

## O Desenvolvedor-Designer e o Design da Interação *"Devigners" and interaction design*

Perry, Gabriela Trindade; Mestre; Universidade do Vale dos Sinos  
gabrielperry@hotmail.com

### Resumo

O personagem central deste artigo é o desenvolvedor-designer, o "*devigner*", uma pessoa cujo papel é desenvolver a parte computacional da aplicação e o design da interação. O ponto central é que mesmo que o "*devigner*" tenha educação formal e experiência em design da interação, o fato de ser responsável pelo lado computacional da aplicação é suficiente para declará-lo incapaz de conduzir atividades de design de interação. A razão é que, para desenvolver o lado computacional de uma aplicação, é necessário usar um *framework* mental que focaliza as necessidades do sistema, o que acaba por deixar as tarefas que o sistema apóia e as necessidades do usuário em segundo plano. Seguindo esta linha de raciocínio, recomenda-se que a os papéis de desenvolvedor e de designer não sejam acumulados pela mesma pessoa.

**Palavras Chave:** devigners; design de interação; equipe de desenvolvimento.

### Abstract

*The central character of this paper is the developer-designer (the devigner), a person who has responsibility both for the implementation and interaction design. The point is that even if these "devigners" have formal education and experience in interaction design, being responsible for the computational side of the application is enough to declare them unable of designing interaction. The reason is that, in order to develop the computational side of an application, you must use a mental framework that focuses the system's needs, not the task's or user's. Following this line of reasoning, it is recommended that projects should have an interaction designer.*

**Keywords:** devigners; interaction design; development team.

## Introdução

O personagem central deste artigo é o desenvolvedor-designer, o “*devigner*”, uma pessoa cujo papel é desenvolver a parte computacional da aplicação e o design da interação. O ponto central é que mesmo que o “*devigner*” tenha educação formal e experiência em design da interação, o fato de ser responsável pelo lado computacional da aplicação é suficiente para declará-lo incapaz de conduzir atividades de design de interação. A razão é que, para desenvolver o lado computacional de uma aplicação, é necessário usar um *framework* mental que focaliza as necessidades do sistema, o que acaba por deixar as tarefas que o sistema apóia e as necessidades do usuário em segundo plano. Seguindo esta linha de raciocínio, recomenda-se que a os papéis de desenvolvedor e de designer não sejam acumulados pela mesma pessoa.

Contudo, antes de expor a argumentação, será feita uma breve ilustração do cenário de desenvolvimento de software e de design de interação. Em seguida, será apresentado o que se chama de “*devigner*”.

## O Processo de Desenvolvimento de Software

Até o início dos anos 80, quando se popularizaram os estudos em interface homem-computador, os profissionais das ciências da computação eram os únicos envolvidos nos processos de desenvolvimento de software (HENDRICK, 2001). Um dos motivos é que, antes desta data, havia poucos exemplos de aplicações que possuíam interfaces visuais. As primeiras tentativas de organizar o ciclo de vida destes produtos são anteriores a este período, como o modelo Waterfall (ROYCE, 1970). Já o modelo Espiral (BOEHM, 1988), que inspira as metodologias baseadas em prototipagem rápida de hoje em dia (por exemplo: XP e RUP), é de 1988.

O aumento das capacidades dos computadores pessoais acabou por viabilizar a produção de aplicativos cada vez mais sofisticados, motivando uma explosão de pesquisas sobre interação homem-computador (doravante IHC), com trabalhos em áreas como: manipulação direta (SCHNEIDERMAN, 1980) utilização de metáforas (KAY, 1990) e modelagem do desempenho do usuário (CARD *et al.* 1980), por exemplo. Desta forma, mobilizados por uma grande demanda por softwares que apoiassem tarefas de forma efetiva e sem causar constrangimentos ao usuário, um novo grupo de profissionais começou a fazer parte do cenário do desenvolvimento: os designers de interação. Dele fazem parte os responsáveis por inserir e implementar práticas de IHC durante o ciclo de vida.

Nas seções seguintes, serão apresentados alguns conceitos-chave utilizados neste artigo: design de interação, designers-desenvolvedores e desenvolvedores-designers.

## Design de Interação

Para apresentar o conceito de design de interação serão utilizadas definições apresentadas por dois autores da área de IHC, Dix *et all.* (1988) e Hix e Hartson (1993). No primeiro, encontra-se a seguinte definição de interação:

Por interação nos referimos a qualquer comunicação entre o usuário e o computador, seja direta ou indireta. Interação direta envolve um diálogo com feedback e controle

durante a realização de uma tarefa. Interação indireta pode envolver processamento de fundo ou em lotes. O importante é que o usuário está interagindo com o computador com algum objetivo.

O designer de interação é responsável por desenvolver o conteúdo, comportamento e aparência do design de interação (HIX e HARTSON, 1993). Pessoas neste papel são diretamente responsáveis por assegurar usabilidade, incluindo desempenho e satisfação do usuário. Eles estão preocupados com elementos críticos de design como funcionalidade, seqüenciamento, conteúdo e acesso à informação, bem como detalhes como aparência de menus, formatação de formulários, e como assegurar consistência através da interface. Uma grande parte do trabalho designer está relacionada com definir indicadores de usabilidades mensuráveis, avaliar designs de interação com usuários e fazer redesigns baseados na avaliação dos usuários.

As atividades do designer de interação transcendem a aparência da interface e definição de guias de estilo, pois é dele a responsabilidade de definir como a aplicação se comunica com o usuário (HIX e HARTSON, 1993). Isso implica não apenas escolher componentes de interação de acordo com cada ocasião - por exemplo: menus, listas, campos de texto, alertas - ou o tipo da interface - por exemplo: se é modal ou não, se tem janelas ou “tabs”<sup>1</sup> - mas definir a forma como estes componentes orientam a ação dos usuários através da aplicação. Um designer de interação projeta sempre pensando na tarefa e no usuário, e seu modelo mental não focaliza as necessidades do sistema (NORMAN, 1988). Com esta exposição, procura-se evidenciar que a atividade de design de interação está essencialmente ligada à compreender o usuário realizando uma tarefa, e que demanda uma natureza de conhecimento diferente daquela do desenvolvedor. Também se deseja mostrar que esta atividade transcende o caráter estético/simbólico, que apenas se preocupa com a aparência das interfaces.

Apesar da importância dos profissionais de design da interação não ser questionada, sua inclusão em equipes de desenvolvimento pequenas ainda não é a prática mais comum (HOULGHIN, 2010), porque (1) os prazos para entrega dos projetos são curtos e (2) o processo é caro, complexo e sujeito à falhas potencialmente graves, muitas equipes não alocam períodos para o projeto da interação. Acredita-se que este cenário tenha motivado o surgimento do papel do *devigner* dentro destas equipes.

### *Devigner: o Desenvolvedor-Designer*

Conforme visto na seção 2, os desenvolvedores estão há mais tempo envolvidos com a produção de software que os designers de interação. Também se pode perceber, através da análise da evolução dos métodos de controle do ciclo de vida (como os citados na seção 2), que o grau de especialização foi aumentando com o passar do tempo, criando novas atividades, tais como: “analistas”, responsáveis pela arquitetura da aplicação; “programadores”, responsáveis pela implementação da arquitetura em uma determinada linguagem e “testadores”, responsáveis por escrever testes automatizados para cada uma das partes da aplicação. Normalmente estes três papéis são bem definidos dentro das equipes,

---

<sup>1</sup> Para exemplos e definição deste controle, ver van Welie (2008).

existindo uma clara hierarquia de responsabilidade entre eles. Em contrapartida, o papel do designer de interação não é tão bem definido – em uma das conseqüências é que ele seja acumulado por um destes profissionais (por experiência própria: normalmente pelo programador). Chama-se este profissional é chamado de *devigner*, uma mistura de desenvolvedor e designer. Este termo foi criado por membros de comunidades de desenvolvedores de RIA - *Rich Internet Applications* - que detectaram a falta de designers visuais capacitados a se integrar ao fluxo de trabalho de projeto de interfaces (STEWART, 2006).

Na próxima seção serão apresentados os argumentos que levam à conclusão da necessidade de haver um responsável pelo design da interação que não esteja também responsável por tarefas de caráter computacional de um projeto.

### Porque um desenvolvedor não deve acumular a tarefa de design de interação em um mesmo projeto

Em princípio, cita-se Theodor Nelson (1990), que sustenta que desenvolvedores não estão capacitados para fazer design de interação. Segundo ele:

Aprender a programar tem tanto a ver com design quanto aprender a digitar tem a ver com escrever poesias. O design da interatividade é raramente ensinado em escolas de programação. O que precisamos em software é o que as pessoas aprendem em escolas de cinema, ao menos até onde isso possa ser ensinado.

Em relação à afirmação de Nelson de que “o design da interatividade é raramente ensinado em escolas de programação”, cita-se que a ACM e IEEE recomendaram que o ensino de IHC fosse introduzido nos currículos dos cursos superiores de computação já no final dos anos oitenta (DENNING et al, 1988). Passados mais de 20 anos desta recomendação, se poderia esperar que esta área fosse mais bem compreendida dentro das equipes de desenvolvimento. Ressalta-se que, neste artigo, não se pretende – de forma alguma!- afirmar que um profissional da área da computação não esteja preparado para desempenhar a atividade de designer de interação. O que se pretende é concluir que não se deve acumular este papel com o de desenvolvedor em um mesmo projeto.

Seguindo a linha de que as atividades de design de interação e desenvolvimento computacional devem ser feitas por diferentes profissionais, Marion (1999) faz uma analogia com arquitetos e engenheiros. Diz o autor que, se as casas fossem feitas apenas por engenheiros, elas seriam certamente mais fáceis de construir, mas não necessariamente melhor para se viver. Como o designer, o arquiteto, além de ser preparado para perceber o que funciona ou não, tem um papel de advogado do cliente (no caso do designer, o usuário).

Neste cenário, a noção de “modelo conceitual” se constitui em um metáfora que pode ser utilizada para compreender o motivo desta incompatibilidade entre o fazer do desenvolvedor e do designer (NORMAN, 1988). Neste “modelo conceitual” estão representados os modelos que o designer e o usuário têm do sistema, realizados na “imagem do sistema”. Se os modelos destes dois sujeitos forem diferentes, o resultado é que, para o

usuário, o sistema parecerá inconsistente e difícil de usar. Norman propõe a metodologia Design Centrado no Usuário como forma de evitar esta situação. Seguindo esta linha, assume-se que o designer (de interação, no caso deste artigo) é o profissional responsável por projetar o sistema considerando as necessidades do usuário. No entanto, para descrever de forma mais precisa o cenário do desenvolvimento, é preciso considerar os indivíduos responsáveis pela implementação do sistema, chamados aqui, de forma genérica, de desenvolvedores. A forma como estas relações se dão, de acordo com a visão deste artigo, está ilustrada na figura 1.

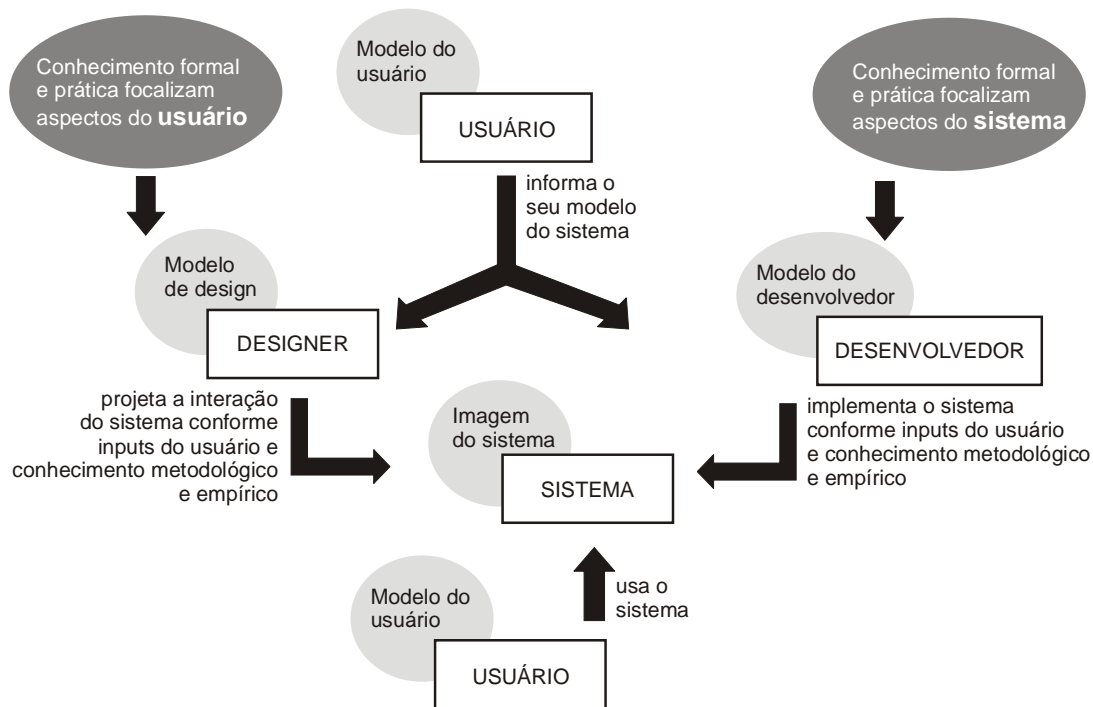


Figura 1: Relações simplificadas entre usuário, designer, desenvolvedor e sistema em um cenário de desenvolvimento de software, adaptadas de Norman (1988).

Na figura 1, percebe-se que o usuário informa tanto o desenvolvedor quanto o designer sobre seu modelo do sistema. De fato, ambos profissionais são capacitados, durante o curso de graduação, a ouvi-lo e extrair destas entrevistas um conjunto de requisitos para a aplicação.

As diferenças estão na forma como estes encontros com o usuário acontecem, na forma que esta informação é utilizada e no resultado deste tratamento. Enquanto o designer procura ouvir o usuário e compreender as necessidades que a aplicação deve satisfazer (através de métodos como análise da tarefa e entrevistas, por exemplo), o desenvolvedor procura ouvir o usuário para extrair o conjunto de requisitos da aplicação, o que permite que seja executado o ponto de entrada em sua metodologia de desenvolvimento: a geração de casos de uso. O designer e o desenvolvedor, à medida que projetam, são informados por conhecimentos formais e por saberes empíricos, que podem misturar experiências práticas com orientações individuais. A diferença fundamental - e este é o argumento central deste

artigo - é que o modelo do designer, por princípio, é centrado no usuário, enquanto o do desenvolvedor é centrado no sistema. Assim, enquanto o designer projeta, ele tem em mente o usuário, enquanto o desenvolvedor tem em mente o sistema.

Por este motivo, mesmo que um desenvolvedor acumule a função de designer de interação, ele estará utilizando, em função das necessidades das atividades relacionadas à implementação, um modelo centrado no sistema. Mesmo que este desenvolvedor seja uma pessoa educada formalmente e com saber empírico consolidado em design de interação, ele não conseguirá “trocar de modelo” a todo tempo. À medida que o desenvolvimento da aplicação avança, em alguns momentos as necessidades do sistema consumirão suas atenções, seja porque tais necessidades são críticas para o desenrolar do ciclo, seja por serem difíceis de satisfazer. Nestes momentos, o desenvolvedor não consegue se portar como se tivesse “dupla personalidade”, ora utilizando um “modelo de designer”, ora um “modelo de desenvolvedor”. Após estes momentos passarem, e as questões terem sido resolvidas, as necessidades do usuário foram deixadas em segundo plano, e retomá-las neste ponto pode ocasionar um nova entrada em ciclos de turbulência no desenvolvimento. Uma outra possibilidade é que estes momentos de atenção extremamente focada nas necessidades do sistema se sucedam durante uma boa parte do ciclo, sem períodos de calmarias – quando, hipoteticamente, o desenvolvedor poderia usar seu “modelo de designer”.

Para ilustrar estas idéias com exemplos, a seção seguinte traz (1) duas mensagens (posts) feitos em uma lista de discussão sobre desenvolvimento de *Rich Internet Applications*; (2) o caminho percorrido pela autora deste artigo (atuando como *devigner*) para implementar um caso de uso em uma aplicação educacional e (3) mensagens da mesma lista de discussão nas quais os autores afirmam considerar o design como um elo importante na cadeia de desenvolvimento.

## Devigners em ação

Esta seção mostra mensagens retiradas de uma lista de discussão sobre *Rich Internet Applications*. Nelas, os autores têm questões que pertencem ao domínio do design de interação, mas tratam-nas como uma questão do sistema.

Seguinte galera, preciso criar uma aplicação gigante em XXX... Estou a milhões de anos programando em XXX e agora estou engatinhando em XXX. Alguém poderia me dar uma mão?

Minha idéia é fazer um menu principal que crie dinamicamente painéis com um moduleloader dentro, que hajam como se fossem janelas com minimizar, maximizar, mover, dimensionar. Precisaria também que quando fosse clicado no menu para chamar essas janelas, verificasse se elas já existem.

Existe alguma coisa pronta pra isso? Essa é a melhor forma? Como vocês fariam pra fazer uma aplicação com uns 50 cadastros e uns 800 relatórios?

Se possível mandem exemplos... Fontes... que não consegui nem criar dinamicamente os painéis.

Quadro 1: Pergunta feita à lista flexdev. Em

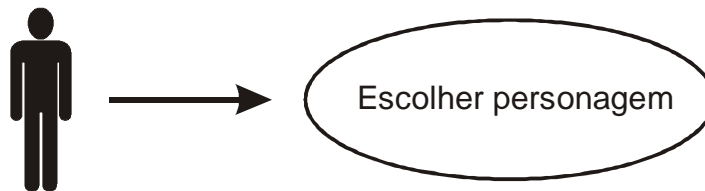
[http://groups.google.com.br/group/flexdev/browse\\_thread/thread/134475aec2010547/b8aea1bff8865830](http://groups.google.com.br/group/flexdev/browse_thread/thread/134475aec2010547/b8aea1bff8865830)

O usuário desta lista demonstra estar utilizando um modelo de desenvolvedor ao fazer estes questionamentos. Ele se preocupa com a implementação, descrevendo os componentes que está pensando em utilizar, informa o tamanho do sistema (esta é uma variável importante, que pode ser um fator de decisão entre diferentes abordagens à implementação), pergunta qual é a melhor forma, pede exemplos... Em listas de discussão este tipo de mensagem não chega a ser incomum. Um exemplo é a mensagem mostrada no quadro 2.

Agora... Eu sou meio radical e me conscientizei que edição de registros diretamente em datagrid não é uma boa prática... Como fazer a validação de cada campo? “if campo1 valida isso”... “else if campo2 valida aquilo”... Fica meio feio não fica?

Quadro 2: Pergunta feita à lista flexdev.

Na mensagem do quadro 2, a argumentação que apóia a conclusão “fazer edição de registro em datagrid não é boa prática” é de natureza computacional, ou seja, não é bom porque o código não fica “bonito”. O próximo exemplo, que mostra o caminho percorrido pela autora deste artigo para implementar um caso de uso (figura 2) é de natureza parecida. Neste exemplo se pode ver como as necessidades do usuário e do sistema se sobrepõe.



### Cenário:

O usuário deve escolher um personagem, representado por um desenho. Este personagem terá um nome, uma atividade (por exemplo: “prefeito”) e a uma fala.

Quando a aplicação for lançada, haverá cerca de 100 Desenhos.

Para facilitar a tarefa de escolha, deve ser disponibilizado um filtro.

Figura 2. Estudo de caso e cenário associado.

Como o caso foi vivenciado pela autora deste artigo, pode-se mostrar a pergunta que guiou a resolução do problema: “como mostrar vários desenhos ocupando o máximo espaço da tela?”. A fim de ilustrar a seqüência de ações que resultaram na tela final, será simulado um diálogo entre a designer e a desenvolvedora – ainda que ambas sejam a mesma pessoa.

Designer: Como mostrar vários desenhos ocupando o máximo espaço da tela? Colocando-os em linhas e colunas.

Desenvolvedora: Eles têm proporções semelhantes?

Designer: Sim.

Desenvolvedora: Menos mal. Bem... Tem um componente que faz isso. Se chama TileList. Você só tem que “dar” um ArrayCollection para ele e...

Designer: Este componente consegue redefinir a quantidade de linhas e colunas se o usuário redimensionar a tela?

Desenvolvedora: Sim.

Designer: Ótimo. Vamos ver como fica.

Desenvolvedora: [estuda exemplos de aplicação, cria testes e descobre que o redimensionamento não é dinâmico como havia afirmado]. Aqui! Está funcionando.

Designer: Muito bom mesmo. Mas tem muitos desenhos... As pessoas vão escolher sempre os primeiros, que são os únicos que ela enxerga. Será que dá pra fazer uma espécie de filtro?

Desenvolvedora: Claro que sim. Tem um recurso chamado “filterFunction”. É só criar uma função que retorna um booleano e...

Designer: Ótimo. Você pode ver se funciona mesmo?

Desenvolvedora: Claro! [o que eu tenho que “passar” para essa função? E o que eu faço com esse booleano? Ah!, ele esconde os elementos do array que não satisfazem a condição determinada pela função... Entendi... Então deixa eu colocar uns botões aqui para “ligar” e “desligar” esses atributos... Ok, funcionando (figura 3)].

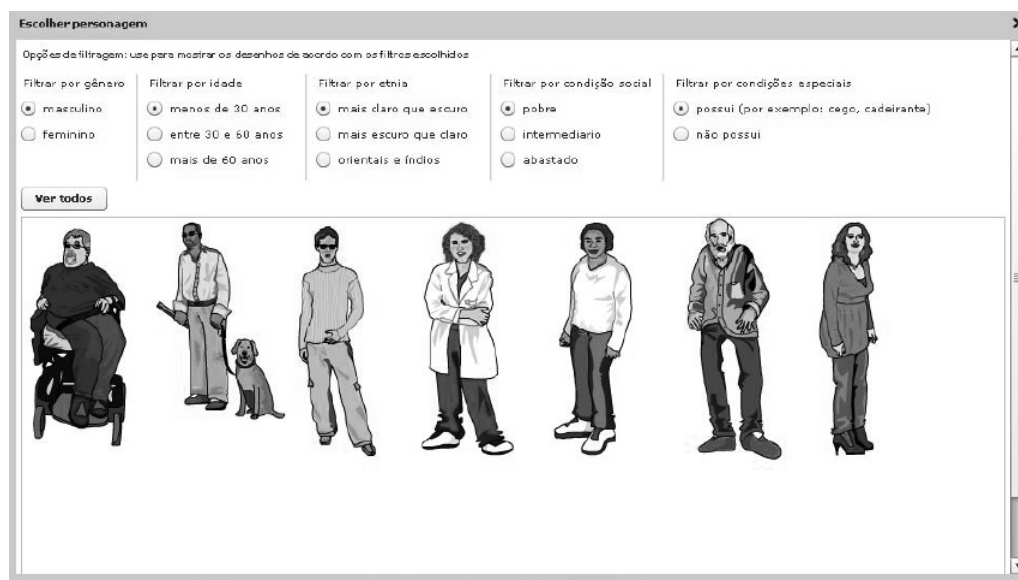


Figura 3: Primeira solução.

Designer: Pois é... Mas e se a pessoa quiser escolher mais de um atributo? Por exemplo: ela quer ver todos os desenhos que representam mulheres. Esta interface não permite, pois como restringe às mulheres com menos de 30 anos, de pele clara, pobres e sem necessidades especiais.

Desenvolvedora: É mesmo...

Designer: Acho que você escolheu mal o componente. Botões de radio não permitem múltiplas escolhas.

Desenvolvedora: Nem percebi. Estava tão preocupada em fazer este filtro funcionar... Mas isso é simples... É só trocar por checkboxes (figura 4).

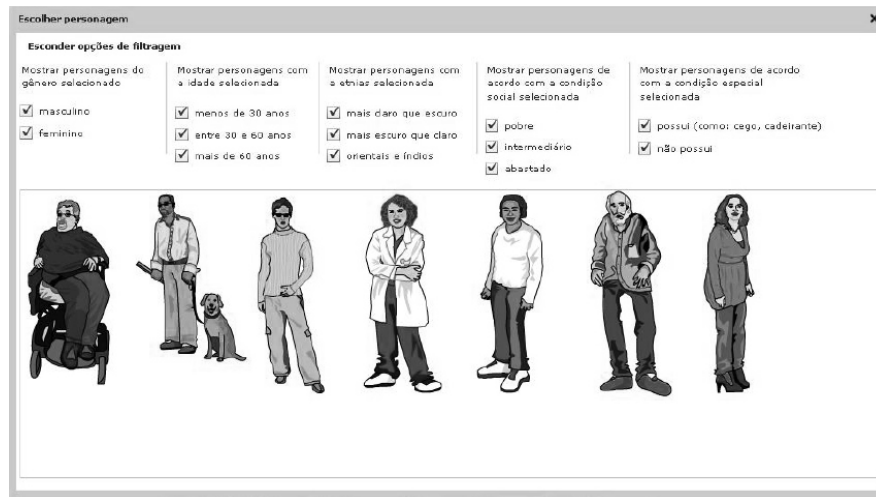


Figura 4: Segunda solução (após perceber erro de design de interação).

Designer: Certo... A questão passou a ser como a seleção dos *checkboxes* afeta a visualização da interface. A primeira opção seria fazer os itens estarem todos marcados no início, e a ação de desmarcar removeria os desenhos que não se encaixam na seleção. A segunda seria iniciar com os itens desmarcados e, à medida que fossem marcados, esconderiam os que não deveriam ser mostrados. Esta escolha leva em consideração se é mais natural para o usuário escolher o que ele quer ver ou o que ele não quer ver (figura 5).

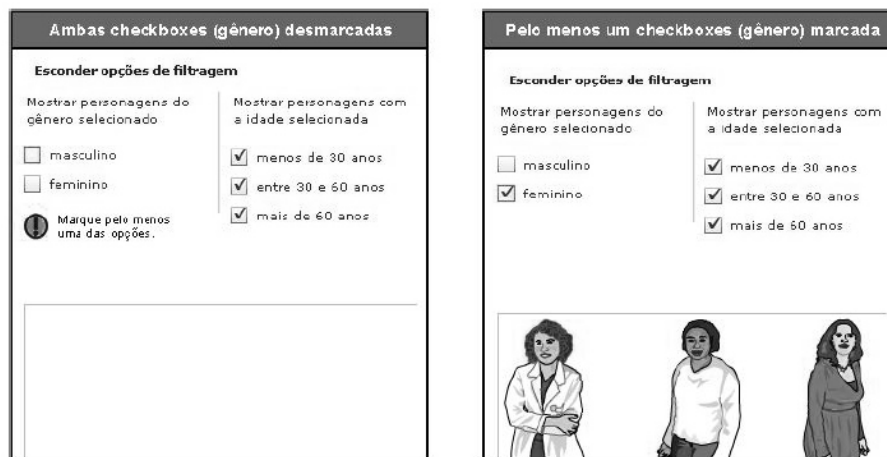


Figura 5: Diferenças de comportamento: escolher “o que quer ver” ou “o que não quer ver”?

Designer: Este problema pode ser resolvido aplicando uma das heurísticas propostas por Norman (1988): “projetar para o erro”. Caso o usuário desmarque todas as opções de qualquer um dos filtros, não haveria desenhos para serem mostrados (figura 5, à esquerda). Nesse caso ainda há como viabilizar a solução, colocando um aviso, como ilustrado na figura 5 à esquerda. A segunda solução seria impedir que todos os itens fossem desmarcados. Porém esta solução interfere na liberdade de ação do usuário e tem potencial de provocar confusão (por exemplo: “porque essa alternativa não pode ser desmarcada e as outras podem?”).

Felizmente o desfecho foi favorável neste caso, ou seja, as especificações de design não deixaram de ser atendidas por falta de prazo ou conhecimento. E, para finalizar este capítulo com uma mensagem positiva, são mostradas no quadro 3 duas mensagens (retiradas da mesma lista de discussão dos quadros 1 e 2) onde a importância do design dentro do ciclo de desenvolvimento é confirmada. Ressalta-se que os autores de tais mensagens são (muito provavelmente) desenvolvedores, pois a lista é voltada à programação.

Talvez, para você que deseja fazer um ERP no estilo MDI, seja a hora de pensar melhor em aparências, usabilidades e design... Novos conceitos estão sendo implantados... Veja o Windows Vista mesmo... E melhor o: Office 200, que não tem menus... Já perceberam a mudança que houve?

Vou repetir o que alguns já disseram e adicionar outros tópicos: - Design, usabilidade e acessibilidade.

Quadro 3: conselho: pensar em design.

Acredita-se que estas mensagens reflitam uma preocupação real com o design, e que está provocando um aumento na demanda por profissionais de design.

## Conclusão

A conclusão que se faz a partir destes relatos, é que, no caso do primeiro grupo de mensagens da lista de discussão, há uma subordinação do design de interação à programação. Provavelmente esta situação se deu porque, na maioria das vezes, pessoas que buscam auxílio em listas de discussão estão aprendendo (1) a trabalhar em um ambiente de desenvolvimento ou (2) uma linguagem de programação ou (3) uma biblioteca de código. Assumindo esta hipótese como verdadeira, é plausível pensar que as preocupações com design fiquem em segundo plano. Este quase foi o caso com a interface implementada pela autora do artigo: as dificuldades com a programação fizeram com que as necessidades do sistema muitas vezes se sobrepusessem às do usuário. Já no segundo grupo de mensagens da lista de discussão se pode ver que os autores afirmam a importância do design dentro do ciclo de desenvolvimento.

Conclui-se que é recomendável haver uma pessoa responsável apenas pela especificação do design de interação. O motivo é que, para realizar esta tarefa, é preciso utilizar um modelo de pensamento que priorize as necessidades e características do usuário e da tarefa. Se, como visto nos exemplos da seção anterior, o profissional acumula esta

atividade com algum aspecto da implementação do sistema, o resultado é que o modelo de raciocínio utilizado para tomar decisões que prevalece é o do desenvolvedor.

Isto pode se dar por vários motivos, entre os quais cita-se: as necessidades de implementação são mais urgentes por representarem um empecilho direto à geração de protótipos funcionais (um requisito de metodologias ágeis, como XP). Também se pode especular que os desenvolvedores, por causa das características de sua formação sintam-se mais à vontade com atividades que possam ser projetadas, analisadas, construídas e testadas, ou seja, que tenham um caráter mais objetivo. As atividades relacionadas ao design de interação, por sua vez, mesmo que possam ser modeladas e testadas, pressupõe um conhecimento do usuário que está impregnado de aspectos subjetivos. Além disso, não se testa todas as possibilidades de interações: deve-se decidir entre algumas delas com base nos conhecimentos formais e empíricos. Isto pode ser uma característica da atividade que pode ser pouco atraente para os desenvolvedores.

De toda forma, parece não haver impedimentos para o desenvolvedor que deseje executar as especificações de interação. Apenas recomenda-se que ele não o faça se estiver responsável por atividades de implementação em um determinado projeto. Nesse caso, a colaboração entre os profissionais pode-se dar através de projeção em duplas, para que os dois portadores de modelos tão importantes para o sucesso de uma aplicação – os modelos de desenvolvedor e de designer – possam contribuir da melhor forma possível, sem negligenciar um aspecto em detrimento de outro.

## Referências

BECK, K. et al. **Manifesto for agile software development**. 2001. Disponível em <http://agilemanifesto.org>, acessado em 29 de abril de 2010.

BOEHM, B. A Spiral Model of Software Development and Enhancement. **IEEE Computer**, 21, 5, 1988, 61-72.

CARD, S. K.; MORAN, T. P.; NEWELL, A. The Keystroke-Level Model for User Performance Time With Interactive Systems. **Communications of the ACM**, 23, 7, 1980, p. 396-410.

DENNING, P., D. *et all*. Computing as a Discipline. **Communications of the ACM**, 32, 1, 1989, p. 9-23.

DIX, A., J. FINLAY, G. ABOVD, R. BEALE. **Human-Computer Interaction**, segunda edição. New York: Prentice Hall, 1998.

HENDRICK, H W. **Macroergonomics : an introduction to work system design**. Santa Monica, Calif.: Human Factors and Ergonomics Society, 2001.

- HIX, D., H. R. HARTSON. **Developing User Interfaces : ensuring usability through product e process.** Nova Iorque: John Wiley, 1993.
- HOULGHIN, A. **QA: your last line of defense.** 2010. INSIDE RIA. Disponível em <[http://insideria.com/2010/08/qa-your-last-line-of-defense.html?utm\\_source=feedburner&utm\\_medium=feed&utm\\_campaign=Feed%3A+oreilly%2Finsideria+%28InsideRIA%29](http://insideria.com/2010/08/qa-your-last-line-of-defense.html?utm_source=feedburner&utm_medium=feed&utm_campaign=Feed%3A+oreilly%2Finsideria+%28InsideRIA%29)>, acessado em 19 de agosto de 2010.
- KAY, A. **Interface: A personal view.** In B. Laurel, B. (Ed). The Art of Human-Computer Interface Design. Addison-Wesley Publishing Company, 1990, p.191-207.
- KRUCHTEN, P. **The Rational Unified Process: An Introduction.** Addison-Wesley Professional, 1990.
- MARION, C. **What is Interaction Design and What Does It Mean to Information Designers?** 1999. Disponível em <http://mysite.verizon.net/resnx4g7/PCD/WhatIsInteractionDesign.html>, acessado em 29 de abril de 2010.
- NELSON, T. **The Right Way to Think About Software Design.** In B. Laurel, B. (Ed). The Art of Human-Computer Interface Design. Addison-Wesley Publishing Company, 1990, 191-207.
- NORMAN, D. **The psychology of everyday things.** Nova Iorque : Basic Books, 1988.
- ROYCE, W. Managing the Development of Large Software Systems. **Proceedings of IEEE WESCON**, 1, 8, 1970, 328-338.
- SCHNEIDERMAN, B. **Designing the User Interface: Strategies for Effective Human Computer Interaction.** Addison-Wesley, 1980.
- STEWART, R. **One Reason Why the "Devigner" Workflow is Important - Designer Shortage.** 2006. Disponível em <http://blog.digitalbackcountry.com/?p=593>, acessado em 29 de abril de 2010.
- van WELIE, M. **Tabs.** 2008. Disponível em <http://www.welie.com/patterns/showPattern.php?patternID=tabbing>, acessado em 19 de agosto de 2010.