouros, talvez até com maior eficiência.

4.1. Interfaces Afetivas

Donald Norman, em *Emotional Design* [Norman, 2004], traz um novo conceito, onde verificada a tamanha complexidade do cérebro humano, podemos dividir os mecanismos cerebrais em três níveis, esquematizados na Figura 6: visceral, comportamental e reflexivo, que refletem as origens biológicas do cérebro.

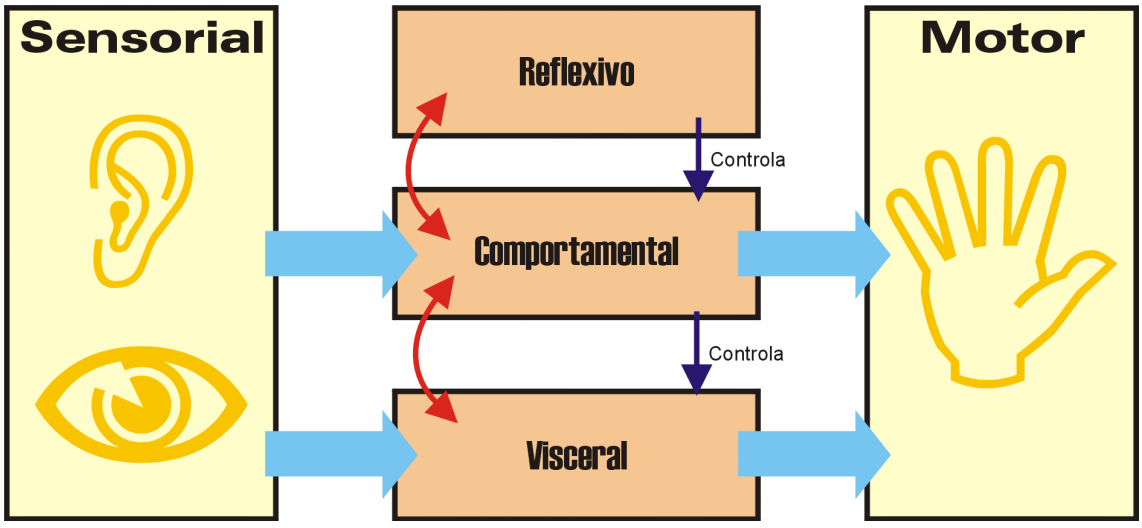


Figura 6. Os Três Níveis de Processamento

No nível visceral, encontram-se as respostas inatas humanas, quase automáticas. É um comportamento rápido, para decidir em um tempo mínimo se algo é bom ou ruim, seguro ou perigoso. É o nível básico, das rotinas fixas, existente tanto em humanos como em animais inferiores. Lagartos, por exemplo...

O próximo nível, comportamental, ainda que não consciente, traz uma capacidade de análise mais sofisticada. É o nível mental onde nossas habilidades, como tocar um instrumento musical, dirigir ou utilizar um editor de texto, estão. Este nível pode inibir ou reforçar as respostas do nível visceral e ainda, ser inibido ou reforçado pelo nível superior. Os princípios de usabilidade tratam essencialmente deste nível.

O nível mais alto, o reflexivo, não tem acesso direto aos sensores corporais ou aos órgãos motores. Neste nível se localiza o pensamento consciente e é presente somente nos organismos mais evoluídos, como nós, humanos. O ser humano, reflete sobre o ocorrido, aprende e transmite novos conceitos e generalizações do mundo.

Estes três níveis compõem nossa resposta afetiva a algo. Muitos produtos podem não ser eficientes, podem ser bonitos ou feios, podem até ser de difícil utilização ou aprendizagem, mas o ser humano, às vezes chamado de usuário final, elege características destes produtos que ainda não podem ser classificadas, porque envolvem aspectos do nível reflexivo, muito pouco ou quase nada atendido pela engenharia atual.

Enquanto em alguns produtos podemos impor seu uso mediante a análise de sua eficiência ou usabilidade, na maioria deles o foco no usuário traz conceitos ainda não compreendidos, onde nos valemos muito mais da heurística e da experiência anterior, do que de conceitos formais e normas. O sucesso ou o fracasso de um produto está ancorado única e exclusivamente na sua capacidade de responder às necessidades do usuário alvo. Ele precisa despertar, de alguma maneira, o afeto do usuário.

Norman discorre sobre os monitores coloridos: não há razão estritamente técnica para os monitores serem coloridos, na maior parte dos casos. Em um livro anterior, ele explica porque não precisamos de cor em monitores e neste livro ele empresta um monitor colorido para provar sua teoria. Final da história: ele prova completamente a teoria, mas não quer mais devolver o monitor, porque a cor trouxe, de alguma maneira, maior afeto do que o monitor monocromático.

Os padrões de projeto trazem em si algo disto, através do seu *rationale*. O *rationale*, ou a motivação de se utilizar determinado padrão pode conter informação sobre utilizações anteriores, até este se tornar um padrão documentado. Esta experiência anterior pode conter elementos que, abrangendo os três níveis cerebrais, incentivem o afeto que o usuário terá com o produto.

4.2. Interfaces Vivas

É um fato: todos gostam de golfinhos. Não há razão óbvia para isto, aliás, existem razões para que haja um sentimento oposto. Golfinhos têm dentes afiados, são molhados, não têm pelo macio, vivem em um ambiente que desperta medo em algumas pessoas e, obviamente, ninguém tem um golfinho em casa, como animal de estimação. Está provado que nem animais "ingênuos" e "bons" eles são. Há registros de matança organizada entre eles, manipulação e outras "maldades".

Então, por que todos gostam dos golfinhos? Vai contra o bom senso? Nem tanto... Os golfinhos, apesar de sua "interface" ser pobre, para os padrões que definimos, têm uma característica essencial que amplifica o afeto humano por eles: o interesse por nós.

"Que importa se eu não tenho um golfinho em casa? Eu continuo gostando de golfinhos e é um prazer indescritível ver ou , melhor, estar perto de um.", diz o homem comum.

Estamos em um processo onde as interfaces entre pessoas e computadores começam a dar mostras que deve haver, de alguma maneira, o sentido de interesse um pelo outro. Para isto, as interfaces devem trazer organicidade, elementos onde pelo menos haja a sugestão de comportamentos que só seres vivos têm.

Os padrões trazem isto. A dualidade cognitiva-afetiva dos padrões estimula os projetistas a desenvolverem interfaces mais criativas e com um princípio de "vida" embutido. Do outro lado, o reconhecimento rápido e a diminuição da frustração dos usuários traz a estes um sentimento de afeto pela interface como se, dentro das devidas comparações, elas estivesse viva e cooperando com o usuário.

Ainda não fazemos programas assim, mas um dia este será o modelo de bom programa.

4.3. Balas de Prata

Dentro do imaginário popular, balas de prata são maneiras muito eficientes para exterminar lobisomens e vampiros. Vampiros ainda podem ser destruídos por estacas de madeira, mas lobisomens somente com balas de prata. Até prova em contrário, nem lobisomens nem vampiros existem, mas as balas de prata são procuradas para exterminar outros "monstros".

No jargão científico, "bala de prata" é um termo utilizado quando algo, seja um artefato ou uma técnica, consegue solucionar um problema de uma maneira cabal. Ainda no meio científico, é quase unanimidade de que não existem tais balas de prata. Contra o que não entendemos completamente, não existe remédio. Só nos resta contar os mortos e torcer para que não sejam muitos.

Quando projetamos uma interface de usuário, o fazemos baseando-nos em metáforas que aproximam o modelo mental do usuário do modelo mental do projetista materializado no modelo do sistema. É sabido que metáforas e modelos mentais não são completamente compreendidos. Eles trazem dentro de si ambigüidades, indefinições, variabilidades, enfim, uma carga de informações que continuarão desconhecidas, haja o que houver.

O que precisamos é de balas de prata para "exterminarmos" estas bestas-feras que habitam nossos projetos. E é aí onde os padrões surpreendem.

Padrões de projeto podem não ser balas de prata genéricas, mas contextualizando o vampiro, ou o lobisomem, o que temos é uma solução que realmente resolve o problema, sem no entanto, tentar compreender o problema em suas minúcias e variâncias. Pouco importa o nome do lobisomem ou os decibéis de seu uivo, se pudermos eliminá-lo, a história terá um final feliz.

Uma linguagem de padrões pode, dentro de um contexto adequado, ser considerada como uma cartucheira de balas de prata, de grande utilidade e eficiência. Para desafios que teremos que enfrentar no desenvolvimento de interfaces em futuros projetos, é sempre bom estar bem armado...

4.4. QWAN

Cristopher Alexander [Alexander, 1979] define o QWAN, ou "*Quality Without a Name*", ou ainda "Qualidade Sem Um Nome", como sendo "*uma qualidade central que é a raiz da vida e espírito de um homem, uma cidade, uma construção, da natureza. Tal qualidade é precisa e objetiva, mas não possui um nome*". Esta "qualidade" traz a uma estrutura um valor imensurável e uma beleza incomunicável, que é universalmente reconhecível. Ela independe de cultura e de situações, assim como do tempo.

Este QWAN proporciona uma conexão quase emocional com a estrutura projetada. Proporciona àqueles que interagem com ele um sentimento de completude, de conforto, de sentirem-se vivos. Os artefatos que o possuem são flexíveis, extensíveis, adaptáveis, reutilizáveis, enfim, possuem qualidades que de um certo ponto de vista, emulam vida.

Esta definição é demasiada subjetiva para se adequar aos paradigmas atuais de engenharia de *software*. O QWAN opera no nível visceral da inteligência humana e as disciplinas da engenharia almejam o nível comportamental. Atualmente, não há compatibilidade o suficiente para a co-existência, mas isto tende a mudar.

A computação se torna cada vez mais ubíqua. A qualidade é cada vez mais declamada como não somente desejável, mas necessária a qualquer projeto.

O futuro das interfaces de usuário volta-se finalmente para o usuário. Será necessário fornecer maior poder, maior influência ao usuário no projeto. Os padrões de projeto proporcionam isto.

Os sistemas computacionais que interagirem com seres humanos deverão, cada vez mais, apresentar qualidades que o identifiquem como um ser consciente de sua existência. A interação entre seres humanos e computadores ruma para uma maior colaboração, quase uma simbiose.

Para haver colaboração, os sistemas devem estar preparados para a variedade de usuários que irão utilizá-los. Mais, devem evidenciar a sensação de que é imediata a sua compreensão e devem reforçar o componente afetivo do usuário.

Ainda não chegamos a este ponto, mas os padrões de projeto parecem conter as qualidades necessárias para, através do Caminho, atravessarmos o Portal, e alcançarmos o QWAN.

Referências

[Alexander, 1977] C. Alexander, S. Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-King and S. Angel, ***A Pattern Language****;* Oxford University Press, New York, 1977

[Alexander, 1979] C. Alexander, ***The Timeless Way of Building****;* Oxford University Press, New York, 1979

[Appleton, 2000] B. Appleton, ***Patterns and Software: Essential Concepts and Terminology****;* localizado em < http://www.bradapp.net > , 2000

[Beck, 1996] K. Beck, ***Smalltalk Best Practice Patterns;*** Prentice Hall, 1996

[Beck, 2002] K. Beck and E. Gamma, ***JUnit - A Cook's Tour****;* localizado em < http://junit.sourceforge.net/doc/cookstour/cookstour.htm > , 2002

[Borchers, 2000] J. Borchers, ***Interaction Design Patterns: Twelve Theses****;* CHI2000 Patterns Workshop, The Hague, 2-3 April, 2000

[Coplien, 1992] J. O. Coplien, ***Advanced C++ Programming Styles and Idioms;*** Reading, MA: Addison-Wesley, 1992

[Coplien, 1996] J. O. Coplien, ***Software Patterns;*** New York (NY), SIGS Books, 1996

[Duell, 1997] M. Duell, ***Non-Software Examples of Software Design Patterns;*** Object Magazine, Vol. 7, No. 5. July 1997, pp. 52-57

[Erickson, 1990] T. D. Erickson , ***Working with Interface Metaphors. The Art of Human Computer Interface Design;*** B. Laurel, ed. Reading MA: Addison-Wesley, 1990

[Folmer, 2003] E. Folmer and J. Bosch, ***Usability Patterns in Software Architecture****;* Department of Mathematics and Computer Science, University of Groningen, NL, Accepted for HCI International 2003

[Gabriel, 1996] R. P. Gabriel, ***Patterns of Software: Tales from the Software Community****;* Oxford University Press; April 1996

[GoF, 1995] E. Gamma, R. Helm, R. Johnson and J. Vlissides, ***Design Patterns: Elements of Reusable Object-Oriented Software****;* Addison-Wesley, Reading, MA, 1995

[Granlund, 2001] A. Granlund, D. Lafrenière and D. A. Carr, ***A Pattern-Supported Aproach to the User Interface Design Process****;* Proceedings of HCI International 2001 9th International Conference on Human-Computer Interaction, August 5-10, 2001, New Orleans, USA

[Johnston, 2001] L. Johnston, ***Learning from Traditional Architects;*** Lorraine Johnston School of Information Technology, Swinburne Computer Human Interaction Laboratory, Swinburne University of Technology. Hawthorn, Australia, 2001

[Koenig, 1995] A. Koenig, ***Neither true nor false;*** C++ Report, SIGS, Vol. 7, No. 8, November/December 1995

[Hussey, 1999] A. Hussey, ***Patterns For Safety and Usability in Human-Computer Interfaces;*** Technical Report No 99-05, University of Queensland, Australia 1999

[ISO9241, 1993] ISO 9241 Part 11 ***Ergonomic requirements for office work with visual display terminals, Part 11 Usability Statements;*** Draft International Standard ISO 9241-11,1993

[Madsen, 1994] K.H.Madsen, ***A Guide to Metaphorical Design;*** Communications of the ACM, 37(12):57–62, December 1994.

[Mahemoff, 1998] M. J. Mahemoff and L. J. Johnston, ***Principles for a Usability-Oriented Pattern Language;*** In Proceedings of Australian Computer Human Interaction Conference OZCHI' 98, Adelaide, 1998

[Microsoft, 2003] Microsoft Corporation, ***Enterprise Solution Patterns Using Microsoft .NET;*** Microsoft Press, September 2003

[Nielsen, 1990] J. Nielsen, and R. Molich, ***Heuristic Evaluation of User Interfaces;*** Proc. ACM CHI' 90 Conf. (Seattle, WA, 1-5 April), 249-256, 1990

[Nielsen, 1994] J. Nielsen, ***Usability Engineering***, Morgan Kaufmann, 1994

[Norman, 2002] D. A. Norman, ***The Design of Everyday Things;*** Basic Books, 2002

[Norman, 2004] D. A. Norman, ***Emotional Design: Why We Love (Or Hate) Everyday Things;*** Basic Books; 2004

[Preece, 2002] J. Preece, et al., ***Interaction Design: Beyond Human-Computer Interaction;*** John Wiley & Sons, 2002

[Shneiderman, 1998] B. Shneiderman, ***Designing the User Interface***, Addison-Wesley, 1998

[SIGCHI, 1992] T. T., Hewett, R. Baecker, S. Card, T. Carey, J. Gasen, M. Mantei, G. Perlman, G. Strong, and W. Verplank. ***ACM SIGCHI Curricula for Human-Computer Interaction;*** New York: The Association for Computing Machinery, 1992.

[Spolsky, 2002] J. Spolsky, ***User Interface Design for Programmers;*** APress, 2001

[Welie, 2000] M. van Welie and H. Troetteber, ***Interaction Patterns in User Interfaces;*** PloP 2000 Conference, 2000

Apendice A
Exemplos de Padrões de Projeto para IHC

A.1. The Wizard (O Assistente) by Martijn van Welie

Problema
O usuário deseja alcançar um objetivo, mas várias decisões devem ser tomadas antes que o objetivo seja completado, decisões estas que podem não ser conhecidas do usuário.

Princípio de Usabilidade
Ajuda ao Usuário (Visibilidade)

Contexto
Um usuário não especialista necessita executar um tarefa complexa e infreqüente consistindo de várias subtarefas onde cada uma destas subtarefas requer tomada de decisão. O número de subtarefas deve ser pequeno, normalmente entre 3 e 10.

Forças
• O usuário almeja o objetivo, mas pode não estar familiarizado ou interessado nas etapas para a conclusão.

• As subtarefas podem necessitar de ordenação, ou seja, uma certa tarefa necessite que outra esteja concluída naquele ponto.

• Para alcançar o objetivo, vários passos serão tomados, mas o número exato de passos necessarios pode variar, por conta das decisões tomadas em passos anteriores.

Solução
Guie o usuário através da tarefa, um passo de cada vez. Deixe o usuário "passear" através da execução e exiba em qual passo ele está e quantos serão no total.

Quando a tarefa complexa começar, o usuário é informado sobre o objetivo que será alcançado e o fato de que várias decisões serão tomadas. O usuário pode ir ao próximo passo usando um elemento de navegação (por exemplo, um botão). Se o usuário não puder iniciar a próxima tarefa antes de completar a atual, ele é informado que não pode prosseguir (por exemplo, desabilitando o elemento de navegação). O usuário deve poder revisar uma decisão já tomada.

Aos usuários é dado retorno sobre o propósito de cada subtarefa e os usuários podem ver a qualquer momento onde estão na seqüência. Quando a tarefa complexa estiver completa, o usuário é informado sobre sua finalização e, opcionalmente, o resultado do que foi processado.

Usuários que saibam as opções padrão podem imediatamente utilizar um atalho para possibilitar que tudo seja feito em uma ação. A qualquer momento da seqüência deve ser possível cancelar a operação, escolhendo uma saída visível.

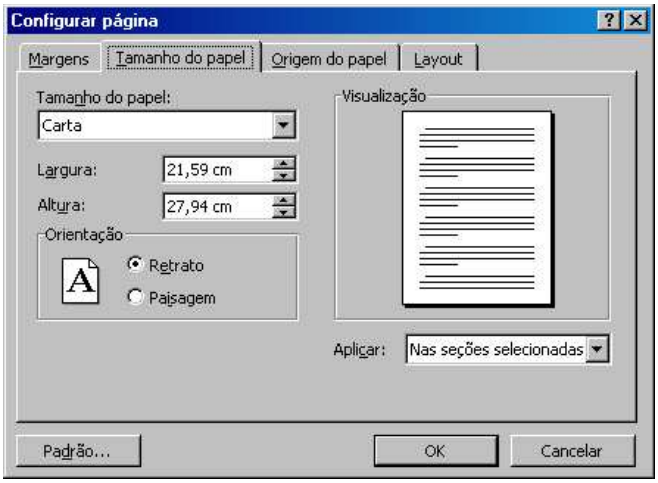
Rationale
Os elementos de navegação sugerem ao usuário que eles estão navegando em um caminho composto de etapas. Cada tarefa é apresentada de modo consistente, reforçando a idéia de que várias decisões serão tomadas. A seqüência de tarefas informa ao usuário o seu progresso. A aprendizagem e recordação da tarefa é melhorada, mas pode haver um efeito negativo sobre o desempenho da mesma. Quando usuários são forçados a seguir uma certa ordem de tarefas, eles ficam menos propensos a esquecer coisas importantes e cometem menos erros.

Exemplos
Este é Assistente para Viagem do PowerPoint. O usuário deseja empacotar uma apresentação de modo que possa apresentá-la em outro computador. Várias decisões relevante devem ser tomadas e o assistente auxilia o usuário a tomar tais decisões. O quadrado verde indica o ponto de execução da tarefa.

Usos Conhecidos	Assistente para Viagem do PowerPoint; Programas de instalação
Padrões Relacionados	NAVIGATING BETWEEN SPACES, BROWSING A LIST

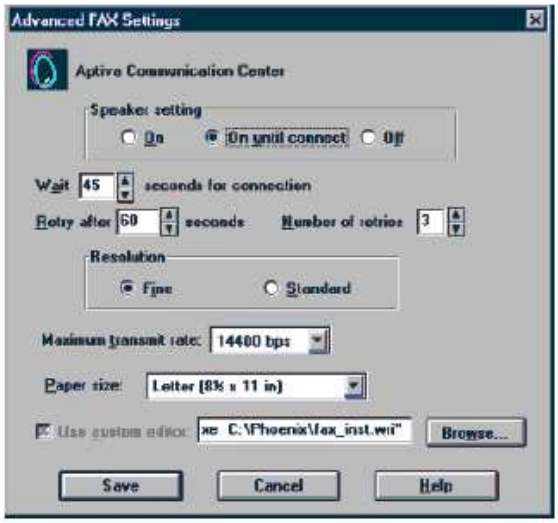
A.2. The Grid (A Grade) by Martijn van Welie

Problema	O usuário necessita entender rapidamente a informação e agir baseado nesta informação.
Princípio de Usabilidade	Consistência, Familiaridade (Modelos Conceituais)
Contexto	Qualquer situação onde muitos objetos de informação são exibidos e distribuídos espacialmente em uma área limitada. Tipicamente em telas de diálogo, formulários e páginas web.
Forças	• O usuário precisa visualizar muitos objetos, mas de forma organizada. • O usuário quer minimizar o tempo necessário para analisar os objetos na tela. • Os objetos têm alguma relação entre si e podem ser agrupados conceitualmente. • A disposição precisa ser compacta, mas clara, agradável e legível.
Solução	Distribua os objetos em uma grade, com o número mínimo de linhas e colunas, fazendo as células o mais largas possível. Os objetos do mesmo tipo devem ser alinhados e exibidos da mesma maneira. Se vários objetos podem ser agrupados, a grade é aplicada ao nível dos grupos também. Elementos pequenos podem ser alargados, adaptando-se aos limites da grade. Elementos maiores podem utilizar mais de uma célula da grade. Ceros objetos podem ter um tamanho fixo, o que aumenta o número de linhas e colunas para que eles possam manter seu tamanho. Botões de resposta padrão devem estar em posições pré-definidas e podem ser acessados de fora da grade.
Rationale	Minimizando o número de linhas e colunas, melhora o tempo necessário para adquirir a informação e tomar a decisão apropriada. Diminuindo a poluição visual, aumenta-se a eficiência e deixa a tarefa mais agradável, aumentando a satisfação.
Exemplos	Esta tela é do Word 97. Vários objetos são dispostos em uma caixa de diálogo. Os elementos estão dispostos em uma grade e alinhados para ocupar o mínimo espaço.



Usos Conhecidos Configuração de Página do Word 97.

Contra Exemplo Esta imagem é do IBM's Aptiva Communication Center, e demonstra que os desenvolvedores quiseram simplesmente ter os controles na tela, em vez de facilitar aos usuários fazer os ajustes. Não há fluxo na tela, seus olhos simplesmente pulam de um lugar para outro e seu cérebro tenta elicitar algum tipo de ordem.



A.3. Progress (Progresso) by Martijn van Welie

Problema O usuário quer saber o quanto da operação já foi realizado e quanto falta para terminar.

Princípio de Usabilidade Orientação (*Feedback*)

Contexto Tarefas que demoram muito (mais que alguns segundos) e devem estar finalizadas antes que outras tarefas possam iniciar.

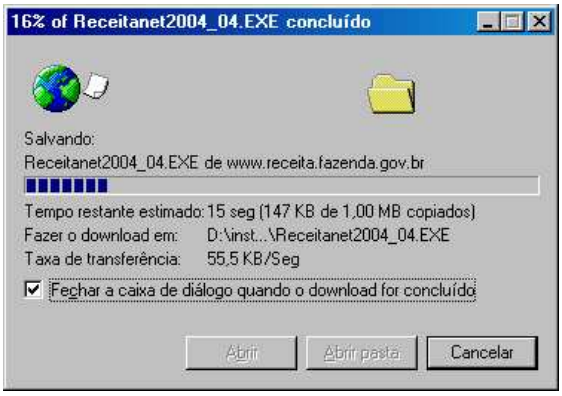
Forças
- O desempenho da operação não pode ser sempre controlado pelo usuário (ou projetista), muitas vezes, porque depende de sistemas externos, que podem falhar, bloquear ou estar com baixo desempenho.
- O usuário não quer esperar e é impaciente.
- O usuário necessita *feedback* claro quanto ao progresso e o tempo restante.
- O usuário pode não estar familiarizado com a complexidade da tarefa.

• Durante a operação, o usuário pode decidir interromper a operação porque esta demorará muito.

Solução Mostre que a aplicação continua rodando e dê uma indicação de seu progresso. Providencie retorno numa taxa que notifique ao usuário de que a operação ainda está ocorrendo, em geral, a cada 2 segundos usando animação. Adicionalmente, providencie uma indicação válida do progresso. Progresso é tipicamente o tempo restante para terminar a operação, o número de unidades processadas ou a porcentagem de trabalho realizado. O progresso pode ser exibido em um elemento visual, como a barra de progresso. A barra de progresso deve ter um rótulo exibindo o progresso relativo ou na unidade em que está sendo medida.

Rationale Providenciando retorno a uma taxa de 1 a 2 segundos, o usuário pode notar que a aplicação está realmente processando e não travou. A indicação de progresso informa quando tempo a aplicação ainda ficará neste estado. Combinando estes dois aspectos, alivia-se a preocupação do usuário. Somente um dos aspectos pode não resoler o problema do usuário. A solução aumenta a satisfação porque o usuário sabe o que está fazendo e quanto tempo precisa esperar.Isto aumenta a sensação de controle. O padrão também evita carga adicional do sistema, evitando novas tentativas do usuário.

Exemplos Fazendo o *download* de um arquivo com o Internet Explorer, o usuário vê um diálogo assim. Ele mostra o progresso em porcentagem assim como quantos *Kilobytes* já foram recebidos. Adicionalmente, o tempo estimado é mostrado e atualizado a cada um ou dois segundos. Uma animação de um documento voando mostra que o *download* não está parado.



Usos Conhecidos Caixa de *download* do Internet Explorer, Atualização de Anti-Virus

A.4. The Shield (O Escudo) by Martijn van Welie

Problema O usuário pode acidentalmente selecionar uma funcionalidade que tenha efeitos irreversíveis.

Princípio de Usabilidade Prevenção de Erros (Segurança)

Contexto Funcionalidades que têm efeitos irreversíveis ou precisem de muito recurso para serem desfeitas. Os efeitos colaterais podem levar a uma situação insegura ou altamente indesejável. O resultado das ações pode ser severo e o usuário pode não ter consciência disto, já que, normalmente, a operação é segura.

Forças
- O usuário deve ser protegido, mas a operação normal não deve ser modificada.
- O usuário busca rapidez ao mesmo tempo em que tenta evitar erros.
- Alguns efeitos colaterais podem ser mais indesejáveis que outros.

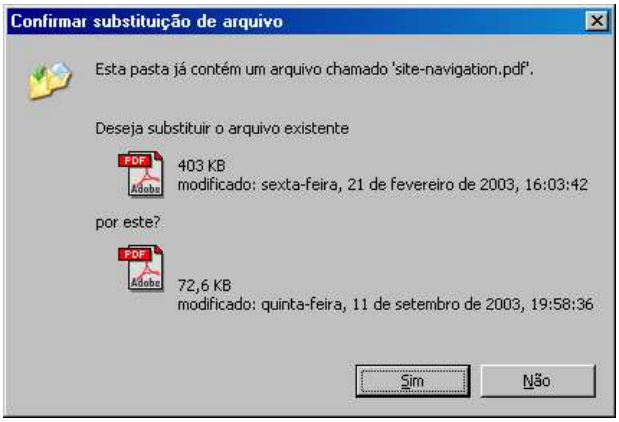
Solução Proteja o usuário inserindo um escudo. Adicione uma camada de proteção extra à funcionalidade para proteger o usuário de cometer enganos. Pede-se ao usuário a

confirmação de seu intento, com a resposta padrão sendo a resposta segura.

Rationale A camada extra exige do usuário dois erros em seqüência em vez de apenas um. A opção segura como padrão diminui a chance de um segundo engano. A solução aumenta a segurança, reduz erros e aumenta a satisfação. Entretanto, requer uma ação a mais do usuário, o que pode levar a uma redução do desempenho.

Exemplo Na câmera digital Mavica da Sony, se você deseja formatar o disquete, a opção selecionada é a de cancelar, sabendo que depois de formatado, não há mais como recuperar a informação.

Contra Exemplo Uma cópia do arquivo já existe em uma pasta. Sobreescrevê-lo resultará em perda da cópia. O padrão deveria ser "Não" para que o usuário apressado tenha que realizar algum esforço para dizer "Sim"., mas o foco está sobre o botão "Sim"...



Usos Conhecidos Microsoft Explorer, Apple Finder

Padrões Relacionados WARNING

A.5. The Preview (A Prévia) by Martijn van Welie

Problema O usuário está procurando por um item em um conjunto pequeno e tenta encontrá-lo pesquisando o conjunto.

Princípio de Usabilidade Compatibilidade (Mapeamento Natural)

Contexto Em muitas aplicações o usuário necessita encontrar um item, por exemplo, um arquivo, uma apresentação, um *clip* de video ou uma imagem, para os quais uma busca visual ou auditiva é mais efetiva, mas o índice do conjunto não é audiovisual (ex. uma etiqueta de texto).

Forças
- Embora o usuário simplesmente queira ver o item real, os recursos necessários para exibí-lo podem não estar disponíveis.
- O usuário vê o nome do índice, mas não é capaz de identificar o item somente pelo

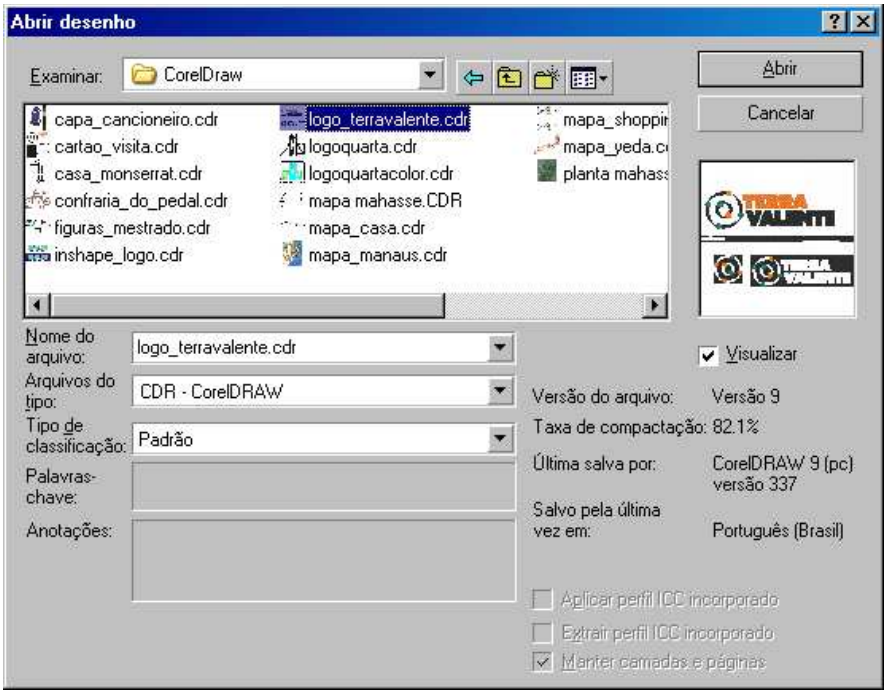
nome.

• O usuário está procurando por um item mas precisa do nome do índice como resultado de busca para outras tarefas.

Solução Permita ao usuário uma prévia do item. Mostre uma prévia representativa de um ou mais itens. A prévia pode utilizar menos recursos como espaço na tela, tempo para exibir ou qualidade do que o original. A prévia deve ser suficientemente boa para poder ser identificada. Se o número de itens no conjunto é pequeno, pode haver prévia de todos os itens ao mesmo tempo. Se o conjunto contém muitos itens ou se a etiqueta de nome é importante para o usuário, exiba uma prévia junto com a etiqueta. A prévia deve estar positionada próxima ao item selecionado para reforçar a ligação entre eles. Quando a seleção muda, a prévia é imediatamente atualizada.

Rationale O tempo de busca diminui porque o usuário pode "ver" ou "ouvir" se o item foi localizado. De outra maneira, o usuário precisaria abrir o item antes de saber se era o procurado. Por usar menos recursos que o original, a prévia é muito mais eficiente e a busca se torna mais efetiva à medida em que a prévia é mais representativa. A solução melhora o desempenho e a satisfação.

Exemplos Este exemplo é da abertura de arquivos do CorelDraw. Uma prévia do arquivo é exibida ao usuário sem que ele necessite abrir o arquivo.



Usos Conhecidos PowerPoint, ACDSee, Acrobat, Adobe Photoshop, Páginas Web

Apendice B
Exemplos de Anti Padrões

B.1. Culto da Carga

Problema	Um projeto está com problemas e é muito visível.
Contexto	Um projeto está próximo de ruir. Muita crítica é dirigida ao projeto, sua equipe e sua liderança. Pessoas no projeto estão sendo desmotivadas por rumores, interferência gerencial e pela contínua demanda por replanejamento, sua moral está ruindo.
Forças	• O projeto tem importância suficiente para não ser descartado; • Mudanças devem ser feitas, pelo menos as externamente visíveis; • Para manter as esperanças de concluir o projeto, a equipe deve ser mantida intacta; • Nova liderança, organização e contexto deve ser encontrado, e rápido!
Solução	Redesenhe o organograma da organização, exibindo o projeto com problemas e a liderança em um contexto novo mais abrangente, talvez diminuindo sua estatura. Se necessário, arranje um "Cordeiro Sacrificial" ou um "Bode Expiatório", mas além do organograma, não mude mais nada.
Contexto Resultante	Administrativamente, o projeto está agora protegido e localizado em uma posição mais segura dentro da burocracia. A ilusão de liderança decisiva é preservada.
Rationale	Sejamos francos: reorganizações são difíceis e raramente acontecem. Freqüentemente, a única mudança é com o organograma, movendo-se nomes, redesenhando caixas e flechas, imaginando que o remanejamento de símbolos realmente organizará a corporação, divisão, departamento, etc. O nome Culto da Carga vem do fenômeno de certas tribos insulares dos mares do sul, que acreditavam que, construindo maquetes de aviões e pistas de pouso, os aviões e toda a ajuda que receberam durante a Segunda Guerra Mundial voltaria, quando as bases dos EUA usaram as ilhas como áreas de estocagem. Em outras palavras, reconstruindo ou simulando os artefatos de uma situação, eles poderiam recriar sua ocorrência.
Moral	O organograma não é a organização. O máximo que você pode esperar é que o organograma reflita a organização real, mas se você se baseia em organogramas para vislumbrar sua situação, você perdeu totalmente a razão de utilizá-los.

B.2. Bode Expiatório

Problema	Um projeto está visivelmente com problemas.
Contexto	Os problemas, até mesmo as falhas, do projeto são altamente visíveis. Pessoas estão bravas e resmungando e algumas até estão abandonando o projeto. O projeto está atrasado ou o orçamento já estourou ou há serias dificuldades técnicas. Ações devem ser tomadas. O projeto assinala cada reunião de metas com um ícone negativo.
Forças	• A gerência está pressionada a retificar ou acabar o projeto. • A solução deve ser rápida, direta e decisiva. • Não há soluções sem dor. • Alguém deve fazer uma chamada para "acordar" a equipe. • A reputação deve ser estabelecida por atitudes duras.
Solução	Escolha alguém para ser punido, através de demissão, mudança de cargo ou remoção

	de suas responsabilidades ou banimento a alguma área sem valor ou importância. Acabar com o projeto é uma medida extrema e usada muito raramente.
Contexto Resultante	Alguns gerentes críticos podem ser temporariamente "amolecidos" com a ação tomada. Dependendo da popularidade do indivíduo escolhido para sacrifício, os membros da equipe podem ficar irados, amedrontados ou agradecidos. É necessário aplicar imediatamente o CultoAPersonalidade, GuruFazTudo ou Pacificador para completar a ilusão de correção.
Rationale	Em qualquer ambiente, a falha tem inúmeras causas. Infelizmente, freqüentemente é necessário concentrar os pecados de todos os envolvidos em uma única pessoa e crer que assim esta levará as imperfeições do projeto embora.

B.3. Processo à Prova de Idiotas

Problema	Uma equipe de software não está cumprindo o prazo.
Forças	• Você está sobre pressão para alterar alguma coisa, porque o projeto está claramente ruíndo. • Um livro promete que a Metodologia X irá diminuir o tempo de análise, remover *bugs* e curar hemorróidas sem cirurgia. • Se você não tivesse todos estes idiotas discutindo, você teria algo já realizado. Talvez se você disser exatamente como fazer o serviço, eles façam direito.
Solução	Desenvolva um processo compreensível, baseado na Metodologia X, mais algumas boas idéias suas. Desenvolva um guia de estilo que liste todos os possíveis erros e advirta-os sobre eles..
O Que Acontece Depois	Você gastará semanas no processo e muitas reuniões argumentando. Em cada reunião, você tenta prevenir todos os possíveis erros que um incompetente ou maligno codificador pode cometer. Depois, você argumenta se um dado erro é sempre um erro, ou às vezes uma boa idéia. Você começa a duvidar da competência de todos na reunião, inclusive de você mesmo. O guia de estilo, quando completo, é mais grosso que a especificação e muito menos útil. A qualidade do código do projeto não aumenta significativamente.
Lições Aprendidas	• Você não pode desenvolver um processo à prova de idiotas. Um bom processo depende de bons desenvolvedores para executá-lo. • Processo não substitui treinamento. • Processo não substitui contratação. • Quanto menor o documento, mais ele é lido. • Não há balas de prata.

Apêndice C
Padrões no Formato Portland

C.1. Decisão do Usuário

Você coletou uma série de Estórias (antigo #1).

Como organizar uma Estória de modo que seja mapeada em uma interface de usuário simples e coerente?

Estórias fornecem muita matéria-prima para moldar a interface do usuário. O problema com Estórias é que elas são organizadas cronologicamente, enquanto interfaces são organizadas espacialmente. Você deve quebrar a Estória em partes, sendo que algumas delas estarão espacialmente mais próximas que outras. A Estória deve ser transposta do domínio temporal para o domínio espacial.

Uma boa interface de usuário possui um senso de plotagem e fluxo. Naturalmente, você segue algum caminho através das partes da interface, primeiramente utilizando uma janela, depois mudando seu foco para outra janela. Interfaces que forçam pular para frente e para trás, entre estilos diferentes de interação (edição de texto para preenchimento de formulário para tabela para manipulação direta...) violam este senso de fluxo.

Para evitar o "choque de interface", é necessário dispor partes da Estória relacionadas cronologicamente em locais próximos espacialmente na interface. Ao mesmo tempo, deve-se utilizar a melhor técnica para apresentação e manipulação para cada parte da Estória. Inicialmente, devemos identificar os "átomos" da interface de maneira que possamos encontrar os "pacotes"

Portanto:

Faça uma lista de toda decisão que o usuário deva tomar durante as Estórias. Para cada decisão, escreva a informação que o usuário precisa para decidir corretamente.

Agrupe as decisões em Tarefas (C.2).

C.2. Tarefa

Você tem uma lista de Decisões de Usuário (C.1).

Como agrupar as decisões em tarefas coerentes?

Um dos marcos de uma boa interface é que qualquer um sente que pode trabalhar do jeito que deseja, que o sistema não impõe nenhum método particular de resolver o problema. De fato, este sentimento é o objetivo principal do meu envolvimento com padrões - dar ao usuário um senso de domínio sobre o ambiente.

É necessário projetar uma interface que atenda estas necessidades. O usuário não pode se sentir tolhido por prerrogativas impostas pelo sistema. A gerência (quem normalmente paga pelo trabalho) deve perceber que nenhum usuário se afasta muito dos caminhos produtivos.

Usuários com maior autonomia não estão sujeitos a um escrutínio tão intenso. Mesmo assim, devemos a estes usuário uma estrutura. Uma interface sem estrutura, não possui ponto de vista, nem motivação. A interface rapidamente se converte em um exercício de projeto de interface para o usuário, através de preferências, macros ou até mesmo uma linguagem de programação embutida.

Aqui há um dilema. Devemos estruturar a interface, porque sem estrutura não há motivação. Nós não devemos estruturar a interface, porque estruturar afasta a experiência do usuário de estar no controle da máquina ao invés de estar sendo controlado.

Você pode não estar confortável com este tipo de dilema. Como é possível saber como qualquer usuário possível irá utilizar o sistema? Como se pode ter certeza de estar tomando as decisões corretas, sem bloquear importantes vias de exploração pelo usuário?

Duas respostas. 1) Os padrões dão uma chance decente de estar fazendo o certo para a maioria dos usuários a maior parte do tempo. 2) É claro que você cometerá erros. Não é motivo para não entregar o sistema. Há sempre os "*power users*" que irão extrapolar qualquer facilidade que você providencie.

Portanto:

Escreva cada Decisão do Usuário e sua informação agregada em um cartão indexado. Ponha cartões com a mesma informação encostados uns aos outros. Ponha pacotes de cartões com informação similares próximos. Cada pacote de cartões é uma tarefa. Nomeie-a.

Suporte cada Tarefa com uma Janela de Tarefa (C.3). Utilize um assistente quando a disposição espacial começar a ficar confusa.

C.3. Janela de Tarefa

Você tem uma Tarefa (2).

Quanto do sistema você mostra em uma simples janela?

Projetos de interface tendem a cair em um de dois campos: o "tudo em uma janela" e o "tudo em janelas separadas". Nenhum destes dois exremos servem para o usuário de sistemas de computador atualmente.

O "tudo em janelas separadas" é exemplificado pelo *design* dos primeiros Macintosh. O resultado é que a maioria dos softwares deste tipo é de difícil utilização, porque o usuário gasta muito tempo gerenciando as janelas. A necessidade de um menu de janelas é um sintoma deste problema.

"Tudo em uma janela" vai longe em outro quesito. Dispondo toda a funcionalidade de um sistema grande em uma única janela resulta em um vasto mar de escolhas. O sistema não ajuda o usuário a estruturar seu tempo, nem apresenta um fluxo através do sistema.

Encontrar um "meio feliz" é crítico à operação da interface (e à viabilidade do resto dos padrões). Como afirmar se já existe informação suficiente em uma janela?

Peça ao usuário que divida seu trabalho em tarefas identificáveis. Crie uma janela para cada tarefa. Nomeie a Janela de Tarefa.

As ações disponíveis serão Ações Visíveis (?). Divida a janela em alguns Painéis (?)

Apêndice D
Padrões na Forma Alexandrina

D.1. Cozinha de Fazenda

Problema A cozinha isolada, separada da família e considerada como uma eficiente mas desagradável fábrica de comida é um legado dos dias de servidão e dos dias mais recentes, quando as mulheres assumiram o papel servil na família.

Solução Faça a cozinha maior que o usual, grande o sificiente para incluir o espaço "sala de estar" e posicione-a próxima ao lugare mais utilizados da casa, não longe como uma cozinha ordinária. Faça-a grande o suficiente para conter uma grande mesa e cadeiras, algumas duras e outras macias, com o fogão, pia e afins nos limites da cozinha; faça isto num ambiente luminoso e confortável.

D.2. Ponto de Ônibus

Problema Pontos de ônibus devem ser de fácil identificação e agradáveis, com atividade ao redor suficiente para deixar as pessoas confortáveis e seguras.

Solução Construa pontos de ônibus de modo que se formem pequenos centros de vida pública. Construa-os como parte de passagens entre vizinhanças, comunidades de trabalho, partes da cidade. Localize-os de modo que operem em conjunto com outras atividades, como bares, mercearias, jardins, banheiros públicos...

D.3. Vista Zen

Problema A vista Zen arquétipa ocorre em uma famosa casa japonesa, que dá a este padrão o nome.

Solução Se há uma bela paisagem, não a esconda, construindo janelas amplas que explicitem a paisagem. Disponha as janelas em locais de transição: caminhos, salões, entradas, escadas, entre salas.

Se a janela for corretamente colocada, as pessoas verão um pouco da paisagem distante assim que se aproximarem da janela ou passarem pela frente: mas a vista nunca é visível de lugares onde as pessoas ficam.

Apêndice E
Padrões GoF

Abaixo, os 23 padrões de projeto catalogados no livro Padrões de Projeto [GoF,1995]

Padrão	Intenção
Abstract Factory	Fornece uma interface para criação de famílias de objetos relacionados ou dependentes sem especificar suas classes concretas.
Builder	Separa a construção de um objeto complexo da sua representação, de modo que o mesmo processo de construção possa criar diferentes representações.
Factory Method	Define uma interface para criar um objeto, mas deixa as subclasses decidirem qual classe a ser instanciada. O Factory Method permite a uma classe postergar (*defer*) a instanciação às subclasses.
Prototype	Especifica os tipos de objetos a serem criados usando uma instância prototípica e cria novos objetos copiando este protótipo.
Singleton	Garante que uma classe tenha somente uma instância e fornece um ponto global de acesso a ela.
Adapter	Converte a interface de uma classe em outra interface esperada pelos clientes. O Adapter permite que certas classes trabalhem em conjunto, pois de outra forma seria impossível por causa de suas interfaces incompatíveis.
Bridge	Separa uma abstração da sua implementação, de modo que as duas possam variar independentemente.
Composite	Compõe objetos em estrutura de árvores para representar hierarquias do tipo partes-todo. O composite permite que os clientes tratem objetos individuais e composições de objetos de maneira uniforme.
Decorator	Atribui responsabilidades adicionais a um objeto dinamicamente. Os decorators fornecem uma alternativa flexível a subclasses para extensão de sua funcionalidade.
Façade	Fornece uma interface unificada para um conjunto de interfaces em um subsistema. O Façade define uma interface de nível mais alto que torna o subsistema mais fácil de usar.
Flyweight	Usa compartilhamento para suportar grandes quantidades de objetos, de granularidade fina, de maneira eficiente.
Proxy	Fornece um objeto representante (*surrogate*), ou um marcador de outro objeto, para controlar o acesso ao mesmo.
Chain of Responsibility	Evita o acoplamento do remetente de uma solicitação ao seu destinatário, dando a mais de um objeto a chance de tratar a solicitação. Encadeia os objetos receptores e passa a solicitação ao longo da cadeia até que um objeto a trate.
Command	Encapsula uma solicitação como um objeto, desta forma permitindo que você parametrize clientes com diferentes solicitações, enfileire ou registre (*log*) solicitações e suporte operações que podem ser desfeitas.
Interpreter	Dada uma linguagem, define uma representação para sua gramática juntamente com um interpretador que usa a representação para interpretar sentenças desta linguagem.
Iterator	Fornece uma maneira de acessar seqüencialmente os elementos de um objeto agregado sem expor sua representação subjacente.
Mediator	Define um objeto que encapsula como um conjunto de objetos interage. O Mediator promove o acoplamento fraco ao evitar que os objetos se refiram explicitamente uns aos outros, permitindo que você varie suas interações independentemente.
Memento	Sem violar a encapsulação, captura e externaliza um estado interno de um objeto, de modo que o mesmo possa posteriormente ser restaurado para este estado.
Observer	Define uma dependência um-para-muitos entre objetos, de modo que, quando um objeto muda de estado, todos os seus dependentes são automaticamente notificados e atualizados.

State	Permite que um objeto altere seu comportamento quando seu estado interno muda. O objeto parecerá ter mudado sua classe.
Strategy	Define uma família de algoritmos, encapsular cada um deles e fazê-los intercambiáveis. O Strategy permite que o algoritmo varie independentemente dos clientes que o utilizam.
Template Method	Define o esqueleto de um algoritmo em uma operação, postergando a definição de alguns passos para subclasses. O Template Method permite que as subclasses redefinam certos passos de um algoritmo sem mudar sua estrutura.
Visitor	Representa uma operação a ser executada sobre os elementos da estrutura de um objeto. O Visitor permite que você defina uma nova operação sem mudar as classes dos elementos sobre os quais opera.