UNIVERSIDADE FEDERAL DE CAMPINA GRANDE

CENTRO DE CIÊNCIA E TECNOLOGIA

COORDENAÇÃO DE PÓS-GRADUAÇÃO EM INFORMÁTICA

DISSERTAÇÃO DE MESTRADO

XPU – UM MODELO PARA O DESENVOLVIMENTO DE SISTEMAS CENTRADO NO USUÁRIO

CESAR ROCHA VASCONCELOS

FRANCILENE PROCÓPIO GARCIA
MARIA DE FÁTIMA QUEIROZ VIEIRA TURNELL
(ORIENTADORAS)

CAMPINA GRANDE
FEVEREIRO – 2004

UNIVERSIDADE FEDERAL DE CAMPINA GRANDE

CENTRO DE CIÊNCIA E TECNOLOGIA

COORDENAÇÃO DE PÓS-GRADUAÇÃO EM INFORMÁTICA

XPU – UM MODELO PARA O DESENVOLVIMENTO DE SISTEMAS CENTRADO NO USUÁRIO

CESAR ROCHA VASCONCELOS

Dissertação submetida à Coordenação de Pós-Graduação em Informática do Centro de Ciências e Tecnologia da Universidade Federal de Campina Grande como requisito parcial para obtenção do grau de mestre em Informática (MSc).

PROF^A FRANCILENE PROCÓPIO GARCIA, DSC.

PROF^A MARIA DE FÁTIMA QUEIROZ VIEIRA TURNELL, PHD.

(ORIENTADORAS)

ÁREA DE CONCENTRAÇÃO: CIÊNCIA DA COMPUTAÇÃO
LINHA DE PESQUISA: ENGENHARIA DE SOFTWARE

CAMPINA GRANDE – PARAÍBA
FEVEREIRO DE 2004

FICHA CATALOGRÁFICA

VASCONCELOS, Cesar Rocha
V331M

XPU – Um Modelo Para o Desenvolvimento de Sistemas Centrado no Usuário.

Dissertação (Mestrado) – Universidade Federal de Campina Grande, Centro de Ciências e Tecnologia, Coordenação de Pós-Graduação em Informática, Campina Grande, Paraíba, Fevereiro / 2004.

158 p. Il.

Orientadoras:

- Francilene Procópio Garcia
- Maria de Fátima Queiroz Vieira Turnell

Palavras Chaves:

1. Interface Homem-Máquina
2. Engenharia de Software
3. Processos de Desenvolvimento
4. *eXtreme Programming*
5. Método para Concepção de Interfaces

CDU – 519.683B

"XPU – UM MODELO PARA O DESENVOLVIMENTO DE SISTEMAS CENTRADO NO USUÁRIO"

CÉSAR ROCHA VASCONCELOS

DISSERTAÇÃO APROVADA EM 26.02.2004

PROFª FRANCILENE PROCÓPIO GARCIA , D.Sc
Orientadora

PROFª MARIA DE FÁTIMA Q. V. TURNELL , Ph.D
Orientadora

PROF. BERNARDO LULA JÚNIOR , Dr.
Examinador

PROFª MARIA ELIZABETH SUCUPIRA FURTADO, Drª
Examinadora

CAMPINA GRANDE – PB

Dedico este trabalho a minha família
que tanto amo.

AGRADECIMENTOS

A minha orientadora, Francilene, pela oportunidade de contar com toda a sua amizade, experiência, dedicação e paciência em cada passo na construção deste trabalho.

A minha co-orientadora, Fátima, pela confiança em que podíamos realizar um bom trabalho.

De maneira especial aos meus pais, Célia e George, e irmã, Renata, pelo amparo irrestrito em todos os momentos de minha vida. Obrigado por estarem por perto e me fazerem sempre lembrar de que jamais estive sozinho.

Aos meus verdadeiros amigos, pela imensa torcida.

A Aninha e à Coordenação do Curso.

Aos professores do DSC, pelos valorosos conhecimentos compartilhados ao longo de todo o mestrado.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE GRÁFICOS

LISTA DE QUADROS

LISTA DE TABELAS

RESUMO

Este trabalho apresenta a proposta de um modelo para o desenvolvimento de sistemas de software centrado no usuário, denominado XPU. Trata-se de uma conduta de desenvolvimento leve, iterativa e incremental que integra não somente alguns dos princípios e valores fundamentais da Engenharia de Software, mas algumas das melhores práticas do universo da Interação Homem-Computador. Em XPU, práticas de usabilidade são evidenciadas como uma característica fundamental na geração de produtos com qualidade. XPU é composto de fases que podem ser facilmente entendidas e aplicadas em um ambiente de projeto. Evidencia-se a figura do usuário final como um parceiro ativo em todo o ciclo de desenvolvimento do produto. Os métodos utilizados e adaptados na geração do processo XPU foram o *eXtreme Programming* (XP) e o Método para Concepção de Interfaces (MCI).

ABSTRACT

This work presents the proposal of a model for the user-centered software development, named XPU. This methodology represents a lightweight process that integrates not only some of the principles and fundamental values of the Software Engineering, but some of the best practices of Human-Computer Interaction universe. In XPU, the usability is evidenced as a fundamental characteristic in the quality of its software products generated. XPU is composed of phases that can be easily understood and applied in a real project. The final user is considered an active partner throughout the software development cycle. Two methods have been adapted in this approach: eXtreme Programming (XP) and Interfaces Concept Method (MCI).

Capítulo 1

Introdução

Ao longo dos anos, a informática tem se mostrado uma área de impacto significativo para a sociedade moderna. Tecnologias da informação e comunicação (TIC's) são necessárias em todas as áreas de conhecimento. A evolução das TIC's, assim como a popularização dos computadores, têm levado um número crescente de pessoas (usuários[1]) a interagir com produtos e sistemas informatizados nas diversas áreas da atividade humana. Vive-se numa época em que o computador é utilizado como ferramenta de trabalho, meio de comunicação, educação, entretenimento, comércio, etc.

Um sistema de software, para um usuário qualquer, é uma ferramenta que tem como objetivo auxiliá-lo na execução de uma determinada tarefa. Porém, o uso dessa ferramenta exige dele conhecimentos e habilidades não apenas acerca do domínio da tarefa, mas também da utilização do próprio sistema.

Aprender a usar o sistema adequadamente torna-se, portanto, uma tarefa adicional para o usuário e deve, então, ser facilitada. Facilitar o uso poderá exigir um esforço do usuário ou do sistema. Se o esforço exigido se concentrar sobre o usuário, com demandas que ultrapassem sua capacidade ou motivação, poderá ocorrer uma degradação no desempenho da atividade ou mesmo levar o usuário a abandoná-la. Portanto, o esforço do usuário deve ser minimizado, desviando-o para a "imagem" da aplicação. O sistema, então, deve se comunicar com o usuário (interface para o usuário) levando em conta suas capacidades, habilidades e ainda seus objetivos [GUE 02].

[1] De acordo com [VAS 03b], entende-se por usuários os seres humanos que irão, de fato, interagir com o sistema. É o usuário que, através dos seus sentidos, percebe e armazena as informações apresentadas para, em seguida, processá-las utilizando um raciocínio lógico. É um ser tomador de decisões. No âmbito de um projeto de software, o usuário pode ser ainda o cliente do projeto ou não.

Atualmente, uma das tendências mais marcantes do mercado produtor de software é a ênfase na interface do usuário. Se, no passado, era perfeitamente razoável sacrificar a facilidade de uso em favor da "eficiência" (estrutura interna) dos programas, hoje em dia, alguns desenvolvedores entendem que devem dedicar parte do seu tempo e recursos computacionais (por exemplo, tempo de CPU, memória, comunicação) visando obter uma melhor interação do usuário final com a aplicação. É neste sentido que vem se exigindo cada vez mais dos desenvolvedores desses produtos de software uma maior preocupação com a eficiência e qualidade da interface.

Essa mudança visível de paradigma tem, naturalmente, reflexos na área de projeto de sistemas. Torna-se necessário incluir, tão cedo quanto possível, considerações de usabilidade no ciclo de desenvolvimento do produto de software.

1.1 Problemática

Conforme foi visto anteriormente, há um grande interesse por parte das organizações que desenvolvem software em todo o mundo em melhorar a interação do usuário final com seus produtos. De acordo com Ferré [FER 01a], as técnicas de Interação Homem-Computador (*Human-Computer Interaction*) são utilizadas por desenvolvedores de sistemas em seus projetos; no entanto, e na maioria dos casos, essas técnicas ainda não estão harmoniosamente integradas com os atuais processos de desenvolvimento de software adotados no âmbito da Engenharia de Software (ES).

Constantine & Lockwood [CON 99] afirmam que, ao longo dos anos, a ES tem tradicionalmente construído sistemas de software tomando como foco principal suas funcionalidades ou ainda suas estruturas internas – um processamento lógico correto, performance adequada, organização das informações eficiente, entre outros aspectos. Conseqüentemente, a qualidade destes tipos de sistemas tem sido mensurada através de fatores como reusabilidade, portabilidade, confiabilidade, modularidade, acoplamento, robustez – aspectos prescritos na ES, mas pouco conhecidos pela maioria dos usuários finais do produto. Percebe-se, assim, que a interação do usuário com o sistema tem sido vista, muitas vezes, como um assunto de pouca relevância na construção de software.

Segundo Guerrero [GUE 02], o projeto de sistemas computacionais tem sido campo praticamente exclusivo dos profissionais de informática. A concepção de software prioriza as exigências da informática antes de responder àquelas relacionadas ao usuário. A autora afirma que os profissionais de informática (projetistas e desenvolvedores) se empenham quase sempre em definir as funções lógicas de um sistema sem antes, de fato, se preocuparem com as necessidades, habilidades físicas, cognitivas e os objetivos do usuário. Essa preocupação, em geral, não faz parte de sua formação.

Se, por um lado, alguns desenvolvedores percebem o desenvolvimento de sistemas deste modo, por outro, os profissionais da área de Interação Homem-Computador (IHC) enxergam o usuário como "a peça fundamental" durante todo o processo de desenvolvimento do produto. Eles direcionam seu foco no modo como ele ou ela interage com o sistema e quais são as tarefas desempenhadas pelos mesmos. Estes profissionais não estão interessados em metodologias para, por exemplo, fazer com que o sistema em desenvolvimento seja reutilizável, portável ou ainda fracamente acoplado.

Ferré [FER 01a] coloca que as condutas de usabilidade não são comumente endereçadas no desenvolvimento de sistemas de software. As mesmas são empregadas somente em projetos onde há um interesse explícito em assuntos relacionados à usabilidade, bem como na melhoria da qualidade da interação do(s) usuário(s) com o sistema a ser desenvolvido. Nesses tipos de projetos, os profissionais de IHC têm um papel crucial em todas as fases do ciclo de desenvolvimento do produto. O autor cita ainda que, em um mercado onde há a redução cada vez maior do tempo de construção e implantação de produtos de software, constituindo-se num fator chave na sobrevivência das organizações, muitos ainda percebem o emprego de técnicas de usabilidade durante a confecção de seus produtos como um atraso no cronograma de seus projetos. Essa abordagem há que ser repensada.

Embora a área de Usabilidade tenha feito um progresso significativo em suas metodologias, é frustrante notar que a preocupação com a interface do usuário é normalmente considerada apenas em etapas finais do projeto (momento em que grande parte das mudanças são difíceis e onerosas de serem realizadas) ou como uma parte separada deste. O resultado é que a interface, em muitos casos, passa a ser entendida

como um “apêndice” do sistema. Contribuem sobremaneira com essa situação o fato das metodologias tradicionais de concepção de sistemas não levarem em conta o usuário, do ponto vista de suas próprias características e seus objetivos; além da crença generalizada de que uma “maquilagem” gráfica é suficiente para proporcionar ao sistema as características: “fácil de aprender” e “fácil de usar”.

Outrossim, não é incomum encontrar outros times de desenvolvimento que acreditam que podem projetar todo um sistema e só então se integrarem com a equipe de usabilidade para que, dessa forma, esta última possa projetar um determinado conjunto de controles para a interface do sistema, fixar uma combinação adequada de cores ou ainda estabelecer um tipo e um tamanho de fonte mais adequados para o produto. Esta abordagem é claramente incorreta. Se um engenheiro de software percebe a usabilidade de um software desta maneira, o mesmo também tende a prover soluções onde os problemas relacionados à usabilidade irão demandar um alto custo no orçamento do seu projeto a fim de serem reparados mais tarde.

Antunes *et al.* [ANT 01], ao abordarem a utilização das técnicas de usabilidade em projetos de sistemas de software, afirmam que, durante as últimas duas décadas, a comunidade de IHC tem desenvolvido uma grande variedade de técnicas e ferramentas para coleta, especificação e avaliação de alguns requisitos de usabilidade tais como: caracterização dos usuários, estudo das tarefas efetuadas pelos mesmos, definição de ambientes de trabalho e de metas de usabilidade (eficiência, satisfação, aprendizado, entre outras). Entretanto, apesar dos desenvolvedores reconhecerem tal importância, ainda não conseguiram integrar adequadamente algumas destas técnicas com as metodologias adotadas para o desenvolvimento de sistemas oferecidas pela ES.

Alguns cenários de integração, conforme descritos em [AND 01] e [RAD 01], têm sido criados (soluções *ad hoc*) na tentativa de produzir uma combinação entre as áreas da ES e Usabilidade, mas é notável a necessidade de um modelo de desenvolvimento de sistemas estruturado que possa ser utilizado por quaisquer organizações, inclusive aquelas com características distintas das que foram apresentadas nos cenários investigados.

Em síntese, é fundamental refletir sobre a importância de se agregar condutas de usabilidade ao processo de desenvolvimento em vez de continuar a enxergá-las como

uma atividade complementar. É nessa direção que as investigações e esforços serão conduzidos nesta dissertação.

1.2 Objetivos da dissertação

1.2.1 Objetivo principal

O principal objetivo deste trabalho é propor um modelo para o desenvolvimento de sistemas de software centrado no usuário que, em particular, valorize a usabilidade como característica fundamental de qualidade.

1.2.2 Objetivos específicos

O modelo proposto deverá ainda contemplar alguns objetivos específicos:

- combinar, de maneira viável e adequada, as melhores práticas do desenvolvimento iterativo e incremental propostas pela ES com práticas de Usabilidade;
- adotar como arcabouço as quatro fases clássicas da concepção de sistemas: análise, projeto, implementação e testes;
- procurar sugerir algum suporte ferramental para a geração dos artefatos[2] envolvidos;
- oferecer flexibilidade para ser adaptado por uma organização ou equipe de desenvolvimento de acordo com suas necessidades, história, cultura e domínio.
- para favorecer o entendimento, o processo de tomada de decisões por parte dos envolvidos no projeto e ajudar a resolver questões estruturais de sistemas complexos, o modelo deve sugerir a elaboração de um projeto da arquitetura do sistema.

[2] Entende-se por artefato um documento gerado durante o processo de desenvolvimento do produto. Este documento pode ser um arquivo de código-fonte, uma parte da documentação do sistema, um diagrama, etc. Alguns artefatos são mantidos durante todo o projeto, enquanto que outros apenas durante uma certa etapa do mesmo [VAS 03b].

1.3 Relevância

A relevância deste trabalho é fundamentada na ausência de um modelo estruturado de desenvolvimento de sistemas que integre técnicas de usabilidade com processos de desenvolvimento de software convencionais da ES.

Algumas metodologias estão começando a emergir numa tentativa de agregar esses dois universos [AND 01][RAD 01]. No entanto, em vez de trazer procedimentos mais genéricos, é notório que a maior parte delas reflete estudos de casos específicos de algumas organizações e, dessa forma, são pouco flexíveis e de escopo bastante limitado.

Por se tratar de um campo de pesquisa de grande interesse nas áreas de ES e Usabilidade, é de extrema importância propor soluções voltadas na geração de um processo de desenvolvimento, harmoniosamente integrado, que possa servir de ponto de apoio para equipes de desenvolvimento que pretendam não somente endereçar técnicas de usabilidade em seus projetos, mas, ao mesmo tempo, melhorar a qualidade de seus produtos de software.

Além disso, as investigações que norteiam a construção do modelo proposto de desenvolvimento podem servir como um importante ponto de apoio na direção da sistematização de formalismos e padrões para uso nas organizações que terceirizam seus sistemas de software. Percebe-se, nas empresas terceiras contratadas, uma certa autonomia no tocante às condutas internas de desenvolvimento (metodologias, competências disponíveis, experiências, padrões e terminologias), complicando a coordenação das empresas contratantes. Como resultado deste processo, as interfaces dos usuários e entre os sistemas, presentes nesses produtos finais, podem ser bastante heterogêneas e até mesmo conflitantes.

Os esforços para a concepção deste modelo de desenvolvimento podem servir não apenas como um importante guia às empresas que desejam fazer uso intensivo das técnicas de usabilidade em seus projetos, mas também para amenizar as freqüentes divergências de usabilidade percebidas nos sistemas produzidos por terceiros.

O presente trabalho irá propiciar também uma maior integração entre o Grupo de Interface Homem-Máquina (GIHM) e o Grupo de Engenharia de Software (GES),

ambos da Universidade Federal de Campina Grande. Em particular, a participação do GIHM na estruturação do modelo de desenvolvimento centrado no usuário é fundamental. Além de tal integração dos grupos, a elaboração deste trabalho possibilitará agregar um valor maior à algumas condutas de IHC propostas pelo GIHM ao combiná-las com um processo de desenvolvimento da ES, num ciclo de desenvolvimento de sistemas mais amplo.

Por fim, a concepção de um modelo estruturado para o desenvolvimento de sistemas, que combine técnicas de usabilidade adequadamente enxertadas num processo de desenvolvimento típico, contribui para o avanço no desenvolvimento de produtos de software em geral.

1.4 Metodologia de trabalho

A metodologia utilizada no desenvolvimento deste trabalho consistiu das seguintes fases:

1 - **Estudo das técnicas de usabilidade -** primeiramente, procurou-se analisar com profundidade algumas das principais condutas atualmente adotadas na área de usabilidade. Fez-se necessário não somente compreender os processos utilizados em cada uma das condutas, seus modelos, artefatos e possível suporte ferramental, mas, sobretudo, identificar e avaliar o impacto da utilização de um determinado conjunto de procedimentos de usabilidade na especificação do modelo pretendido. Esta etapa proporcionou a fundamentação teórica necessária.

2 - **Levantamento dos processos de desenvolvimento -** esta atividade contemplou o outro lado, não menos importante, relacionado ao tema proposto: investigação dos processos de desenvolvimento de sistemas adotados no âmbito da ES. Diferentes processos, cada um com suas características e princípios, foram analisados com o objetivo de propiciar uma metodologia de desenvolvimento clara, flexível e de qualidade. Da mesma forma que na fase 1, também fez-se necessário aqui identificar e avaliar,

criteriosamente, o impacto da utilização dessas práticas da ES na especificação do modelo proposto.

3 - **Análise comparativa dos dados coletados e diagnóstico -** de posse dos dados coletados que definem um conjunto de propósitos e princípios pertencentes às áreas de Usabilidade e ES, foi possível, então, dar início à formulação e estruturação de um primeiro arcabouço do modelo. Esta fase permitiu ainda vislumbrar algumas hipóteses realistas relacionadas ao escopo do trabalho, os riscos envolvidos, os recursos exigidos, dentre outras informações existentes.

4 - **Definição e especificação do modelo -** neste momento, deu-se a transição do arcabouço para uma versão mais estruturada e, sobretudo, realista do modelo. Buscou-se especificar, estruturar e detalhar a metodologia integrando os aspectos, técnicas, artefatos e ferramentas que foram levantados em cada um dos dois universos investigados - Usabilidade e ES.

5 - **Estudo de caso e avaliação do modelo -** consistiu na aplicação do modelo proposto em um cenário real, com o objetivo de não apenas avaliar a metodologia, mas, principalmente, confrontar os resultados obtidos com alguns objetivos e aspectos específicos iniciais definidos.

1.5 Organização da dissertação

A dissertação encontra-se organizada em 6 (seis) capítulos:

Neste capítulo, o assunto é contextualizado; são apresentados a problemática, os objetivos e a motivação pela escolha do tema.

O Capítulo 2 apresenta ao leitor um panorama no que se refere à utilização de técnicas de usabilidade em projetos de sistemas de software. Alguns conceitos fundamentais relacionados à área, bem como o impacto, evolução e importância que as atividades de usabilidade exercem sobre o sucesso de um projeto de software são abordados neste capítulo. Também são apresentados a fundamentação e o detalhamento acerca da metodologia escolhida para confecção de interfaces a ser usada na definição do modelo.

Já no âmbito da ES, o Capítulo 3 provê não apenas um panorama do mercado produtor de software atual como também, de maneira mais específica, a boa aceitação que as metodologias ágeis de desenvolvimento de sistemas de software têm encontrado neste cenário. Ainda neste capítulo, será indicado o processo de desenvolvimento utilizado na proposição do modelo centrado no usuário.

No Capítulo 4 é fornecida a apresentação do modelo. Aspectos importantes relacionados à sua estrutura, tais como fluxos de trabalho, especificação de atividades, artefatos sugeridos, suporte ferramental, benefícios e limitações existentes são esclarecidos neste capítulo.

No Capítulo 5, uma proposta particular para a avaliação do modelo é apresentada ao leitor. A avaliação tem como objetivo verificar e mensurar a eficácia do modelo em projetos de desenvolvimento de software reais. Além disso, informações importantes relacionadas às amostras que foram levantadas, pessoal envolvido para participar deste evento, recursos necessários e, principalmente, os dados que foram levantados, são expostos.

No Capítulo 6, são apresentadas as conclusões deste trabalho, contribuições providas pelo mesmo, bem como as atividades futuras sugeridas.

Capítulo 2

A Usabilidade no Desenvolvimento de Sistemas de Software

2.1 Considerações iniciais

Neste capítulo, é apresentado um panorama acerca da utilização das técnicas de usabilidade em projetos de sistemas de software. Também serão discutidas as diferentes visões dos profissionais das áreas de Engenharia de Software e Usabilidade durante o desenvolvimento de um produto. É exposto ainda, neste capítulo, o impacto positivo que as condutas de usabilidade exercem na qualidade e aceitação de um sistema por parte dos usuários. Cenários de integração envolvendo as duas áreas supracitadas, propostos por algumas organizações produtoras de software, serão apresentados e examinados. Em seguida, será analisada a metodologia para concepção de interfaces escolhida na área de Usabilidade para a confecção do modelo.

2.2 Introdução

Técnicas de Interação Homem-Computador (IHC) têm se destacado no mercado produtor de software. Um número cada vez maior de organizações está começando a considerar o emprego de práticas de usabilidade no desenvolvimento de seus produtos como uma estratégia de negócio. Para uma organização de desenvolvimento de sistemas de software que está inserida no atual mercado competitivo, a falta de atributos de usabilidade em seu produto final pode implicar numa perda considerável da fatia de mercado, além de tornar seu produto pouco competitivo e de baixa qualidade.

Apesar da necessidade de se integrar técnicas de usabilidade com processos de desenvolvimento de software ser percebida por alguns times de desenvolvedores, muitos ainda não estão atentos para as possibilidades, capacidades e, sobretudo, o impacto que essas práticas podem ter não apenas no amadurecimento das suas metodologias de trabalho, mas também na qualidade dos produtos finais gerados.

Outrossim, o fato de projetistas e desenvolvedores de software não disporem de um "guia" que abrace condutas de IHC e Engenharia de Software (ES) – com informações sobre "o quê" deve ser feito, "quando" e "como" – contribui para o surgimento de sistemas com características indesejáveis como, por exemplo: (1) difíceis de aprender e de usar, (2) ineficientes, (3) pouco atrativos em relação aos objetivos dos usuários, (4) linguagem inadequada, dentre outros aspectos.

Ferré *et al.* [FER 01b], diante deste cenário, colocam que, atualmente, também é bastante comum encontrar soluções de software onde o usuário é quem se adapta ao sistema na interação homem-máquina. Entretanto, deve-se observar que esta prática é inadequada, uma vez que esta adaptação pode trazer a reboque impactos negativos em questões relacionadas à eficiência na execução das tarefas e, principalmente, à satisfação para com o produto.

Tradicionalmente e, no que diz respeito à construção de sistemas, a escolha das práticas inerentes à concepção de um produto pode variar de organização para organização. Os projetos de software podem apresentar uma variedade de características, com maiores ou menores diferenças. Entretanto, é importante notar que, atualmente, muitos deles possuem um ponto em comum: tentam construir um sistema tomando como foco principal o rigor na implementação de suas estruturas internas.

Há projetos em que a participação do cliente e usuários se dá apenas no início dos mesmos (esclarecendo regras de negócio e outros detalhes acerca do domínio do problema, por exemplo). A partir daí, nas demais fases do projeto, o usuário é quase que "esquecido" até o final do primeiro *release*[3] do produto – momento este em que a interação do usuário com o sistema será observada pela primeira vez. Esta abordagem precisa ser repensada. Usuários e projetistas de software devem se "conhecer" mais uns

[3] Entende-se por *release* o período de tempo para implementação e entrega de uma versão funcional do sistema ao cliente. Normalmente e, em um projeto, o mesmo é dividido em ciclos iterativos e incrementais de desenvolvimento [VAS 03b].

aos outros. Há que se ter em mente que o envolvimento contínuo por parte do usuário e a importância da inclusão de algumas práticas de usabilidade (por exemplo, levantamento do perfil de usuário, análise e modelagem da tarefa do usuário, etc.) devem ser cultuados ao longo de todo o projeto [VAS 03] [VAS 03a] [VAS 03b].

Neste sentido, este capítulo provê uma revisão bibliográfica situando o leitor não somente no atual cenário de desenvolvimento de sistemas, mas, principalmente, discutindo as diferentes visões contidas nos universos da IHC e ES. As dificuldades na integração destas duas áreas, os benefícios trazidos pelas técnicas de IHC em projetos de software e alguns relatos de experimentos, refletindo importantes propostas de integração entre IHC e ES realizados por organizações no mercado, também estarão sendo analisados e discutidos no decorrer deste capítulo. Em seguida, com base em discussões de algumas metodologias da área de IHC, será comentada e detalhada a metodologia para concepção de interfaces escolhida na integração e confecção do modelo XPU (*eXtreme Programming* + Usabilidade (MCI)).

2.3 Engenharia de Software e Usabilidade

Medeiros [MED 98] afirma que o termo Engenharia de Software foi mencionado pela primeira vez em 1968, em um encontro histórico que reuniu, na época, cientistas da Computação, programadores e gerentes da indústria de software em Garmisch, Alemanha. O termo, em particular, buscava uma nova direção formal para a confecção de software, baseada em métodos, técnicas e ferramentas de produção tal qual utilizava-se em outras áreas da engenharia.

Mais tarde, em meados da década de 90, a IEEE (*Institute of Electrical and Electronics Engineers*) [IEE 90] veio apresentar uma definição para ES que é, atualmente, bastante conhecida por toda a comunidade de desenvolvimento de sistemas:

> *"A Engenharia de Software é a aplicação de um processo sistemático, disciplinado e quantificado à concepção, implementação e manutenção do software".*

Em outras palavras, a ES provê métodos que refletem detalhes de "como agir" para se construir um determinado software [DIA 02]. Segundo Pressman [PRE 95], de forma não tão abrangente, esses métodos envolvem um amplo conjunto de tarefas que incluem: (1) planejamentos e estimativas de projeto, (2) análise de requisitos de software, (3) projeto da estrutura de dados e arquitetura de programas, (4) codificação, (5) testes e (6) manutenção. Para um engenheiro de software, existe, portanto, uma clara e conhecida visão do ciclo de desenvolvimento de um sistema.

Pode-se dizer, ainda, que a definição apresentada pela IEEE constitui, hoje, o paradigma predominante no desenvolvimento de pequenos, médios e grandes projetos de sistemas de software. Contudo, deve-se observar que, nesta busca por métodos sistemáticos e disciplinados, a usabilidade tem sido tradicionalmente deixada em segundo plano, fazendo, portanto, com que os processos de desenvolvimento da ES passem a não endereçar produtos de software com um nível adequado de usabilidade.

2.3.1 Visão tradicional da Usabilidade

Ferré [FER 03], ao abordar a integração e utilização das técnicas de usabilidade em processos da ES, afirma que muitas vezes e, equivocadamente, a usabilidade é vista por engenheiros de software e desenvolvedores como sendo somente o processo de construção da interface com o usuário (cores, fontes, etc.), o qual, segundo eles, pode ser realizado após as funcionalidades internas do sistema terem sido implementadas.

Neste sentido, o autor coloca ainda que um projeto de software centrado no usuário final se opõe ao outro que possui seu foco na implementação dos componentes funcionais internos (estruturas internas) de um sistema. O padrão ISO 13407:1999 [ISO 99], ao abordar os principais conceitos de um processo centrado no usuário, corrobora este fato, afirmando que um processo centrado no usuário deve possuir algumas atividades e artefatos de IHC a serem gerados durante todo o processo. Além disso, o padrão sugere que estas atividades e artefatos fiquem sob a responsabilidade exclusiva de um especialista em usabilidade, enquanto que as demais práticas, relacionadas à ES, devem ser executadas pelo desenvolvedor de software.

As referências da literatura de IHC são unânimes quanto ao papel e a influência das técnicas de usabilidade sobre a qualidade e a aceitação final do produto por parte

do(s) usuário(s). Todavia, na visão de boa parte da comunidade de ES, esses conceitos e afirmações nem sempre estão fortemente arraigados.

Considerando que a usabilidade diz respeito a outros aspectos além da aparência da interface do usuário, no contexto deste trabalho adotar-se-á a definição de usabilidade com base na norma ISO 9241-11 [ISO 98]: "*Até que ponto um produto pode ser utilizado por usuários específicos para alcançar metas também especificadas com eficácia, eficiência e satisfação, em um determinado contexto de uso*".

Ferré [FER 03] coloca que, apesar da usabilidade ter sido diretamente mencionada já na década de 70 como um fator de qualidade na literatura de ES, não havia uma orientação sobre como ela deveria ser combinada com as técnicas de desenvolvimento de sistemas. Uma indicação superficial relacionada à usabilidade não passava de uma descrição bastante genérica, informando que o sistema de software a ser produzido deveria ser "utilizável". Mais tarde, uma relação entre as condutas de IHC e processos de desenvolvimento da ES pôde ser encontrada nos padrões ISO/IEC 12207:1995 [ISO 95] e IEEE 1074-1997 [IEE 97].

Mesmo assim, vislumbrar uma integração harmoniosa e adequada entre as áreas de IHC e ES, com base nos padrões citados, consistia num grande desafio; isso devido a alguns fatores, como: (1) o usuário, quando mencionado, atuava muitas vezes de forma passiva nas atividades que ele podia participar e (2) não era oferecido um guia sobre como gerenciar atividades de IHC e, portanto, acompanhar e controlar a evolução da usabilidade de um produto de software.

Uma emenda para o padrão ISO/IEC 12207:1995 [ISO 02], aprovada em 2002, é um pouco mais detalhada que sua versão inicial. Com a inclusão de um novo processo, denominado Processo de Usabilidade, a ISO reconheceu a importância de se gerenciar atividades de usabilidade no processo de desenvolvimento de sistemas. A norma sugere um novo integrante na equipe de desenvolvimento – o Especialista em Usabilidade – que tem o papel de definir, acompanhar e validar as atividades de IHC durante a concepção do produto. A ISO cita ainda que algumas atividades de usabilidade são de responsabilidade deste especialista, enquanto que outras devem ser executadas com a(s) presença(s) indispensável(is) do(s) usuário(s).

O fato de um padrão internacional fazer alusão às técnicas de usabilidade no processo de concepção de um produto de software é uma clara indicação de mudança no paradigma do cenário de desenvolvimento de software em todo o mundo.

2.3.2 Dicotomia - Engenharia de Software e Interação Homem-Computador

Uma das virtudes do campo da IHC é a sua essência multidisciplinar. Entretanto, essa característica torna-se, ao mesmo tempo, a principal barreira para sua integração com a ES; isto porque enquanto as técnicas, metodologias e valores da Interação Homem-Computador estão embasadas em disciplinas como a Psicologia, Sociologia, Design Industrial, Comunicação, Design Gráfico e assim por diante, os engenheiros de software, normalmente, possuem uma visão bem diferente: eles trabalham sob um aspecto fundamentalmente técnico centrado na construção de código. Assim, pode-se concluir que, apesar desses dois campos lidarem com uma atividade comum - o desenvolvimento de sistemas de software - eles operam com perspectivas muito diferentes.

Seffah [SEF 02], ao abordar a dicotomia existente entre as áreas de IHC e ES, cita algumas diferenças entre os desenvolvimentos de software tradicional e aquele centrado no usuário, tomando, como cenário, o atual mercado produtor de software. Para ilustrar o que foi dito anteriormente, pode-se observar, na Figura 1, as principais divergências entre a filosofia tradicional de desenvolvimento de software (que possui seu foco na tecnologia e na implementação das estruturas internas de um programa) e a filosofia de desenvolvimento centrada no usuário final:

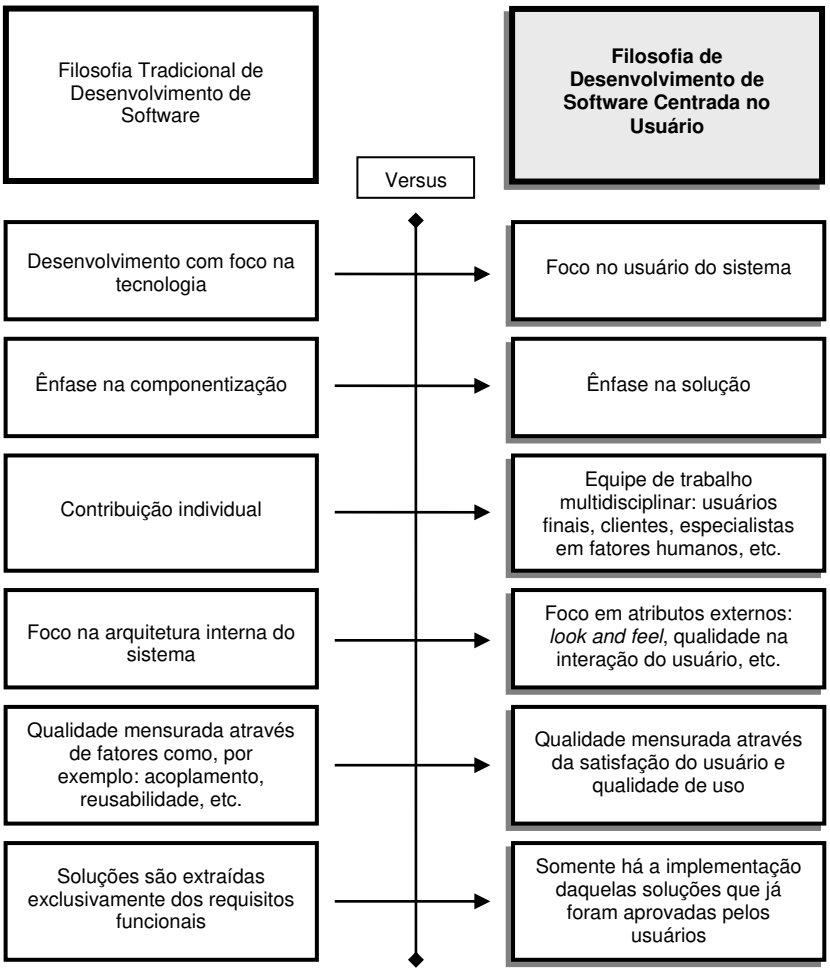

Figura 1: Dicotomia Engenharia de Software e Interação Homem-Computador [SEF 02].

Embora reconhecendo a dicotomia, é importante enfatizar que não se pode privilegiar apenas a IHC ou a ES. Devem-se cultuar ambas. É uma relação inclusiva e não excludente. O engenheiro de software deve estar ciente dos benefícios oriundos das técnicas de usabilidade e o impacto que elas detêm na qualidade e aceitação de um produto de software por parte dos seus usuários.

Neste sentido, Cybis *et al.* [CYB 98], ao abordarem a concepção de sistemas interativos, colocam que o desenvolvimento de sistemas interativos de qualidade exige a utilização conjunta e integrada de conceitos e técnicas específicas do domínio da IHC e de conceitos e métodos de desenvolvimento de sistemas considerados integrantes do domínio da ES.

2.4 Importância das técnicas de usabilidade

Segundo Cybis *et al.* [CYB 98] a qualidade geral percebida de um software, bem como o desempenho e produtividade do usuário, são fatores intimamente ligados à qualidade da interface e interação deste usuário com o sistema. Por sua vez, Ferré *et al.* [FER 01b] afirmam que as técnicas de usabilidade podem ajudar os usuários de qualquer sistema de software a efetuar suas tarefas, fazendo com que o sistema venha também a alcançar plenamente seu objetivo.

Muitos são os benefícios ganhos em se empregar bem as técnicas de usabilidade em um processo de desenvolvimento. Dentre eles, destacam-se:

- redução no tempo de treinamento do usuário;
- aumento na produtividade do usuário;
- aumento de satisfação do usuário;
- redução nos custos de *marketing*;
- impacto nas vendas do produto e sua penetração no mercado;
- redução nos custos de manutenção;
- aumento da qualidade das tarefas desempenhadas pelo usuário;
- redução nas mudanças em fases avançadas de projeto;
- redução no número de erros cometidos pelo usuário e,
- redução na demanda por suporte ao usuário.

Logo, a usabilidade é crítica para a aceitação de um sistema de software por parte de um usuário. Se o produto não provê suporte na realização de suas tarefas, provavelmente será rejeitado. É possível que haja também uma subutilização dos recursos oferecidos pelo sistema, descontentamento de usuários e até mesmo o fracasso de um projeto inteiro. Vale ressaltar que, mesmo que um produto esteja sendo utilizado por seus usuários, isso não significa necessariamente que o software possua um nível adequado de usabilidade. E importante destacar que existem outros aspectos que condicionam o uso de um produto de software, a exemplo da capacidade de escolha do(s) seu(s) usuário(s), baseada em opções de produtos similares e dos custos associados às escolhas.

Assim, de um modo geral, a principal razão para se aplicar práticas de usabilidade nos processos de desenvolvimento que norteiam a concepção de um produto de software é aumentar a eficiência, satisfação e, conseqüentemente, a produtividade do usuário.

2.4.1 O envolvimento do usuário em projetos de software

É muito importante frisar que além da adoção de técnicas de usabilidade, uma maior participação do usuário ao longo do projeto deve ser enfatizada.

Este argumento é apoiado por vários autores, entre eles, Wang *apud* Stábile [STÁ 01], que ao abordar a presença do usuário em projetos de sistemas, destaca que a desconexão entre usuários e projetistas, em muitos projetos de software, afeta bastante a qualidade dos sistemas de informações disponibilizados; isso porque quanto maior a ênfase no usuário final, maior a probabilidade de sucesso decorrente da participação de quem realmente conhece suas necessidades informacionais.

Moraes [MOR 98] apresenta um importante estudo que dá uma idéia clara acerca do poder de decisão dos usuários em projetos de software: o autor afirma que 35% (trinta e cinco porcento) dos usuários envolvidos em projetos não têm autonomia para aprovar os sistemas que são desenvolvidos para eles mesmos.

Whitaker [WHI 94] cita que uma razão para o afastamento dos usuários em projetos de sistemas de software se deve ao fato de, em geral, os profissionais de informática não serem bons na comunicação e entrosamento com os clientes. Gause & Weinberg [GAU 91] colocam uma outra razão que leva os usuários a se distanciarem dos projetos: o fato dos projetistas os tratarem de forma padronizada, tomando, na maioria das vezes, decisões que não lhes competem. Neste sentido, os autores alertam que a maioria dos participantes é leiga somente com relação ao processo de desenvolvimento, mas perita nos assuntos nos quais os engenheiros são leigos. Portanto, sua participação ao longo de um projeto de software é extremamente necessária. Estimam, ainda, que bilhões de dólares são desperdiçados todos os anos fabricando-se produtos que não atendem aos requisitos dos usuários, na maioria dos casos, porque os requisitos do sistema não foram entendidos de forma clara.

Dados do estudo Chaos, realizado sistematicamente pelo Standish Group [JOH 01], também refletem a importância do envolvimento do usuário no progresso e sucesso de um projeto de software. Na busca pelos principais fatores que têm influência direta no sucesso dos projetos, dez deles foram destacados. Para cada fator encontrado, foi atribuído um peso, indicando, portanto, o impacto que cada um possui no êxito de um projeto. Neste sentido, o estudo estabelece uma espécie de "Receita do Sucesso", a qual pode ser seguida pelas organizações e por seus times de desenvolvimento. O estudo aponta ainda que os projetos que centrarem seus esforços nos fatores de maior pontuação terão maiores chances de serem concluídos com sucesso e, portanto, com risco menor de apresentarem falhas, ou pior: de não serem concluídos. A Tabela 1 ilustra os resultados deste estudo:

A Receita do Sucesso	
Fatores de sucesso	**Pesos**
Suporte Executivo	18
Envolvimento do usuário	16
Experiência do gerente de projeto	14
Clareza nos objetivos de negócio	12
Escopo minimizado	10
Infra-estrutura padrão	8
Estabelecimento dos requisitos básicos	6
Uso de metodologias formais	6
Estimativas confiáveis	5
Outros	5

Tabela 1: Fatores que influenciam diretamente o sucesso de um projeto de software [JOH 01].

De acordo com a Tabela 1, pode-se observar que mesmo quando um projeto é concluído dentro do prazo ou ainda com um orçamento reduzido, ele ainda pode falhar por não atender às expectativas dos seus usuários.

Vale lembrar que dados oriundos de um estudo Chaos anterior ao apresentado, realizado no ano de 1998 [SGI 98], mostram que, tradicionalmente, o envolvimento do usuário no projeto tem estado em primeiro lugar no *ranking* dos dez fatores que têm influência direta no sucesso de um projeto de software. Entretanto, em 2001, este estudo foi atualizado e o fator-usuário deslocado para a segunda posição. A respeito desta queda de posto, o estudo afirma que não quer dizer que o envolvimento do usuário tenha

perdido sua importância; e sim que muitos times de desenvolvimento, bastante preocupados com o primeiro item da lista anterior de 1998, passaram a rever seus conceitos junto aos usuários.

2.5 Cenários de Integração IHC e ES

Ultimamente, algumas organizações produtoras de software têm procurado endereçar condutas de usabilidade em seus projetos. De acordo com Ferré [FER 03], é possível classificar os experimentos de integração entre IHC e ES em dois grandes grupos: soluções *ad hoc* e propostas com um caráter genérico.

Nesta seção, são apresentados dois cenários relacionados à primeira classificação. Em seguida, será discutido um experimento realizado à luz da segunda classificação. Ao final de cada estudo de caso, será oferecida uma breve análise.

2.5.1 Soluções *ad hoc*

No primeiro cenário, realizado por Radle & Young [RAD 01], os autores afirmam que não há uma única e, portanto, melhor maneira de se implantar técnicas de usabilidade em uma organização produtora de software. Isso se deve ao fato de cada ambiente ser único e possuir características distintas. Os cenários apresentados por eles descrevem como três empresas procuraram integrar condutas de IHC em seus projetos. Em seus estudos, os autores destacam a importância de especialistas de usabilidade não somente na composição das equipes de desenvolvimento, mas também no relacionamento com os desenvolvedores. É enfatizada ainda uma lista de atividades de usabilidade que serão assistidas pelos especialistas em IHC: (1) análise das necessidades dos usuários, (2) definição de cenários para deliberar quais tarefas serão desempenhadas pelos usuários e (3) prototipagem.

É sabido que testes de usabilidade são atividades prevalecentes na engenharia da usabilidade [FER 03]. Estes, normalmente, requerem um planejamento preliminar não-trivial, infra-estrutura tipicamente onerosa, pessoal especializado, além de tempo por parte dos envolvidos nesta tarefa. Testes desta natureza devem sem realizados repetidas vezes com os usuários do sistema, sob condições controladas, até que se encontrem

resultados satisfatórios. Não obstante a importância dada aos testes de usabilidade pelos seus autores, estes não são suficientemente destacados e detalhados neste cenário, resultando em ceticismo no que se refere à qualidade na avaliação da usabilidade dos produtos gerados.

O segundo cenário de integração, proposto por Anderson *et al.* [AND 01], propõe a reunião de técnicas de IHC com um processo da ES conhecido como Processo Unificado (*Unified Process*) [JAC 99] em uma organização.

Atualmente, um dos mais populares processos de desenvolvimento de sistemas orientados a objetos é o Processo Unificado [JAC 99]. Trata-se de um processo denso (*heavy*) – sobretudo na quantidade e detalhamento dos artefatos a serem produzidos – mas que reúne algumas das melhores práticas da indústria de desenvolvimento de software. Grande parte das atividades presentes nessa metodologia é descrita a partir de modelos, os quais são suportados pela linguagem UML (*Unified Modeling Language*) [BOO 00]. A partir da linguagem UML, o projetista: especifica, visualiza, constrói e documenta artefatos de sistemas de software que o ajudam a construir sistemas grandes e complexos, através de modelos prontos, expressivos e independentes da linguagem de programação empregada. Com a UML, o desenvolvedor consegue representar e entender o sistema que está sendo construído, tanto a partir de uma visão natural, quanto técnica.

Ainda no âmbito do segundo cenário apresentado por Anderson, foi gerada uma instância do Processo Unificado subsidiada com base nas necessidades, processos de negócio e demais características peculiares à organização envolvida no cenário. Essa abordagem, atualmente, é muito utilizada por diversas empresas em todo o mundo, uma vez que este processo da ES é bastante denso e, portanto, necessita de ser adaptado conforme as necessidades de cada ambiente. Porém, deve-se notar que isso constitui um grande obstáculo às outras empresas produtoras de software do mercado que queiram se orientar através deste cenário, isto porque mesmo que estas últimas utilizem o Processo Unificado – o que é bastante improvável, face à diversidade de processos de ES existentes no mercado – suas culturas organizacionais, processos internos, valores, características e necessidades seriam, seguramente, diferentes daqueles da organização original de Anderson.

Vale lembrar ainda que, no Processo Unificado, os detalhes relacionados à estrutura interna do sistema são normalmente deslocados para a fase de projeto. Ferré [FER 01a], considera o Processo Unificado basicamente orientado ao *design* e, dessa forma, mais voltado à implementação.

Com base em toda esta exposição, pode-se perceber que as duas propostas de integração supracitadas constituem soluções *ad hoc*, as quais foram concebidas e adaptadas para cenários específicos de organizações particulares e, por conseguinte, de escopo bastante reduzido.

2.5.2 Soluções de caráter genérico

De natureza mais genérica é o experimento de integração entre as práticas de IHC e ES proposto por Costabile [COS 01]. Este experimento, em concordância com os dois cenários apresentados na seção anterior, tem como objetivo aumentar a usabilidade e, assim, a qualidade dos produtos de software. Inicialmente, o autor reúne as atividades de usabilidade em três grupos: (1) análise dos usuários e das tarefas a serem desempenhadas, (2) projeto e implementação iterativa dos protótipos e (3) avaliação dos protótipos e decisões de projeto com o(s) usuário(s). Após a definição deste processo, é proposta uma integração destas atividades com o modelo "Cascata" de desenvolvimento de software, da ES.

O Modelo Cascata (ou Clássico), conhecido por muitos profissionais pela sua abordagem *top-down*, surgiu na década de 70 e, até meados da década de 80, foi um modelo de desenvolvimento de aceitação geral pela comunidade de ES. Mesmo sofrendo algumas variações ao longo do tempo, este modelo de desenvolvimento pressupõe o mesmo conceito geral: a idéia puramente seqüencial; onde uma etapa fornecerá saídas que serão utilizadas como entradas para a etapa seguinte.

O modelo cascata talvez seja um dos paradigmas mais vistos e amplamente empregados na ES, porém sua aplicabilidade, em muitos campos, tem sido fortemente questionada. Isso se deve ao fato de, atualmente, os projetos raramente seguirem um fluxo seqüencial rígido, proposto no modelo. Na verdade, hoje, é o desenvolvimento iterativo e incremental que impera nos ambientes de projetos de sistemas, criando, portanto, problemas na aplicação deste modelo.

Vale ressaltar ainda que, o modelo cascata só é bem aplicado em situações nas quais o software a ser desenvolvido é simples, os requisitos são muito bem conhecidos, a tecnologia usada é bem acessível e os recursos para o desenvolvimento estão disponíveis – o que, muitas vezes, não condiz com a realidade da maioria dos ambientes de projetos de software atuais.

Outrossim, um processo de desenvolvimento como proposto no modelo cascata, que não prevê suporte às modificações nos requisitos, não está de acordo com a realidade da indústria de desenvolvimento de sistemas, isso porque é bastante difícil para o cliente especificar todos os requisitos do sistema, como resultado da incerteza natural no início de qualquer projeto.

Dentre outros problemas deste modelo, pode-se citar ainda: (1) não fornece *feedback* entre as fases e não permite a atualização ou correção das fases anteriores, (2) não prevê mudanças nos requisitos, (3) devido a sua filosofia puramente seqüencial, o cliente deve ser muito paciente, pois a versão funcional do sistema não estará disponível até o final do desenvolvimento e (4) não prevê atividades de manutenção.

Apesar destas dificuldades, vale lembrar que o modelo Cascata tem um lugar importante e bem definido em muitos trabalhos de ES.

Com base no contexto apresentado nesta seção, fica claro que é essencial definir um modelo de desenvolvimento que integre as áreas de IHC e ES - combinando e coordenando, mais especificamente, seus valores, atividades, artefatos e ferramentas de trabalho - adequadamente. Percebe-se que a falta de um modelo mais genérico, que possa ser adaptado por um conjunto maior de organizações, aumenta a demanda por pesquisas neste assunto.

2.6 Metodologias para concepção de interface baseadas na tarefa

Nesta seção, serão apresentadas algumas metodologias existentes para concepção de interfaces que se baseiam no modelo da tarefa. Descrevendo a tarefa do usuário de forma precisa e exaustiva, abre-se espaço para a aplicação de regras

ergonômicas que associem elementos obtidos na análise da tarefa com elementos da interação, ou seja, torna possível relacionar objetivos e características da tarefa com as características da interface que se pretende conceber.

Na concepção de interfaces, a análise da tarefa tem o objetivo de descrever uma tarefa[4] precisamente com o intuito de entender a "lógica do usuário" [GUE 01a][SCA 88]. Analisar a tarefa significa estudar os detalhes de sua natureza, seu propósito, as partes que a compõem, a forma e a ordem em que devem ser realizadas. Neste contexto, Sousa [SOU 99] coloca que esta análise envolve também o fluxo de informação e o papel do usuário do sistema.

Barbosa *et al.* [BAR 02] afirmam que, para auxiliar o projetista de interfaces ao longo do processo de concepção, as metodologias de concepção de interfaces, em geral, definem e fazem uso não somente de diferentes modelos (modelos de tarefa, de interação, entre outros) para representar o conhecimento, mas também de ferramentas computacionais para apoiar a produção das especificações em cada etapa do processo.

Guerrero [GUE 01] coloca que, quando o processo de concepção de uma interface tem como base a integração da representação da tarefa do usuário, fala-se da abordagem baseada na tarefa. Esta, é uma especialização da abordagem baseada em modelos, com a particularidade de que o principal modelo utilizado é o da tarefa do usuário. A mesma autora cita ainda que as metodologias que se baseiam no modelo da tarefa do usuário constituem um apoio expressivo à concepção de interfaces. Elas permitem, a partir da descrição da tarefa, do perfil do usuário e de princípios ergonômicos, a construção de uma especificação executável (protótipo) da interface levando em conta os objetivos do usuário.

As principais etapas do processo definido pelas metodologias baseadas na tarefa são:

- análise da tarefa e das características do usuário;

[4] O termo tarefa apresentado neste contexto refere-se à atividade que o usuário deseja desempenhar no sistema - sua natureza, propósito, partes que a compõem e ordem em que devem ser realizadas; e não à tarefa de programação (*e.g.*, gerar um *script* de consulta a uma tabela de um banco de dados, gerar código de teste para um método, etc.) as quais são normalmente atômicas, realizadas exclusivamente pelos desenvolvedores e citadas em alguns processos de desenvolvimento de sistemas da Engenharia de Software.

- definição das representações abstratas da interação;
- definição de especificações para a implementação e,
- avaliação do protótipo da interface.

Vários formalismos têm surgido com o objetivo de integrar a análise e a descrição da tarefa ao processo de concepção de interfaces. Dentre alguns, pode-se destacar: **TKS** [JOH 88], **MAD** [SCA 89], **ETAG** [TAU 88] [TAU 90] [HAA 92] [HAA 00], **TAOS** [MED 95] [MED 00] e **MAD*** [HAM 95] [GAM 98]. Todos assumem que o conhecimento sobre uma tarefa é estruturado de acordo com o paradigma do planejamento hierárquico.

Serão expostas, a seguir, três dessas metodologias para concepção de interfaces: **MEDITE, MCI e MACIA**, além de suas características e especificidades:

2.6.1 MEDITE (Mad* + EDITor + Ergonomia)

De acordo com Guerrero [GUE 01b] [GUE 02], MEDITE é uma metodologia para a concepção de interfaces ergonômicas orientada a modelos e baseada na análise e descrição da tarefa. Concebido no ambiente do Grupo de Interface Homem-Máquina (GIHM) da Universidade Federal de Campina Grande (UFCG), o seu desenvolvimento surgiu de um estudo inicial de diferentes metodologias para a concepção de interfaces baseadas na tarefa; o que, por conseguinte, fez com que o processo de desenvolvimento definido por MEDITE partisse da descrição da tarefa e do perfil do usuário.

De forma mais detalhada, MEDITE reflete um método que, utilizando-se de regras formuladas a partir do conhecimento ergonômico já existente, guia a construção da especificação conceitual e arquitetural da interface segundo um modelo bem definido a partir do modelo da tarefa. Os modelos considerados nestas abordagens são: MAD* [HAM 95] [GAM 98] para análise, descrição e modelagem das tarefas do usuário e EDITOR [LUL 92] para modelagem e especificação da interface. Cada etapa do processo utiliza um modelo que permite a construção de artefatos específicos à etapa. O conhecimento ergonômico é integrado na forma de uma base de regras utilizadas para auxiliar o projetista de interfaces na passagem do modelo da tarefa para o modelo da interação.

MEDITE define um processo geral de construção de interfaces ergonômicas dividido em 5 (cinco) etapas:

1) análise da tarefa e do usuário;
2) especificação conceitual inicial da interação (construção da Árvore EDITOR);
3) refinamento da especificação conceitual (definição dos atributos);
4) geração do protótipo e,
5) avaliação (distribuída em todas as outras etapas do processo).

São apresentados, nesta parte, os processos, os produtos, as etapas e os modelos envolvidos na metodologia MEDITE. A Figura 2, abaixo, ilustra os *processos* através de círculos e por meio de retângulos os *produtos* gerados e as *ferramentas conceituais* (modelos) utilizadas em cada processo:

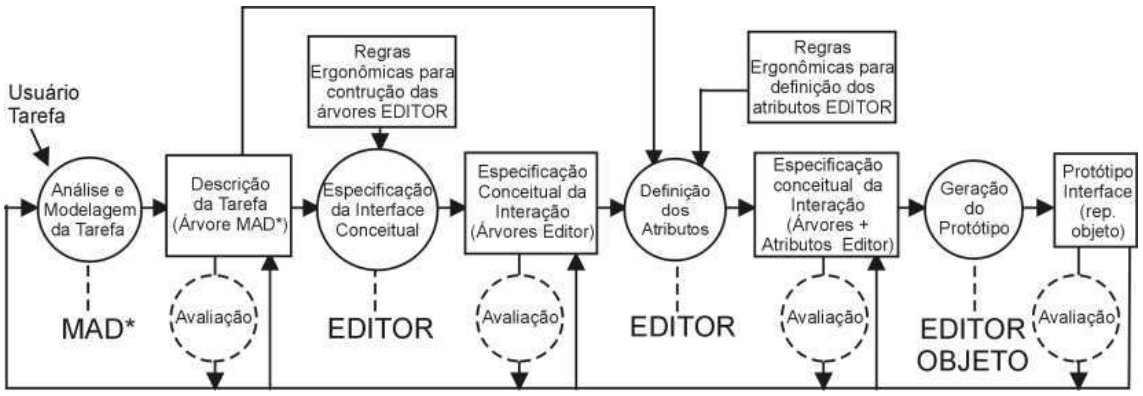

Figura 2: Metodologia MEDITE [GUE 02].

A partir da Figura 2, pode-se notar que o processo de especificação definido em MEDITE é iterativo (de forma que cada etapa permite um retorno às etapas precedentes com o objetivo de avaliar os produtos gerados) e incremental, uma vez que cada produto serve como entrada para a etapa seguinte.

Vale ressaltar que uma das virtudes do MEDITE deve-se ao fato desta metodologia se caracterizar pela sua simplicidade e objetividade. Isso dá ao projetista de interfaces um rápido e fácil entendimento do método. Além disso, um outro ponto positivo é que a presença do usuário é enfatizada ao longo da execução de suas atividades.

Na época em que a metodologia MEDITE foi concebida, todos os seus processos eram realizados manualmente, ou seja, era uma metodologia que não dispunha de ferramentas computacionais. Segundo sua autora, a inexistência de suporte ferramental poderia vir a impactar negativamente sua utilização uma vez que, dependendo do tamanho e complexidade do sistema, sua aplicação poderia ser impraticável, resultante da perda de noção de coerência e completude da descrição.

Diante deste cenário, o GIHM/UFCG investiu na concepção de uma interessante ferramenta de suporte a análise e modelagem da tarefa - iTAOS [COR 03] [MED 03] - a qual foi incorporada ao MEDITE. Atualmente, é prevista a concepção de mais um ambiente computacional para auxiliar a modelagem da interação (implementação do Modelo EDITOR). A ferramenta iTAOS utiliza o formalismo TAOS [MED 95] [MED 00] para efetuar a modelagem da tarefa.

2.6.2 Metodologia de Assistência à Concepção e à realização de Interfaces Adaptadas (MACIA)

De acordo com Furtado [FUR 97] [FUR 99], MACIA se caracteriza por descrever um processo incremental e iterativo próprio ao desenvolvimento de um sistema interativo de supervisão, onde as interfaces são geradas automaticamente a partir do modelo de tarefa feito pelo analista de sistema.

Conforme apresentado abaixo, MACIA define um processo de construção de interfaces dividido em 5 (cinco) etapas:

1) análise do domínio;
2) modelagem do domínio;
3) especificação da interface conceitual;
4) elaboração da arquitetura da interface real e,
5) geração/utilização/adaptação das interfaces reais e seus recursos associados (modelos e regras).

Ainda, segundo Furtado, esta metodologia integra fatores humanos no processo de desenvolvimento das interfaces, a fim de diminuir a complexidade do projeto das mesmas e melhorar a usabilidade das já construídas. Além do processo automático de geração de interfaces descrito por MACIA, a metodologia basicamente ajuda o analista

de sistema durante o desenvolvimento de um sistema informático através de dois modelos: 1) modelo de tarefa inicial: funciona como o "pontapé inicial", ajudando o analista a compreender melhor as tarefas que devem ser realizadas pelo operador do sistema, evitando que o analista parta do início na modelagem do domínio e, 2) modelo de interface conceitual: funciona como um protótipo das interfaces, permitindo ao analista validá-las antes mesmo de suas gerações definitivas.

É apresentado a seguir, na Figura 3, o ciclo de concepção da metodologia MACIA:

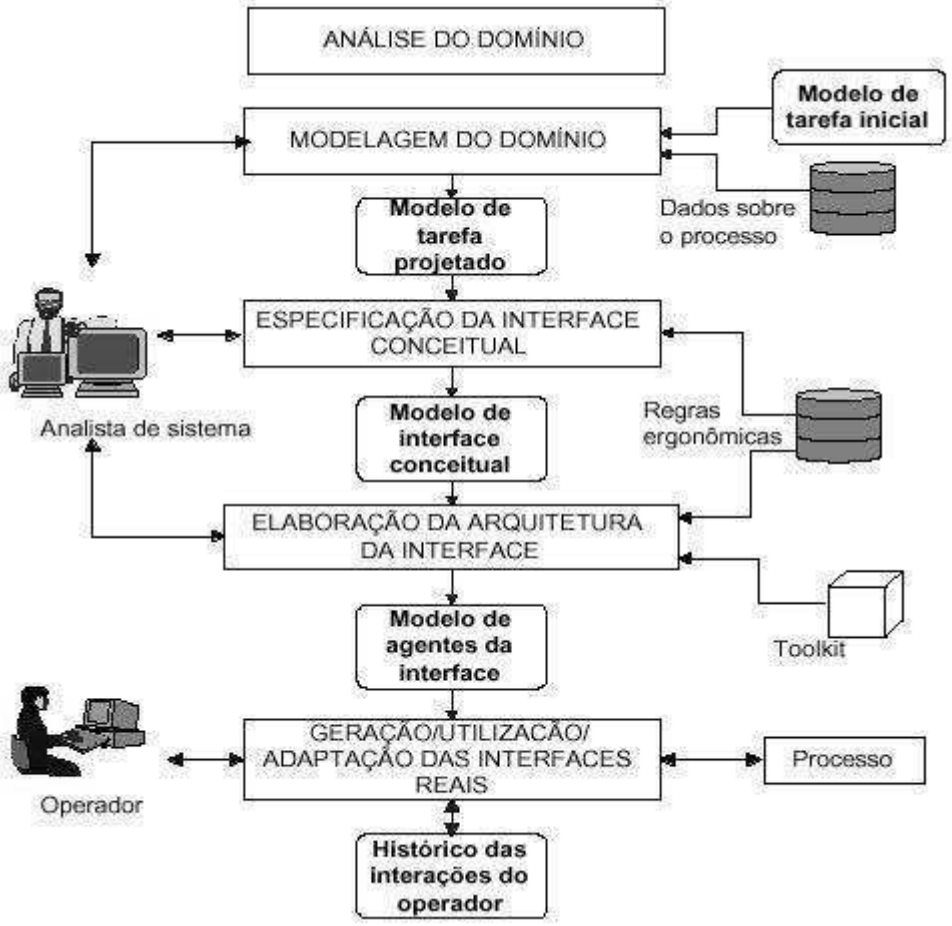

Figura 3: Metodologia MACIA [FUR 99].

2.6.3 Método para Concepção de Interfaces (MCI)

De acordo com Turnell [TUR 02], o MCI (Método para Concepção de Interfaces) objetiva fornecer suporte à concepção de interfaces visando produtos centrados no usuário. Desta forma, seu método de concepção visa abranger as questões

ergonômicas baseando-se no ciclo de vida de produtos interativos. Da mesma forma que o MEDITE, citado anteriormente, trata-se de um método do GIHM/UFCG que possui uma abordagem iterativa e incremental. Assim, ao final de cada etapa, tem-se a geração de um artefato que, por sua vez, é submetido à avaliação. Uma vez aprovado este artefato, este se constitui na entrada da etapa seguinte.

O MCI é um método para o projeto da interação que aborda tanto o nível conceitual, (a identificação das funções necessárias, o sequenciamento destas funções e a definição do fluxo da interação), quanto o nível perceptivo (projeto da representação visual para o usuário).

Este método tem como principal característica o ciclo de concepção centrado na avaliação, de forma que os resultados de cada etapa são avaliados antes de prosseguir no processo ou retornar a etapas anteriores. Uma vez que sua formalização originou-se de trabalhos na área de avaliação da usabilidade de produtos [SOU 99] e [TUR 02], o processo de avaliação é bem definido e conta com ferramentas desenvolvidas no GIHM/UFCG.

De acordo com Turnell [TUR 02], as principais etapas do ciclo de vida de produtos interativos são:

- **análise e especificação de requisitos:** consiste em um processo formal de especificação dos requisitos para o projeto do sistema, de forma a determinar os objetivos básicos e características desejadas para o produto;
- **análise do usuário:** consiste em definir o perfil dos usuários potenciais do sistema, visando coletar as tarefas, aptidões e conhecimentos necessários para realizá-las;
- **análise das tarefas:** descrição detalhada das tarefas, sub-tarefas e métodos envolvidos na utilização do sistema;
- **análise funcional:** visão interna das funções técnicas a serem projetadas no componente funcional do sistema e que, por fim, serão combinadas com a interface.

2.6.3.1 Etapas do método MCI

Com base nos preceitos expostos acima, são apresentadas, abaixo, as principais etapas do método MCI [TUR 02]:

- **Análise e Especificação de Requisitos**
  - Levantamento das necessidades do cliente;
  - Levantamento do perfil do usuário;
  - Análise e modelagem da tarefa e,
  - Estabelecimento de objetivos de usabilidade.
- **Concepção do Modelo da Interação**
  - Levantamento dos objetivos e das ações envolvidas na realização da tarefa;
  - Seleção e representação de cenários de interação;
  - Concepção das metáforas e dos manipuladores para execução das ações;
  - Concepção do projeto visual e,
  - Concepção dos mecanismos de navegação e ajuda.
- **Construção de um protótipo relativo aos cenários selecionados**
  - Geração e avaliação do protótipo.

A Figura 4 ilustra o ciclo de concepção do método [SCH 03]:

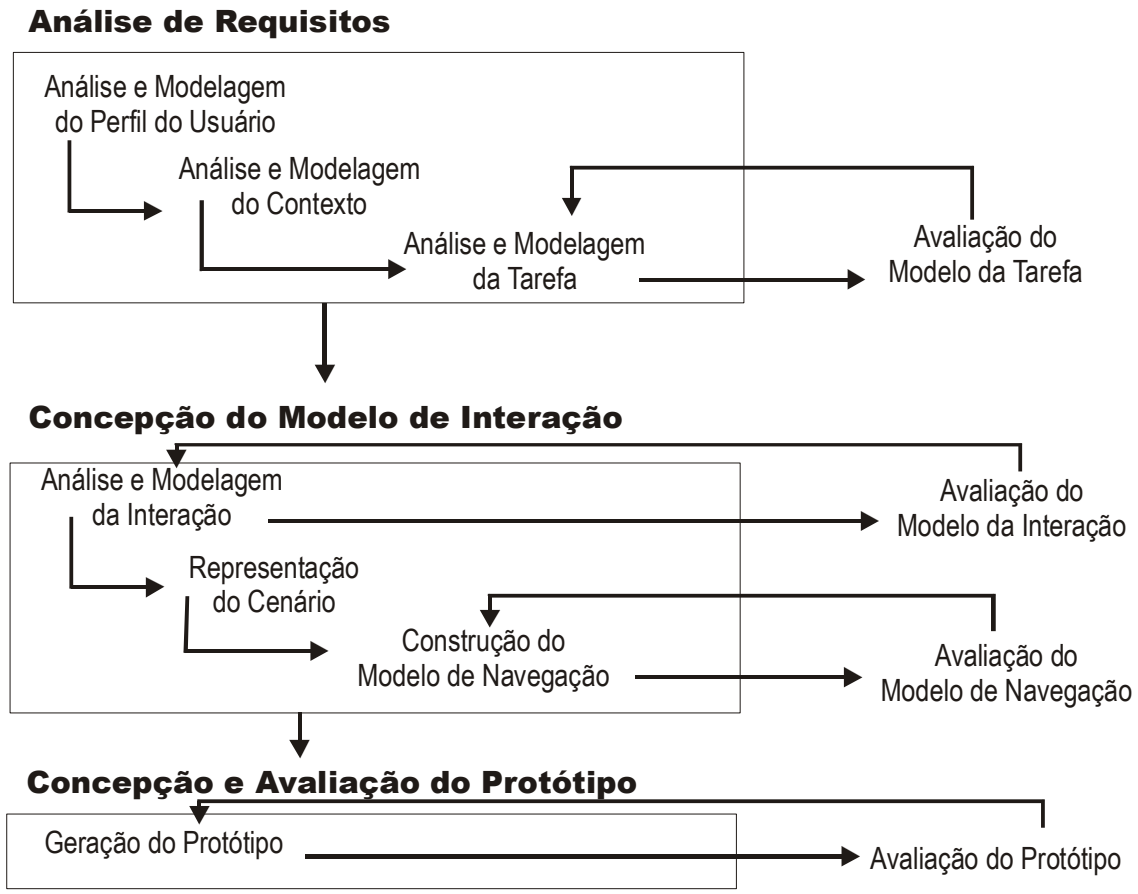

Figura 4: Metodologia MCI [SCH 03].

A seguir, uma breve explicação acerca de cada uma das principais atividades que compõem o método MCI:

- análise das características do usuário, em termos das tarefas que serão realizadas e do conhecimento necessário para realizá-las. Como artefato resultante, tem-se o "perfil do usuário" que contém as características relevantes para a interação;
- construção do modelo da tarefa: tendo em mãos os dados obtidos na etapa anterior, constrói-se o modelo da tarefa baseado no formalismo MAD;
- avaliação da consistência do modelo da tarefa: consiste na verificação da completude da tarefa modelada;
- concepção do modelo de interação: a partir do modelo da tarefa obtido anteriormente e, associando-se em uma tabela os objetos da tarefa com os objetos da interação, chega-se ao modelo da interação, o qual,

posteriormente, receberá a descrição de detalhes das janelas e objetos que as constituem;

- avaliação da consistência e completude do modelo da interação: uma vez gerado o modelo da interação, este é submetido a uma análise ergonômica, objetivando verificar a coerência e a consistência;
- construção do modelo de navegação: utilizando-se do formalismo CPN (Redes de Petri Coloridas), sintetizam-se as situações de transição mais comuns em uma interface com o usuário (situações de diálogo);
- verificação do modelo de navegação: uma vez modelada a navegação e utilizando-se da ferramenta Design/CPN, pode-se verificar se o modelo atende ao conjunto de propriedades de usabilidade definidas para um determinado contexto;
- construção do protótipo: fundamentando-se no modelo de interação e no modelo da navegação, desenvolve-se o protótipo utilizando a ferramenta de prototipação DevGuide;
- validação do protótipo quanto a aspectos ergonômicos do projeto visual: consiste em analisar o protótipo, usando-se a ferramenta FAIUNIX, a fim de obter informações sobre o projeto visual com posterior análise ergonômica;
- preparação do protótipo para validação: geração do seu código a ser validado junto ao usuário;
- validação do protótipo junto ao usuário final: após a geração do código, submete-se o protótipo a sessões de testes com os usuários, de forma a realizar sua validação.

2.6.4 Discussão

De acordo com as metodologias apresentadas, pode-se notar algumas similaridades: todas fazem uso de modelos em suas principais etapas. Essas metodologias fazem uso de modelos de tarefa para dar início à especificação da interface. Além disso, esses modelos são bastante similares: descrevem a tarefa de forma hierárquica, apresentam uma sintaxe gráfica de descrição e proporcionam uma descrição completa e coerente da tarefa (em termos de objetivos, estratégias, procedimentos, ações, objetos da tarefa, etc.).

Vale ressaltar que tanto MCI, quanto MEDITE e MACIA possuem abordagens iterativas e incrementais, ou seja, permitem retorno às etapas anteriores e a saída de uma etapa constitui-se na entrada da etapa seguinte. Esse tipo de abordagem favorece bastante uma integração com um processo de desenvolvimento também iterativo e incremental da ES.

Um diferencial entre MEDITE e MACIA é que, em MACIA, são definidos 3 (três) modelos: um para a descrição das tarefas (extensão de MAD), um para a especificação da interação (Modelo de Interface Conceitual (MIC)) e um outro para arquitetura (Modelo de Agentes). Já em MEDITE, são adotados 2 (dois) modelos: um para descrição das tarefas (MAD*) e outro unificado de interação e de arquitetura (o Modelo EDITOR). No MCI também são definidos dois modelos principais: o modelo da tarefa (MAD) e o modelo da interação.

2.6.4.1 Quadro de Resumo

Com base no que foi discutido, é apresentado, no Quadro 1 abaixo, um resumo comparativo entre as metodologias:

		MODELOS E RECURSOS				
		Modelo da Tarefa	**Modelo da Interação / Navegação**	**Modelo de Arquitetura**	**Fatores Humanos**	**Suporte Computacional**
METODOLOGIAS	**MEDITE**	MAD*	EDITOR (interação)	EDITOR	Regras ergonômicas	iTAOS
	MCI	MAD	Redes de Petri Coloridas (navegação)	-	Regras ergonômicas oriundas de (diretrizes, normas, guias de estilo) usadas na Concepção e na Avaliação	Euterpe Design/CPN FAIwin, FAIunix
	MACIA	Extensão MAD	MIC (interação)	Agentes	Regras ergonômicas	-

Quadro 1: Resumo das metodologias MEDITE, MCI e MACIA.

Uma diferença entre MCI e as demais metodologias citadas reside no fato de que, no MCI, há uma maior explicitação e simplicidade dos passos a serem seguidos para a concepção e avaliação da interface, ao passo que MEDITE e MACIA preocupam-se em explicitar o uso de regras ergonômicas visando um público-alvo que não conheça tais regras.

Essa maior compreensibilidade constituiu um elemento-chave para a escolha do MCI, uma vez que quanto mais simples for o processo – com seus fluxos de atividades, artefatos, ferramentas – maior o entendimento, interesse e aceitação por parte dos engenheiros e desenvolvedores da ES (leigos em usabilidade) que irão utilizar o modelo centrado no usuário.

Em síntese, foi possível elencar algumas características que se destacaram no MCI em relação à MEDITE e MACIA:

- **melhor adaptação e menor esforço:** na ausência de um projetista de interfaces no time de desenvolvimento, o MCI apresenta maior clareza, melhor explicitação e simplicidade das suas atividades. Isso reduz o impacto, favorece a adaptação a um processo da ES e, conseqüentemente, um melhor entendimento por parte dos engenheiros de software e desenvolvedores (em sua maioria, leigos em usabilidade) que irão, de fato, aplicar o método em seus projetos;
- **suporte ferramental mais completo:** um outro ponto positivo diz respeito ao suporte ferramental. O método sugere ferramentas em quase todas as suas fases. No âmbito da IHC, é importante ressaltar aqui que, quando não há suporte ferramental que apóie uma metodologia, toda a responsabilidade da aplicação desta e conseqüente concepção da interface recai sobre o "projetista de interfaces". Assim, apóia-se unicamente nos conhecimentos (empíricos, quando os possui) deste profissional para que se consiga aplicar bem o método e, dessa forma, prover uma interface com maior usabilidade e qualidade;
- **ampla utilização no GIHM**: conforme já foi dito, o MCI é um método do GIHM/UFCG e assistido por uma de minhas orientadoras. Busca-se, com a proposição do trabalho, não somente dar continuidade (extensão) a um trabalho iniciado no desenvolvimento desta metodologia, mas, sobretudo, agregar um maior valor ao integrar o MCI com um processo de desenvolvimento da ES num ciclo de desenvolvimento mais amplo.

Nesse sentido, foi adotado e adaptado, para realização deste trabalho, o Método para Concepção de Interfaces (MCI).

2.7 Considerações finais do capítulo 2

Neste capítulo, foi visto um panorama acerca da utilização das técnicas de usabilidade no atual mercado produtor de software. Discutiu-se não somente as diferentes perspectivas adotadas pelos profissionais das áreas de IHC e ES, mas, sobretudo, a visão tradicional da usabilidade dos muitos engenheiros, projetistas e desenvolvedores de sistemas da ES. Ainda neste capítulo, foram destacados alguns benefícios e os impactos positivos que as técnicas de usabilidade exercem na qualidade e aceitação final de um produto de software. Algumas propostas de integração envolvendo as duas áreas citadas acima também foram apresentadas e examinadas. Finalmente e, com base na análise de algumas metodologias da área de IHC, foi apresentada a metodologia para concepção de interfaces escolhida para a integração e confecção do modelo centrado no usuário.

Capítulo 3

Metodologias Ágeis de Desenvolvimento

3.1 Considerações iniciais

Neste capítulo, serão abordados não apenas o atual mercado produtor de software como também, de maneira mais específica, a boa aceitação que as metodologias ágeis de desenvolvimento de sistemas de software têm encontrado neste cenário. É exposta ainda uma discussão acerca das metodologias tradicionais de desenvolvimento de aplicações: suas características, princípios e qualidades. Algumas metodologias ágeis conhecidas no mercado também serão apresentadas e examinadas neste capítulo. Em seguida, será discutido o processo de desenvolvimento escolhido para incorporar o modelo centrado no usuário.

3.2 Abordagem tradicional de desenvolvimento de sistemas de software

Raccoon [RAC 98], ao abordar o desenvolvimento de software tradicional, afirma que a história da Engenharia de Software (ES) está intimamente ligada às organizações militares norte-americanas. As primeiras metodologias de desenvolvimento de sistemas (que, por sua vez, utilizavam-se do modelo cascata, citado no Capítulo 2) foram criadas por volta da década de 60, através da adaptação de trabalhos realizados pelo Departamento de Defesa dos Estados Unidos (DoD), no desenvolvimento de hardware. O autor expõe ainda que, como o DoD sempre foi um grande e importante cliente, suas características e padrões de desenvolvimento rígidos têm sido seguidos por diversas organizações do setor privado ao longo dos anos.

Embora Raccoon seja informal em boa parte de sua descrição, seu enfoque é interessante no sentido de enunciar a complexidade como uma realidade nas metodologias de desenvolvimento de sistemas tradicionais, e não um problema a ser resolvido no futuro.

Lehman [LEH 88] afirma que, de forma geral, um processo de desenvolvimento de software convencional se caracteriza por capturar ao máximo os requisitos do sistema a ser desenvolvido, colocando-os na forma de diversos procedimentos e documentos para limitar e controlar mudanças. Estes procedimentos e documentos, juntos, têm como objetivo prover uma espécie de "mapa conceitual" do processo de desenvolvimento. A maior parte das atividades existentes neste tipo de processo é baseada em premissas bastante tradicionais da área de ES: (1) uma divisão discreta das atividades do processo de software e (2) foco no gerenciamento, medições detalhadas e registros precisos destas atividades através da geração de documentos.

Assim sendo, a documentação extensiva, a execução rígida de medições e a previsibilidade são atividades fundamentais cultuadas por estas metodologias ao longo de todo o desenvolvimento de um produto. De forma sucinta, essa filosofia da ES reflete uma abordagem baseada nas seguintes características:

- definição documental detalhada de um processo previsível, de seus padrões e procedimentos, para que eles possam ser seguidos por toda a organização de uma forma repetida e bem controlada;
- utilização de ferramentas para auxiliar o processo da engenharia de software;
- produção extensiva de documentação;
- estabelecimento de acordos e contratos entre as partes envolvidas;
- planejamento e controle visando administrar riscos, requisitos, recursos e outras variáveis inerentes ao desenvolvimento.

3.2.1 Discussão

Bollinger *et al.* [BOL 99], ao abordarem os processos tradicionais, afirmam que existe discussão até que ponto modelar e documentar exaustivamente um processo de desenvolvimento de software é relevante. Os autores colocam que certo é que a maior

parte dos modelos tratam de questões essencialmente conceituais da organização e do desenvolvimento.

É claro que um processo de software pode ser visto como um gerador de produtos. Contudo, é importante ter em mente que o produto final, e principal, é o software em si. Neste sentido, Fowler [FOW 03], ao abordar as metodologias tradicionais de desenvolvimento de sistemas (apelidadas pelo autor de metodologias "monumentais"), coloca que estas não são orientadas ao produto (o software), mas ao processo. Logo, a construção precisa de documentos conceituais. Em suas palavras:

> *"Estas metodologias de desenvolvimento [monumentais] existem há muito tempo. Elas não são conhecidas por serem particularmente de sucesso [...] A crítica mais freqüente é que são extremamente burocráticas [...] Como uma reação a estas metodologias, um novo grupo de modelos de desenvolvimento apareceu nos últimos anos [...] Estes novos métodos tentam estabelecer um compromisso útil entre nenhum processo e um processo demasiado, provendo apenas um modelo suficiente para fornecer uma vantagem razoável".*

De acordo com esta visão, na última década, um segmento crescente da comunidade da ES vem defendendo a existência de problemas fundamentais na aplicação sistemática e institucionalizada de processos de software convencionais [BEC 00][DEM 95][BAC 94]. Estes proponentes advogam processos de desenvolvimento mais simplificados, focados não apenas no produto, mas também nas pessoas que compõem o processo.

3.3 Panorama do mercado atual

Na seção anterior, foi visto que a realização de controles precisos, cronogramas rígidos, medição detalhada e documentação demasiada são importantes atividades dos processos convencionais da ES. Porém, seguir fielmente todos esses procedimentos pode não necessariamente assegurar o sucesso de um projeto de software.

De acordo com o Gartner Group [GAR 02], cerca de 70% dos projetos de software fracassam atualmente. E as causas continuam sendo as mesmas: (1) o não cumprimento dos prazos, (2) "estouro" nos orçamentos dos projetos e (3) o não atendimento das necessidades do cliente. Parece que, na prática, caminha-se na mesma direção daquela dos anos 80. E a exigência por software no ambiente atual é bem maior que naquela época; pois as necessidades dos negócios existentes estão inseridas em um contexto norteado por fortes pressões introduzidas por um mercado turbulento, altamente competitivo e, principalmente, em constante mudança.

É nesse panorama que as chamadas "metodologias ágeis" se inserem. A necessidade de processos minimalistas - com pouca burocracia envolvida no desenvolvimento - e de equipes ágeis de software que sejam adaptáveis, capazes de reagir rapidamente a mudanças e de agregar valor ao cliente num ritmo constante é muito grande. Segundo Beck [BEC 00], a abordagem tradicional de desenvolvimento de software é pouco aderente ao contexto dinâmico e imprevisível do atual mercado e suas tecnologias. Assim sendo, as metodologias ágeis buscam uma abordagem adequada a esta realidade: focar pessoas ao invés de processos, gerar rapidamente valor para o cliente e, sobretudo, "abraçar" as freqüentes mudanças de requisitos.

O Gartner Group afirma ainda que, nos próximos cinco anos, as organizações irão sofrer alterações radicais na maneira como desenvolvem aplicações. A previsão é que, até 2005, apenas um terço das organizações irá desenvolver software de forma tradicional. Projetos grandes, com processos de desenvolvimento "pesados" (*heavyweight processes*) e equipes compostas de dezenas ou centenas de desenvolvedores já estão cedendo espaço para equipes menores e ágeis, com alta adaptabilidade e capacidade de entregar resultados ao cliente em um curto espaço de tempo.

3.4 Metodologias ágeis

Nesta seção, são apresentadas algumas metodologias de desenvolvimento de aplicações que possuem seu foco em aspectos práticos relacionados ao desenvolvimento de software. Estas metodologias modernas de concepção de sistemas propõem soluções práticas simples para problemas muito comuns ao longo do desenvolvimento.

Finalmente, elas são também chamadas de processos "leves" (*lightweight processes*) ou processos ágeis.

3.4.1 Ágil versus Tradicional

A constante necessidade de maior agilidade e adaptabilidade em ambientes reais de desenvolvimento de aplicações tem alavancado o surgimento e amadurecimento da maioria das modernas metodologias de desenvolvimento de sistemas.

Ao longo dos anos, muito se tem ouvido falar na comunidade de ES sobre métodos como *eXtreme Programming* (XP) [BEC 00], SCRUM [SUT 01][SCW 95], *Feature Driven Development* (FDD) [FDD 03], *Dynamic System Development Method* (DSDM) [STA 97][DSD 03], etc. Esses princípios e práticas se propõem a tratar os requerimentos de software instáveis e incertos de uma maneira mais apropriada que as metodologias tradicionais. E estas metodologias, em particular, são relativamente recentes: o DSDM, por exemplo, foi proposto em 1994 na forma de um consórcio de empresas britânicas que precisavam desenvolver rapidamente aplicações de forma iterativa; o XP foi concebido em um projeto na *Chrysler*, em 1997; o SCRUM foi lançado em 1995 e o FDD em um grande projeto para um banco de Cingapura, também em 1997.

Apesar do pouco tempo, o movimento ágil já possui uma força expressiva nos Estados Unidos, Europa e Japão não apenas pelos resultados obtidos, mas também pelo envolvimento de profissionais reconhecidos no âmbito do desenvolvimento de software como *Kent Beck*, *Tom DeMarco*, *Jim Highsmith* e *Martin Fowler*. Inúmeras publicações têm dado a merecida atenção ao tema. A *Software Development* [SWD 03], importante referência para projetistas e desenvolvedores da ES em todo o mundo, por exemplo, dedica uma coluna mensal ao tema, escrita por *Scott Ambler*. Além disso, é grande a freqüência deste assunto em congressos internacionais e publicações acadêmicas.

Não obstante a variedade de metodologias classificadas como ágeis, todas compartilham diversas características que, por sua vez, são distintas da abordagem tradicional. Estes processos não estão amarrados às questões filosóficas profundas da ES, e sim, são organizados e centrados em aspectos relacionados a uma rápida produtividade sem que haja perda de qualidade.

Wells [WEL 02] afirma que, enquanto as metodologias convencionais buscam a previsibilidade, as ágeis buscam capacidade de adaptação; enquanto metodologias tradicionais são orientadas a processos, os ambientes ágeis de desenvolvimento de software estão centrados nos desenvolvedores e seus clientes. A respeito deste assunto, é importante lembrar que as pessoas constituem um fator-chave no sucesso de qualquer projeto de software. Fowler [FOW 03] nos diz que um processo descrito não garante o êxito de um projeto, mas a colaboração entre os valores individuais das pessoas com o objetivo de entregar valor, sim. O autor indaga que, como as pessoas são "componentes não-lineares de primeira ordem no desenvolvimento de software", por que tratá-las apenas como um papel a ser desempenhado dentro do processo? Ainda, Boehm *apud* Glass [GLA 01] coloca que a qualidade de um time de desenvolvimento - em particular dos programadores - tem uma influência maior no sucesso de um projeto que os processos ou mesmo suas ferramentas.

Assim sendo, a comunicação entre as pessoas que compõem o projeto aparece como um dos principais pilares de sustentação do desenvolvimento ágil de aplicações.

Um outro ponto importante diz respeito à geração de documentos. Como foi dito anteriormente, as metodologias tradicionais, normalmente, são orientadas à documentação excessiva e não à comunicação e interação entre pessoas. Nestas, alguém realiza uma atividade, produz um documento e passa adiante. Outra pessoa lê aquele documento, realiza outra atividade exatamente conforme descrito pela metodologia, produz um outro documento e passa adiante. É um esquema "linha de produção", que valoriza a burocracia e, em alguns casos, desvaloriza o pensamento criativo das pessoas.

Todavia, o software deve ser elaborado através de um processo criativo, onde o conhecimento deve ser um fator crucial para o sucesso do projeto. A equipe precisa não somente conhecer as necessidades do usuário, mas: (1) perceber os conhecimentos e a experiência deste último, (2) auxiliá-lo a compreender suas necessidades e (3) transformá-las em um produto de software, adequadamente. Afinal, qual objetivo uma metodologia deve ter: produzir documentação ou **produzir conhecimento**?

É importante ressaltar, entretanto, que isso não significa que em ambientes ágeis não se produz documentação. Apenas deixa de existir a filosofia "tudo deve ser documentado" para dar lugar a idéia de "documentar apenas o que realmente precisa ser

documentado". Para cada projeto e contexto é exigido um tipo de documentação. Então, melhor que documentar tudo é ter bom senso em relação ao exagero.

Não apenas devido ao fato de, normalmente, grande parte das atividades dos atuais processos de desenvolvimento de software serem de natureza iterativa, mas para antever e mitigar eventuais riscos durante todo o projeto é que as metodologias ágeis operam com iterações (ou janelas de tempo) menores. Dessa forma, rápidas e freqüentes *releases* de porções do software final ajudam a descobrir, por exemplo, problemas decorrentes de novas tecnologias antes que um grande investimento tenha sido feito.

O foco na colaboração também cria a oportunidade de uma relação diferente entre cliente e desenvolvedor: a de **parceria**. Neste momento, contratos fechados e detalhados, especificações formais de requisitos e gerenciamentos rígidos de mudanças abrem espaço para a flexibilidade. Todas as metodologias ágeis trabalham com *releases* do produto ocorrendo em poucas semanas, o que promove não só uma geração constante de valor para o cliente, mas trata mudanças nos requisitos como oportunidades para este último (e não como variáveis indesejáveis a serem rigidamente controladas). Glass [GLA 01] coloca que, entre os profissionais de software, há uma máxima: *"Só existem duas certezas na vida: a morte e que os requisitos do sistema mudarão"*. Em um ambiente ágil, portanto, as mudanças são bem vindas. Outrossim, sendo capaz de responder às mudanças, muito do tempo gasto tentando planejar e controlar o incerto pode ser utilizado para desenvolver o produto.

Em síntese, ao se utilizar de práticas que promovam não apenas a sinergia e comunicação entre as pessoas envolvidas no desenvolvimento, mas também a colaboração e a distribuição constante de conhecimento, a necessidade de controle e documentação excessiva em um processo pode diminuir abruptamente.

3.4.2 O Movimento Ágil

Com base na exposição feita na seção anterior, é possível resumir estes princípios ágeis através do Manifesto Ágil [MAN 03], publicado em 2001 por dezessete conceituados metodologistas da área de software. Esse manifesto declara que, em ambientes ágeis, se preza:

- indivíduos e interações **mais que** processos e ferramentas;
- software funcionando **mais que** documentação extensiva e detalhada;
- colaboração do cliente **mais que** negociações contratuais;
- responder às mudanças **mais que** seguir um plano.

É importante enfatizar que o manifesto reconhece a existência de valor nos itens da direita, porém, os itens da esquerda são os que, segundo seus idealizadores, fazem a diferença em relação ao sucesso na entrega eficaz de valor constante ao cliente.

A discussão do manifesto teve seu foco nos valores do movimento ágil. Entretanto, o manifesto expõe ainda alguns princípios fundamentais:

- *"nossa maior prioridade é satisfazer o cliente através da entrega rápida e contínua de software valioso";*
- *"mudanças nos requerimentos do sistema são bem vindas, mesmo que de última hora";*
- *"forneça releases de pequenas porções de software freqüentemente, em poucas semanas";*
- *"cliente e desenvolvedor devem trabalhar juntos ao longo de todo o projeto";*
- *"faça projetos com pessoas motivadas, disponibilize o ambiente e o suporte necessário, então confie que elas possam fazer o trabalho";*
- *"o método mais eficiente para colher informações sobre o projeto é através de conversas face-a-face com o cliente";*
- *"a principal medida de progresso de um projeto é trabalhar em cima de software e não de documentação";*
- *"processos ágeis promovem desenvolvimento sustentável. Clientes e desenvolvedores devem descobrir seu ritmo de trabalho e mantê-lo constantemente";*
- *"a atenção contínua às boas práticas e a um bom design promovem agilidade no projeto";*
- *"simplicidade ao longo do projeto é essencial";*
- *"as melhores arquiteturas, requerimentos e design emergem de times auto-organizados";*

- *"em intervalos regulares, o time de desenvolvimento refletirá como se tornar mais efetivo. Então poderá ajustar seu comportamento conforme as necessidades, adequadamente".*

Serão apresentadas, nas próximas seções, algumas metodologias de desenvolvimento de software norteadas por estas diretrizes; com e*Xtreme Programming* sendo abordada primeiro. Suas características, valores e princípios serão discutidos. Mais adiante, outras duas metodologias: SCRUM e Crystal/Clear também serão apresentadas e discutidas.

3.4.3 *eXtreme Programming* (XP)

Segundo Beck [BEC 00], um dos principais problemas relacionados ao desenvolvimento de software é o risco. Atrasos no cronograma, projetos cancelados devido a esses atrasos, sistemas tornando-se obsoletos, alta taxa de defeitos, mudanças de requisitos e saída de importantes membros da equipe de desenvolvimento são exemplos de riscos que podem resultar no fracasso de um projeto de software. O *eXtreme Programming* (XP) é uma metodologia da ES que vem ganhando cada vez mais adeptos pela simplicidade de suas práticas e pelos resultados surpreendentes que vem demonstrando em lidar com estes tipos de riscos.

3.4.3.1 Definição

XP é um novo modelo para o processo de desenvolvimento de software que visa alcançar duas metas almejadas pela indústria de tecnologia da informação: desenvolvimento rápido e consistente com as reais necessidades do cliente e fácil manutenibilidade, permitindo que o software seja modificado à medida que as necessidades do negócio se alterem ou ampliem. Apesar do termo "programação extrema", XP reflete um conjunto bem definido de regras que vem conquistando um grande número de adeptos, particularmente entre os programadores de tecnologias orientadas a objetos, como os desenvolvedores *Java* [RAD+ 01].

Comunicação, simplicidade, *feedback* e coragem são os quatro lemas adotados pelos seguidores de XP [JEF 02], que correspondem a quatro dimensões nas quais os projetos podem ser melhorados. XP oferece condições para que os desenvolvedores

possam responder de forma confiável às alterações de requisitos propostas pelos clientes, mesmo em estágios finais do ciclo de vida do processo.

De acordo com Maurer [MAU 02], XP tem se firmado como o mais conhecido e bem-sucedido entre os modernos processos ágeis. O termo *extreme* enfatiza o uso extremo de práticas que se mostram funcionais, como: revisões de código, testes, simplicidade e ciclos curtos e iterativos [BEC 00].

Ao contrário das metodologias tradicionais da ES, o XP não gera uma grande quantidade de artefatos nem uma seqüência rígida de etapas a serem cumpridas. XP prega simplicidade e um conjunto de práticas focadas em resultados concretos. Trata-se de um processo leve que possui ênfase numa rápida e elevada produtividade de código – isso sem comprometer a qualidade ou flexibilidade do processo. Tal flexibilidade constitui um grande atrativo para a confecção do modelo centrado no usuário proposto.

3.4.3.2 Histórico

eXtreme Programming começou a ser desenvolvido em 1996 por Kent Beck no departamento de computação da montadora de automóveis DaimlerChrysler, e possui muitas diferenças em relação aos outros processos de software, podendo ser aplicado a cenários de desenvolvimento com altos riscos e requisitos dinâmicos.

Na Chrysler, um dos maiores fabricantes de automóveis do mundo, um sistema denominado "C3" seria o responsável pelo pagamento de 86.000 funcionários - entre horistas, diaristas e mensalistas - parte deles com direito a diferentes bonificações. Tratava-se de um sistema grande e complexo, cujo desenvolvimento foi iniciado e, alguns meses depois, abandonado e oficialmente declarado como um fracasso. Mais ou menos um ano depois, o projeto foi completamente reelaborado com a metodologia XP. Em menos de quatro meses de projeto, não apenas o pagamento de 10.000 mensalistas já estava em produção, mas também o de 16.000 executivos estava sendo migrado para o novo sistema produzido.

3.4.3.3 Valores, princípios e requisitos básicos de XP

XP segue um conjunto de valores, princípios e requisitos básicos que visam alcançar eficiência e efetividade no desenvolvimento de software [BEC 00]. Conforme

foi dito anteriormente, os valores são quatro: comunicação, simplicidade, *feedback* e coragem. Nestes valores, estão fundamentados alguns princípios básicos: *feedback* rápido, simplicidade, mudanças incrementais e apenas quando necessárias e trabalho com qualidade. Por fim, nesses princípios se baseiam os doze requisitos básicos adotados por XP [MAU 02]:

1) ***Planning Process, também chamado de Planning Game****:* o processo de planejamento de XP permite que o "cliente XP" defina o valor de negócio dos recursos desejados e utilize estimativas de custo fornecidas pelos programadores para decidir o que é necessário ser feito e o que pode ser adiado.
2) ***Pequenos lançamentos:*** as equipes XP colocam um sistema simples em produção com antecedência, e o atualizam freqüentemente em ciclos bastante curtos.
3) ***Metáforas do sistema:*** as equipes XP utilizam um "sistema de nomes" e uma descrição do sistema sem a utilização de termos técnicos para guiar o desenvolvimento e, sobretudo, favorecer a comunicação com o cliente.
4) ***Projeto simples:*** um programa construído através do método XP deve ser o mais simples possível satisfazendo os atuais requisitos, sem a preocupação de atender outros que surgirão no futuro. O foco está em prover valor de negócio ao cliente o mais rapidamente possível.
5) ***Testes:*** as equipes XP focalizam a validação do software durante todo o processo. Os programadores escrevem primeiro os testes, e só então desenvolvem o software que atenda aos requisitos destes testes. Além disso, os clientes provêem testes de aceitação para ter certeza de que os recursos necessários estão sendo fornecidos.
6) ***Refatoramento (ou reestruturação de código):*** as equipes XP procuram aperfeiçoar o código do sistema durante todo o desenvolvimento, mantendo sua clareza, eliminando ambigüidade, mantendo-o simples, porém completo.
7) ***Programação em pares:*** os programadores XP produzem o código em pares, ou seja, dois programadores trabalhando juntos na mesma máquina. Muitos experimentos têm mostrado que a programação em duplas produz software de melhor qualidade com um custo similar ou menor do que o produzido por programadores trabalhando individualmente.

8) ***Propriedade coletiva:*** todo o código gerado pertence a todos os programadores. Essa característica permite que a equipe trabalhe a toda velocidade, uma vez que as alterações podem ser feitas sem atrasos, pois todos têm liberdade para fazê-las. Além disso, o impacto de uma eventual saída de um membro da equipe será menor, uma vez que o código gerado por este é conhecido por todos.
9) ***Integração contínua:*** as equipes XP integram pequenas porções de código e, assim, constroem o sistema de software várias vezes por dia. Isso mantém todos os programadores em sintonia e possibilita um progresso rápido da aplicação.
10) ***40 horas de trabalho semanal:*** programadores exaustos cometem mais erros. As equipes XP não trabalham por um tempo que exceda este limite, mantendo-se, assim, mais eficazes.
11) ***Cliente dedicado:*** um projeto XP é conduzido por um indivíduo dedicado (um cliente, ou representante deste), que determina os requisitos, atribui as prioridades e responde às dúvidas dos programadores (relacionadas aos requisitos). Essa prática melhora a comunicação e gera menor documentação, o que, em geral, é uma das partes mais onerosas num projeto de software.
12) ***Código padrão:*** para que uma equipe trabalhe em duplas de forma efetiva e compartilhe a propriedade de todo o código, todos os programadores precisam escrever da mesma forma (ou estilo), com regras que assegurem e favoreçam a clareza e fácil entendimento de partes do código.

3.4.3.4 Ciclo de Vida e Fases do Processo

O ciclo de vida XP é uma das abordagens mais discutidas por diversos grupos que adotam essa metodologia. A Figura 5 representa, de forma simplificada, o ciclo de vida do processo XP:

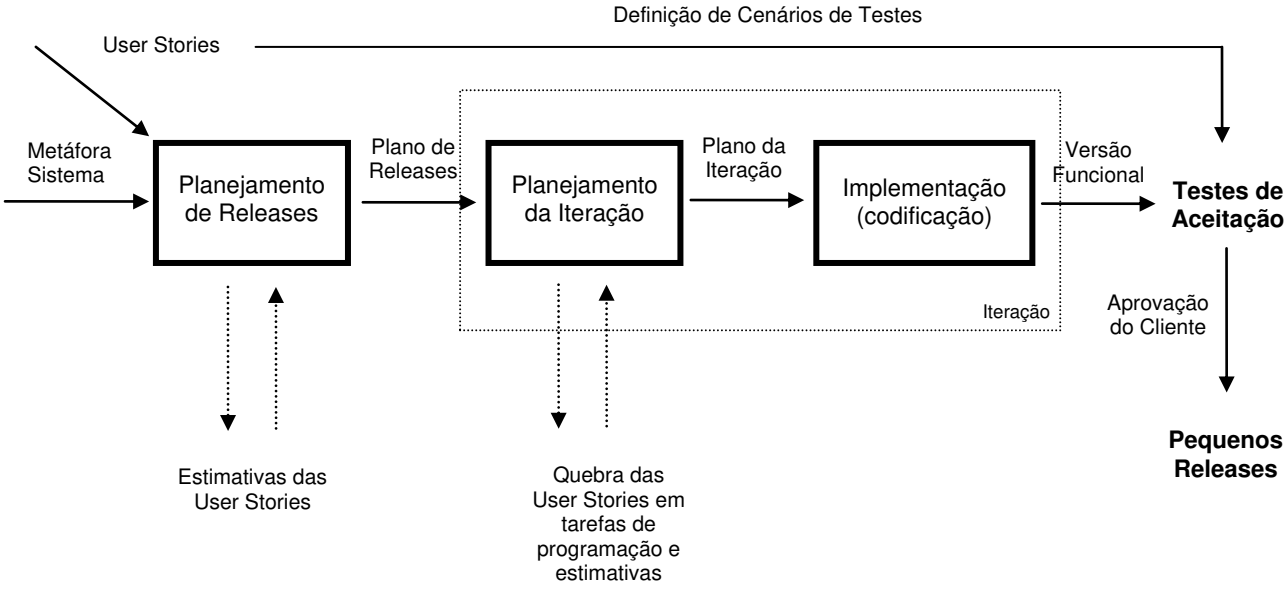

Figura 5: Ciclo de vida simplificado de XP [WEL 02].

O ciclo de vida XP é bastante curto e, à primeira vista, difere dos padrões dos modelos de processo convencionais. No entanto, esta abordagem pode fazer sentido em um ambiente onde as mudanças de requisitos do sistema são fatores dominantes.

De acordo com Oshiro [OSH 03], podemos simplificar todas as premissas expostas por XP agrupando-as em quatro fases gerais: (1) Planejamento, (2) Testes, (3) Codificação e (4) Projeto. A Tabela 2 representa, detalhadamente, as atividades presentes em cada uma destas fases do ciclo de vida XP:

Fases	Planejamento	Testes	Codificação	Projeto
Atividades	*User Stories*	Testes de Unidade	Programação em pares	Simplicidade
	Lançamento do produto (planejamento de *releases*)	Testes de Aceitação	Estratégias de integração contínua	Metáfora do sistema
	Métricas		Código coletivo	Cartões CRC
	Planejamento das iterações			Soluções pontuais
	Standup Meetings			Não adicionar funcionalidade antes do necessário
				Refatoramento

Tabela 2: Fases gerais do processo XP e suas atividades [OSH 03].

Na fase de planejamento, os requisitos do cliente são cuidadosamente coletados à medida que fornecidos. A seguir, os testes são elaborados a partir das especificações do cliente e a fase de codificação é realizada visando atender a esses testes. Existe uma relação próxima e contínua entre as fases de teste e codificação, como veremos mais adiante. Por fim, o sistema é novamente projetado (ou reconstruído) à medida que novas funcionalidades são incorporadas.

A seguir, serão descritas as quatro fases do processo XP, bem como os conceitos e os requisitos básicos relacionados a elas que, de forma conjunta, procuram viabilizar o desenvolvimento de software em ambientes em contínua evolução.

FASE DE PLANEJAMENTO: a fase de planejamento consiste em estimar diversos fatores que podem afetar o desenvolvimento do software. Algumas das tarefas do planejamento incluem: definir escopo e prioridade do projeto, estimar custos e cronogramas e criar um plano para a entrega de uma nova versão do produto. Uma diferença entre o XP e a maioria dos modelos de processo convencionais é que XP não define a especificação formal e completa de requisitos. Os conceitos e requisitos necessários à fase de planejamento compreendem: (a) *user stories*, (b) lançamento do produto (ou plano de *releases*), (c) métricas, (d) planejamento das iterações e (e) *standup meetings*.

(a)User Stories – definição incremental dos requisitos do sistema

Fazendo uma analogia com o papel dos "casos de uso" (*use cases*) [JAC 94], encontrados no Processo Unificado (*Unified Process*) [JAC 99], XP também provê meios de levantar os requisitos funcionais do sistema: ele o faz através das chamadas "estórias do usuário" (*user stories*). Estas, têm a finalidade de não apenas declarar os requisitos funcionais do sistema a ser desenvolvido, mas também criar estimativas de tempo que irão integrar o planejamento dos *releases*.

User stories são escritas pelo cliente do sistema, utilizando uma linguagem natural, ao invés de um vocabulário técnico, visando especificar as tarefas que o sistema precisa realizar. Trata-se de um processo semelhante a cenários de uso, mas não se limita a descrever a interface do programa. As *stories* sempre focam a necessidade do usuário e, por serem escritas em uma linguagem natural, permitem a compreensão independentemente de uma tecnologia específica, projeto de dados ou algoritmo.

As *user stories* são também utilizadas para a criação dos testes de aceitação que verificam se estas foram corretamente implementadas. A única diferença entre as *user stories* e o documento de requisitos de algumas metodologias tradicionais é que o último apresenta maior grau de detalhamento. As *user stories* devem possuir detalhes suficientes para estimar, com baixo risco, quanto tempo levará sua implementação. Na fase de implementação, os requisitos serão mais bem explicitados através da interação dos desenvolvedores com os clientes.

Para cada *user story*, a equipe de desenvolvedores estima, inicialmente, qual seria o tempo ideal para o desenvolvimento. O tempo ideal é aquele necessário para codificar a *story* supondo que não existam distrações, nenhuma outra tarefa a ser feita e os implementadores saibam exatamente como codificar o problema. Se o tempo ideal resultante for menor que uma semana, essa *user story* é considerada como um detalhe, e deve ser combinada à outra *user story* de escopo mais amplo. Se o tempo ideal for maior do que três semanas, deve-se analisar a possibilidade de subdividir a *story* em tarefas mais específicas.

(b) Lançamento do produto - criação do plano para a entrega (ou plano de releases) de uma nova versão do produto

Uma reunião para o planejamento da entrega de uma nova versão do produto define as características gerais do software. Este plano irá subsidiar os planos de iteração para desenvolvimento das versões do produto.

O processo de desenvolvimento XP é incremental e, a cada iteração do software, um conjunto de *user stories* é implementado. Durante a reunião para criação do plano de entrega do software, um conjunto de *user stories* semelhantes são priorizadas e agrupadas. Depois, determina-se o que será feito em cada iteração do ciclo, através da utilização de cartões com as *user stories* agora identificadas. Estes cartões são organizados em ordem de implementação para as próximas versões do software. O cronograma para a entrega de uma nova versão do software é baseado no escopo de *user stories* definidas para a implementação no ciclo de iteração. Os ciclos de desenvolvimento são curtos com entregas freqüentes de novas versões do software.

O plano de entrega permite que as decisões sejam tomadas de forma a assegurar a viabilidade técnica e comercial do projeto. As regras para a criação do plano de entrega envolvem métodos de negociação de cronograma, que permitem o comprometimento das áreas comercial e técnica da empresa.

O objetivo principal da reunião para a criação do plano de entrega do produto é permitir ao time de desenvolvimento estimar cada *user story* em termos de tempo ideal de programação (em semanas). A única tarefa extra, além da codificação existente no tempo ótimo de programação, é a seção de testes. Se as estimativas para implementação de *user stories* não agradarem a gerência ou a área comercial, ao invés de subestimar o tempo de implementação, deve-se diminuir o escopo de *user stories* que serão entregues na próxima versão do software.

Pode-se definir a velocidade do projeto por tempo ou por escopo. A estimativa por tempo considera o número de *user stories* que podem ser implementadas em um dado período, enquanto a estimativa por escopo analisa o tempo necessário para implementação de um dado conjunto de *stories*. Ambas as formas de estimativa consideram os recursos humanos disponíveis que podem ser alocados de acordo com as restrições de tempo e escopo da versão.

A gerência ou área comercial da empresa pode definir apenas duas das três variáveis de projeto: tempo, escopo e recursos humanos. A variável restante sempre será ditada pela equipe de desenvolvimento de forma a viabilizar a entrega da próxima versão.

(c) Métricas: mensurabilidade da velocidade do projeto

Uma estimativa inicial da velocidade do projeto pode ser obtida através do *fator de carga*. Este pode ser calculado através da divisão dos dias úteis disponíveis para completar uma tarefa pela estimativa de tempo ideal restante. Ou seja, trata-se da comparação do tempo estimado com o tempo realizado. Se o fator de carga variar muito durante um ciclo de iteração para finalização de uma versão do software, uma nova reunião de planejamento de entrega do produto é realizada com a finalidade de reavaliar o cronograma e o escopo comprometidos na versão. Uma desvantagem do fator de carga é que, devido às características do processo de implementação, ele não pode ser usado como base histórica para estimativas.

Outro fator indicativo da velocidade que deve ser considerado durante a fase de implementação é a quantidade de *user stories* implementadas até o momento na iteração. Este fator simples pode ser utilizado para estimar quantas *user stories* serão implementadas em um dado prazo.

A quantidade de *user stories* implementadas durante um ciclo de iteração é utilizada nas posteriores reuniões de planejamento de entrega da versão de software para estimar o número máximo de *user stories* que podem ser alocadas no próximo ciclo de iteração.

(d) Planejamento das iterações de desenvolvimento

Uma reunião de planejamento é realizada no início de cada iteração. Nessa reunião, são selecionadas as *user stories* mais importantes (priorizadas pelo cliente) para a versão, além dos testes de aceitação que apresentaram problemas no ciclo anterior.

As *user stories* são divididas (ou "quebradas") em tarefas de programação que devem ter tempo ideal de codificação de 1 a 3 dias. Tarefas menores que 1 dia podem ser agrupadas e maiores que 3 dias podem ser quebradas. O fator de velocidade do

projeto é utilizado para verificar se a quantidade de tarefas condiz com o tempo estimado para aquela iteração. Caso se detecte uma sobrecarga na quantidade de tarefas, o cliente é contatado para definição das tarefas que serão adiadas para a próxima iteração. Da mesma forma, caso se detecte superdimensionamento do tempo, novas *user stories* podem ser aceitas (e, então, quebradas) para o ciclo de iteração.

(e) Stand up meetings - acompanhamento do projeto através de reuniões diárias de status

As reuniões semanais de toda a equipe de desenvolvimento para discussões gerais sobre o projeto geralmente são consideradas uma tarefa complexa que consome muito tempo e, muitas vezes, se mostra improdutiva. Uma alternativa proposta por XP são os *stand up meetings*.

Os *stand up meetings* são realizados todos os dias, em um dado horário, para discutir problemas e soluções, além de orientar o time para o que deve ser feito. Essa abordagem evita a complexidade de estabelecer um horário durante um dia da semana no qual toda a equipe de desenvolvimento possa comparecer.

Essas reuniões diárias de *status* contribuem para o conhecimento geral do projeto por todos os membros da equipe. Problemas específicos podem ser discutidos em reuniões separadas, onde compareçam apenas as pessoas envolvidas com o problema em questão. Tais reuniões podem ter caráter prático e informal com o líder de projeto e alguns desenvolvedores, revisando certo código em frente a um micro ou projetando soluções em papel ou em uma lousa.

FASE DE TESTE: um dos doze requisitos básicos de XP, conforme mencionado na seção anterior, é o de escrever testes antes do código. Os testes em XP são divididos em duas categorias: (a) testes unitários e (b) testes de aceitação (também chamados de testes funcionais) [FOW+ 03]. As atividades de testes são realizadas durante todo o processo de desenvolvimento e o código é construído com o propósito de satisfazer os resultados esperados. À medida que um novo código é adicionado, novos testes devem ser realizados para assegurar que não ocorram impactos negativos.

(a) Testes unitários (ou também chamados de testes de unidade)

Testes unitários são, em geral, escritos pelos desenvolvedores e têm a finalidade de testar uma classe individual ou um pequeno grupo de classes.

Os testes unitários constituem um dos elementos chave em XP [WEL 02], pois são criados antes do código e armazenados em um repositório junto ao código que será testado. Um código que não possua seu respectivo teste unitário não deve ser liberado, pois cada parte do sistema com possibilidade de falha precisa ser verificada. Além disso, a criação de testes unitários antes do código provê uma melhor compreensão dos requisitos e direciona o foco dos desenvolvedores.

Um fator importante para a utilização de testes unitários, especialmente se estes forem automatizados, é a economia de tempo que pode proporcionar ao identificar e oferecer proteção contra erros. É relevante considerar que descobrir todos os problemas que podem ocorrer durante o desenvolvimento consome uma grande quantidade de tempo, tornando-se necessário que um conjunto completo de testes unitários esteja disponível logo no início, e não apenas nos estágios finais do projeto.

Nesta etapa, os desenvolvedores criam e executam testes unitários toda vez que um código é escrito ou modificado. Consertar pequenos problemas assim que são encontrados, em geral, leva menos tempo do que consertar grandes problemas a poucas horas do prazo final. Outro benefício dos testes unitários automatizados é a possibilidade de combinar um conjunto de alterações com a última versão liberada e liberar novas versões em um curto espaço de tempo.

(b) Testes de Aceitação

Já os testes de aceitação são usualmente escritos pelos próprios clientes, usuários finais ou por uma equipe de teste externa (com a ajuda dos desenvolvedores) e têm a finalidade de testar todo o sistema, de ponta-a-ponta.

Os testes de aceitação constituem uma das principais diferenças entre o XP e os processos de desenvolvimento tradicionais. Em geral, no XP, são escritos pelos clientes ou usuários finais através das *user stories* [WEL 02], com a assistência de um indivíduo da equipe responsável por testar o software. Durante uma iteração, as *user stories* selecionadas durante a reunião de planejamento de iteração serão traduzidas em testes de aceitação. Uma *user story* pode ter um ou mais testes de aceitação para assegurar que

a sua funcionalidade esteja de acordo com a especificação. Após a implementação da *user story*, o cliente especifica cenários para serem testados, e ela não é considerada completa até que tenha passado por esses testes de aceitação. Isso significa que novos testes de aceitação devem ser criados a cada iteração, ou a equipe de desenvolvimento não deverá reportar progresso.

Testes de aceitação são testes de sistema de caixa preta e cada um deles representa algum resultado esperado. Testes de aceitação também são usados como testes de regressão, anteriores à liberação de uma versão do software. Nesse contexto, a Garantia de Qualidade (GQ) é uma parte essencial do processo XP. Em alguns projetos, GQ é realizada por um grupo especializado, enquanto que, em outros, a GQ é integrada à própria equipe de desenvolvimento. Em ambos os casos, XP requer que o desenvolvimento tenha uma relação muito próxima com GQ.

Dentre as principais metas dos testes de aceitação estão: fornecer aos clientes, gerentes e desenvolvedores a confiança de que o produto inteiro está progredindo na direção certa e checar cada incremento no ciclo XP para verificar se o valor de negócio está presente [CRI 00]. Os testes de aceitação, que são responsabilidade do testador e do cliente, são testes ponta-a-ponta sob a perspectiva do cliente, e não testes que verificam cada caminho possível no código (essa é a finalidade dos testes unitários).

FASE DE CODIFICAÇÃO: essa fase do processo está focada nos métodos para a codificação dos módulos que compõem o projeto. Em projetos de XP, a qualidade do código é muito importante. Para tentar manter esta qualidade, algumas práticas são utilizadas, como: (a) programação em duplas, (b) integração contínua e (c) código coletivo. Além disso, a participação contínua do cliente é indispensável para que a equipe de desenvolvimento faça um bom trabalho.

Uma outra forma de manter a boa qualidade da codificação é usar padrões de codificação definidos e não permitir que as equipes de desenvolvimento possam fazer horas extras visando terminar o projeto dentro do prazo, pois os estudiosos da área acreditam que horas extras apenas diminuem a motivação dos desenvolvedores e, conseqüentemente, o rendimento e a qualidade do código produzido.

A seguir serão apresentados os principais conceitos e requisitos referentes à fase de codificação:

(a) Programação em pares

O objetivo da programação em pares é reduzir as chances de se produzir um código ruim, encorajar o espírito de colaboração entre os programadores e diminuir os riscos gerados quando uma única pessoa tem o conhecimento de partes do código, assegurando que todos os elementos da equipe de desenvolvimento tenham a visão completa do projeto.

(b) Integração contínua

Em muitas equipes de desenvolvimento, ao terminar a codificação de um novo módulo, os desenvolvedores fazem alguns testes e já o integram ao projeto. Como essa integração pode ser feita de forma paralela, códigos que nunca foram testados juntos acabam sendo combinados, causando assim numerosos problemas.

A solução mais simples e geralmente adotada é a integração estritamente seqüencial feita pelos próprios desenvolvedores. Essa opção exige que os códigos sejam armazenados em repositórios protegidos, mantendo, assim, a integridade das informações após alterações. Com uma integração contínua dos módulos, é possível identificar facilmente os problemas de compatibilidade nas diferentes versões geradas pelas várias equipes depois da inclusão ou modificação de novos módulos.

A integração é o tipo de atividade "pague agora para não pagar mais depois". Ou seja, se os desenvolvedores integrarem gradativamente pequenas partes novas ao projeto, será mais fácil e rápido a integração do projeto todo; caso contrário o projeto acabará gastando muito tempo depois do seu término para realizar a integração de grandes módulos, podendo ocasionar atraso na entrega do projeto.

(c) Código Coletivo – o código pertence a todos

O código coletivo (*code ownership*) encoraja todos a contribuir com novas idéias em todas as partes do projeto. Qualquer desenvolvedor pode mudar qualquer parte do código para adicionar novas funcionalidades, consertar erros, ou refazê-lo.

Contudo, para que esse tipo de trabalho funcione, é necessário um controle rígido sobre quem executa as mudanças e quando essas mudanças são feitas, garantindo, assim, a integridade do código que está armazenado no repositório. Isto pode ser feito exigindo-se que, quando um dos desenvolvedores alterar um determinado módulo, testes unitários para esse módulo sejam criados enquanto as alterações estiverem sendo feitas. Após o término das alterações no módulo, todos os testes unitários são verificados para se ter certeza de que o código está de acordo. Se isso acontecer, ele pode ser liberado e integrado ao repositório.

FASE DE PROJETO: em XP, o projeto é responsabilidade de toda a equipe e não de apenas uma pessoa. Desta forma, todos os membros da equipe podem cooperar para a elaboração de um projeto com resultado superior do que o melhor dos projetistas poderia, individualmente, produzir [WEL 02].

Um dos aspectos mais polêmicos de XP é sua rejeição à elaboração de um projeto antecipado em prol de uma abordagem mais evolucionária. Esse aspecto é considerado por seus opositores como um retorno ao desenvolvimento baseado em tentativa e erro (enquanto seus adeptos cultuam que, se o código for bem elaborado, um bom projeto surgirá como conseqüência). Além disso, XP dispõe de práticas como integração contínua, teste e refatoramento, que tornam o projeto evolucionário plausível.

Assim, segundo Wells [WEL 02], há muito mais projeto em XP do que nos modelos tradicionais da ES (por exemplo, o Modelo Cascata, o Incremental e o Espiral) [PRE 95]; a diferença reside no fato de que, em XP, o projeto é feito em partes, porém com maior freqüência.

Algumas considerações de XP a respeito do projeto serão mostradas a seguir. Todas visam atingir os seguintes objetivos: (1) manter o código o mais claro e simples possível, (2) melhorar o sistema de uma forma confiável usando refatoramento, (3) garantir um bom conhecimento dos padrões em termos de utilização, implementação simples, evolução e remoção, e (4) saber como comunicar um projeto a pessoas que precisam entendê-lo, usando códigos, diagramas e, principalmente, conversação.

(a) Simplicidade

Um dos principais *slogans* de XP é: "sempre faça a coisa mais simples que poderia possivelmente funcionar". A simplicidade deve ser mantida durante o maior tempo possível. Essa característica pode ser entendida se considerarmos o fato de que um projeto simples sempre leva menos tempo do que um projeto complexo e também é muito mais fácil de entender e modificar. No entanto, é importante ressaltar que manter a simplicidade de um projeto é um trabalho difícil.

Além disso, é preciso definir o que é simplicidade. Caracterizar um sistema simples como aquele que: passa em todos os testes, garante clareza de código, não possui duplicação e usa o menor número de classes e métodos possível, é uma alternativa. Por outro lado, a restrição de simplicidade de um projeto não precisa ser tão rígida. Afinal, a reestruturação de código deve e será feita mais tarde.

(b) Metáfora do sistema

Uma metáfora do sistema é o que XP utiliza no lugar de uma arquitetura formal para descrever como o sistema funciona. Tipicamente, envolve um conjunto de classes e padrões que formam o núcleo do sistema em construção.

A metáfora ajuda as pessoas envolvidas no projeto a compreender o sistema sem a necessidade de um conhecimento específico, na maioria das vezes, difícil de adquirir. Para isso, o sistema é descrito em poucos parágrafos, deixando de lado o uso de termos técnicos, facilitando a compreensão não apenas dos desenvolvedores, mas também dos clientes.

(c) Cartões CRC

Os cartões CRC são um conjunto de cartões de índice utilizados para gravar as responsabilidades e colaboradores das classes dos objetos, por isso o nome derivado de Classe-Responsabilidade-Colaboração.

Individualmente, os cartões CRC são usados para representar objetos. A classe do objeto pode ser escrita no topo do cartão, as responsabilidades listadas do lado esquerdo e as classes de colaboração listadas à direita de cada responsabilidade. Um dos maiores benefícios dos cartões CRC é o estímulo à criatividade; e uma das maiores

críticas é a falta de projeto escrito, o que, se necessário, pode ser resolvido com a retenção de alguns cartões CRC, completamente preenchidos, para documentação.

(d) Soluções pontuais (específicas)

Uma solução pontual pode ser criada concentrando-se apenas no problema que está sendo examinado e ignorando-se todas as outras preocupações. Desta forma, soluções potenciais poderão ser exploradas. O objetivo dessa abordagem é reduzir o risco de um problema técnico e aumentar a confiabilidade das estimativas.

(e) Nunca adicione funcionalidade antes do necessário

O projeto, como dito anteriormente, é elaborado em partes e, desta forma, cada ciclo de iteração busca adicionar somente a funcionalidade especificada pela *user story*. XP considera que grande parte das "tarefas extras" não chegará a ser usada e apenas o que é necessário para hoje deve ser levado em consideração. As funcionalidades extras aparecerão naturalmente quando forem necessárias.

O objetivo desta prática é diminuir a complexidade do projeto ao máximo e facilitar o teste, a alteração e a reconstrução. Também contribui para a agilidade do processo de iteração, garantindo o cumprimento do cronograma de entrega da versão.

(f) Reconstrução - refatoramento

A reconstrução (ou refatoramento) baseia-se na remoção de redundância, eliminação de funcionalidades inúteis e reconstrução de projetos obsoletos. Essa prática garante a atualização constante do projeto em XP.

Além disso, a reconstrução praticada durante todo o ciclo de vida do projeto traz alguns benefícios, como: (1) economia de tempo e aumento da qualidade, (2) manutenção da simplicidade do projeto à medida que esse evolui, (3) a desordem e a complexidade desnecessária são evitadas e (4) o código permanece claro e conciso garantindo maior facilidade de compreensão, modificação e extensão.

3.4.3.5 Limitações do XP

A visão para o desenvolvimento de software apresentada por XP promete: reduzir o risco do projeto através de ciclos curtos e desenvolvimento incremental e aumentar a produtividade das equipes e a qualidade do software que é entregue dentro dos custos e cronogramas planejados.

Entretanto, XP não se aplica bem a todo e qualquer tipo de projeto e, como todo processo, possui algumas limitações. Para que sua aplicação seja produtiva, são necessárias algumas características, entre elas:

- **grupos pequenos:** XP supõe que as equipes de desenvolvimento de um projeto possuem de 2 a 10 programadores;
- **trabalho em equipe:** XP expande a equipe de desenvolvimento incluindo forte integração entre gerentes e clientes durante todo o processo de desenvolvimento;
- **testabilidade**: é mister poder criar testes funcionais e unitários automatizados. Às vezes é necessário alterar o projeto do sistema para facilitar os testes;
- **produtividade:** é necessária uma equipe de desenvolvimento comprometida e dinâmica para assegurar um alto grau de produtividade, que é indispensável na realização dos projetos XP;
- **agilidade na comunicação com o cliente:** a metodologia de desenvolvimento enfoca a rapidez da implementação e, desta forma, a comunicação com o cliente deve ser ágil. É preciso que o cliente especialmente dedicado ao projeto possa tomar decisões rápidas para garantir o cronograma do projeto.

Desta forma, XP requer uma mudança cultural profunda, o que nem sempre é fácil de alcançar. Além disso, alguns obstáculos à sua implementação podem surgir, como: gerentes ou clientes que insistem em ter um conjunto completo de especificações ou um projeto detalhado antes da fase de codificação, ou ainda, sistemas com uma grande quantidade de aplicações já existentes e difíceis de serem alteradas, não oferecem flexibilidade suficiente para garantir a simplicidade no código, um dos requisitos de XP.

Um outro ponto importante diz respeito aos detalhes da especificação da arquitetura do sistema a ser desenvolvido. XP, ao contrário de alguns outros processos da ES, não sugere a elaboração de um projeto arquitetural em suas atividades. Porém, vale lembrar que o projeto arquitetural é um artefato de fundamental importância, face que a discussão da arquitetura do sistema, ainda em fases iniciais do projeto, favorece bastante o entendimento do problema, o processo de tomada de decisões por parte dos envolvidos no projeto, a adoção (ou a rejeição) de determinadas tecnologias e, é claro, ajuda a resolver questões estruturais de sistemas complexos.

Apesar de adotar uma abordagem minimalista, XP requer uma alta disciplina e espírito colaborativo entre os membros da equipe. Levando-se em consideração os valores, princípios e requisitos necessários, é possível tirar proveito de seu principal ponto forte: o desenvolvimento de software em ambientes instáveis que exigem respostas rápidas às constantes mudanças que lhes são inerentes.

De modo geral, os limites para se implementar um processo como XP não estão totalmente definidos. Contudo, da mesma forma que os outros modelos de processo de desenvolvimento de software, XP possui benefícios e limitações que devem ser analisados antes de se decidir sobre sua aplicação.

3.4.4 Scrum

Uma outra metodologia de desenvolvimento classificada como ágil é o SCRUM. Esta metodologia tem como principal objetivo fornecer um processo claro e conveniente para projeto e desenvolvimento de sistemas orientado a objeto.

De acordo com Sutherland [SUT 01] e Scwaber [SCW 95], o SCRUM é um processo de desenvolvimento leve e baseado em princípios semelhantes aos de XP: (1) equipes pequenas, (2) requisitos pouco estáveis ou desconhecidos e (3) iterações curtas (para promover melhor visibilidade para o desenvolvimento e melhor gerenciamento de riscos). No entanto, as dimensões apresentadas em SCRUM diferem de XP.

SCRUM divide o desenvolvimento em *sprints*[5] de 30 (trinta) dias. Equipes pequenas, de até 7 (sete) pessoas, são formadas de projetistas, programadores,

[5] O termo *sprint* refere-se à janela de tempo (iteração) utilizada no processo.

engenheiros e gerentes de qualidade. Estas equipes trabalham em cima de funcionalidades (os requisitos, em outras palavras) definidas no início de cada *sprint*. A equipe é responsável pelo desenvolvimento desta funcionalidade.

Diariamente, é feita uma reunião de 15 minutos onde o time expõe à gerência o que será feito no próximo dia. Nestas reuniões, os gerentes podem levantar os fatores de impedimento (*bottlenecks*) e o progresso geral do desenvolvimento.

SCRUM se mostra interessante porque fornece não somente um mecanismo de informação de *status* que é atualizado continuamente, mas também porque utiliza a divisão de tarefas dentro da equipe de forma explícita.

3.4.5 Crystal/Clear

Crystal/Clear faz parte, na realidade, de um conjunto de metodologias criado por Cockburn [COC 03a][COC 03b]. Segundo o autor, as premissas que suportam a existência deste conjunto são:

- todo projeto tem necessidades, convenções e uma metodologia diferente;
- o funcionamento do projeto é influenciado pelas pessoas e, logo, há melhora neste quando os indivíduos produzem melhor;
- comunicação melhor e lançamentos freqüentes reduzem a necessidade de se construir produtos intermediários do processo.

Crystal/Clear é uma metodologia direcionada a projetos pequenos, com equipes de até 6 desenvolvedores. Assim como com SCRUM, citado na seção anterior, os membros da equipe têm especialidades distintas. No Crystal/Clear existe uma forte ênfase na comunicação entre os membros do grupo, e a organização do espaço de trabalho deve permitir este tipo de atividade.

Toda a especificação e projeto são feitos informalmente, utilizando quadros publicamente visíveis. Os requisitos do sistema a ser produzido são elaborados utilizando *use cases*, um conceito similar às *user stories* em XP, onde são enunciados os requisitos como tarefas e um processo para sua execução. Os *releases* de software são feitos em incrementos regulares de um mês e existem alguns subprodutos do processo que são responsabilidade de membros específicos do projeto.

Grande parte da metodologia é pouco definida e, segundo o autor, isto é proposital; a idéia central de Crystal/Clear é permitir que cada organização implemente as atividades que lhe parecem adequadas, fornecendo um mínimo de suporte útil do ponto de vista de comunicação e documentos.

3.4.6 Diversidade de processos e organizações

Embora muitos gerentes sejam céticos com relação a essas novas metodologias, elas funcionam em muitos casos, mas requerem atenção especial a algumas questões, tais como perfil da equipe, adaptação de processos e sistemas de controle e gestão.

A definição de um software, seu desenvolvimento e os processos de implantação necessitam ser revistos e customizados de modo que os métodos ágeis possam ser usados com sucesso. Cada organização é diferente, tem necessidades particulares e os processos e métodos escolhidos devem se ajustar a essas necessidades sob o risco de comprometer o sucesso do projeto. A respeito deste assunto, Lindvall [LIN 00], ao abordar a variedade de metodologias ágeis, afirma que, da mesma forma que nos processos ditos convencionais, não existe um único modelo ágil uniforme que possa descrever com precisão o que de fato acontece durante todas as fases da produção de um software. Os processos ágeis também necessitam ser adaptados e implementados de acordo com as necessidades de cada organização, as quais diferem substancialmente.

3.5 Discussão

No XP, o cliente e usuário devem fazer parte da equipe de desenvolvimento. Estes, não devem ser vistos como entidades externas que encomendam um sistema e, meses depois, o recebem. No projeto, é importante que se estabeleça um relacionamento de participação e confiança entre o time com o cliente e usuário desde o início, e que esta parceria seja mantida até o final.

A transparência é fundamental, e o XP dá uma grande importância a manter o usuário informado de forma honesta para que ele possa compreender todo o *status* atual do sistema. Os testes viabilizam isto, pois o usuário sempre sabe quantos testes foram

bem-sucedidos e quantos falharam. Ele nunca tem uma visão irreal de que, por exemplo, o sistema está “80% pronto”, sem saber 80% de quê.

O estímulo às participações do cliente e do usuário, além da transparência em todo o projeto permitem que se lide de forma efetiva com o maior problema do desenvolvimento de sistemas: requisitos incompletos e incorretos. O fato é que, muitas vezes, o cliente e usuário não sabem o que realmente deve ser feito por um sistema até que este esteja em uso. O conceito abstrato que se tem no início de um projeto nunca sobrevive ao teste da realidade e, inevitavelmente, surgem mudanças a serem feitas para que qualquer sistema atinja seus objetivos de negócio. O processo interativo do XP permite que o cliente, usuário e analistas aprendam mais cedo o que o sistema realmente deve fazer e aumenta as chances de que ele seja implementado dentro de prazo e custos aceitáveis.

Alem disso, o XP estipula que, entre cada iteração, as *user stories*, testes e cenários devem ser revistos e todas as estimativas de prazo reavaliadas. Ao contrário das metodologias tradicionais, o XP não “congela” requisitos, prazos e custos no início do projeto: ele parte de uma estimativa inicial grosseira, que vai melhorando à medida que se avança no desenvolvimento (e aprendizado) do sistema.

Com base no que foi apresentado, escolheu-se o XP para, junto com o Método para Concepção de Interfaces (MCI), fundamentar a confecção do modelo centrado no usuário.

3.6 Considerações finais do capítulo 3

Grande parte das pesquisas feitas na área de ES e, em particular, no desenvolvimento de processos de software, continuam em andamento e têm contribuído com melhorias na construção de produtos de software. No entanto, existe uma tendência atual para a simplificação e pragmatização do processo visando acomodar novas necessidades de desenvolvimento do mercado; os requisitos e demandas por sistemas são muito diferentes dos conhecidos e estabelecidos nos anos 70 e 80.

A aplicação de um processo de desenvolvimento de software não é sempre tão fácil como parece. As técnicas de gerência tradicionais de projeto são pouco capazes de adaptar-se bem às mudanças e aos novos riscos. Isto faz com que as equipes tenham cada vez mais dificuldades para reagirem rapidamente às mudanças intrínsecas de um mercado moderno turbulento, ágil e altamente competitivo. Entretanto, as novas técnicas ágeis podem tornar mais fácil a vida de gerentes de projeto e equipes, desde que executadas corretamente. Uma equipe qualificada e comprometida conseguirá entregar software melhor e mais barato, rapidamente.

Foram discutidos inicialmente, no capítulo, características, princípios e valores das metodologias tradicionais de desenvolvimento de aplicações. Em seguida, foram abordados não somente o panorama do mercado produtor de software atual como também, de maneira mais específica, a boa aceitação das metodologias ágeis de desenvolvimento de sistemas de software neste cenário. Ainda neste capítulo, algumas metodologias ágeis foram apresentadas. Finalmente e, com base na exposição destes modelos de desenvolvimento da área de ES, foi apresentado e detalhado o processo de desenvolvimento a ser adotado na geração do modelo centrado no usuário.

Capítulo 4

A Metodologia XPU

4.1 Considerações iniciais

Neste capítulo, é apresentada a proposta de um modelo para o desenvolvimento de sistemas de software centrado no usuário, o XPU (*eXtreme Programming* (minimizado) + Usabilidade (MCI)). Inicialmente, aspectos importantes relacionados à descrição de sua estrutura, bem como fluxos gerais de trabalho, serão comentados. Em seguida, um detalhamento acerca das fases do processo XPU - especificando atividades, artefatos indicados, suporte ferramental, benefícios e limitações existentes - estará sendo apresentado e analisado neste capítulo.

4.2 Introdução

Com base na exposição e análise de um conjunto de princípios e valores pertencentes às áreas de Usabilidade e Engenharia de Software (ES), presentes nos Capítulos 2 e 3, respectivamente, será descrita, neste momento, a metodologia para concepção de sistemas centrada no usuário, denominada XPU (*eXtreme Programming* + Usabilidade (MCI)). Os conceitos apresentados nos dois capítulos precedentes serviram de um preâmbulo importante antes que pudéssemos detalhar o processo XPU.

No início do trabalho, onde foi possível coletar dados tanto em torno da ES quanto Interação Homem-Computador (IHC), pôde-se, sob discussões iniciais, dar início à idealização de um primeiro arcabouço da metodologia pretendida. É importante ressaltar, no âmbito da ES, que este último foi o resultado de pesquisas e comparações realizadas em torno de recomendações, práticas, técnicas e artefatos propostos por

algumas metodologias de desenvolvimento de software da ES (sejam densas ou não), como: Processo Unificado (*Unified Process*) [JAC 99], *easYProcess* (YP) [EAS 03][PET 03], *eXtreme Programming* (XP) [BEC 00][MAU 02][JEF 02], SCRUM [SUT 01][SCW 95], Crystal/Clear [COC 03a][COC 03b]. Entretanto, conforme o leitor pôde observar no Capítulo 3, os estudos realizados neste trabalho apontaram para a escolha de recomendações, práticas, técnicas e artefatos adotados nos ambientes ágeis de desenvolvimento de sistemas; com uma ênfase maior em XP como o processo base a ser utilizado na integração e geração do modelo pretendido.

Por outro lado, no âmbito da IHC, técnicas de usabilidade são sugeridas na composição de XPU, segundo uma adaptação ao Método para Concepção de Interfaces (MCI), uma metodologia que incorpora princípios de Usabilidade. No âmbito da ES e IHC, portanto, XPU é um processo de desenvolvimento de sistemas ágil, leve (*lightprocess*) e centrado no usuário.

Assim sendo, esses dois universos - ES e IHC - foram combinados, uma vez que algumas das melhores práticas do desenvolvimento iterativo e incremental foram integradas com práticas de usabilidade na intenção de gerar-se um novo processo para a concepção de aplicações.

O objetivo principal do XPU é prover à equipe de desenvolvimento um modelo para a construção de sistemas de software centrado no usuário que valorize a usabilidade como característica fundamental de qualidade. Portanto, XPU deve guiar o time de desenvolvimento, passo a passo em todo o projeto, segundo técnicas e artefatos simplificados, a fim de se obter não apenas sistemas portáveis, fracamente acoplados ou reutilizáveis, mas, sobretudo, interfaces que reflitam os objetivos, as características e as necessidades do usuário.

Nesse sentido, o presente capítulo apresenta uma descrição e análise da metodologia XPU e de suas partes. Primeiramente, será apresentada a estrutura geral desta conduta de desenvolvimento de software. Ainda neste capítulo, serão expostos e comentados os fluxos de trabalho fundamentais de XPU. Em seguida, cada uma das principais fases do processo será detalhada, apresentando sua descrição, atividades, documentação e suporte ferramental indicado. Ao término do capítulo, algumas considerações finais serão apresentadas.

4.3 Estrutura do processo XPU

Nesta seção, são apresentados de forma sucinta as principais fases, atividades e artefatos envolvidos na metodologia XPU. A Figura 6, mais adiante, ilustra as fases através de pequenos retângulos; por meio de setas são indicados os fluxos de trabalho do processo e, finalmente, algumas atividades e artefatos relacionados a cada uma das fases são revelados.

XPU reflete um processo de construção de sistemas de software dividido em 7 (sete) fases específicas: (1) definição de papéis, (2) conversa com o cliente, (3) inicialização, (4) planejamento de *releases*, (5) planejamento da iteração, (6) implementação e (7) verificação de testes. Conforme já foi dito anteriormente, na figura adiante, algumas atividades e artefatos também são revelados com o objetivo de favorecer a compreensão do processo:

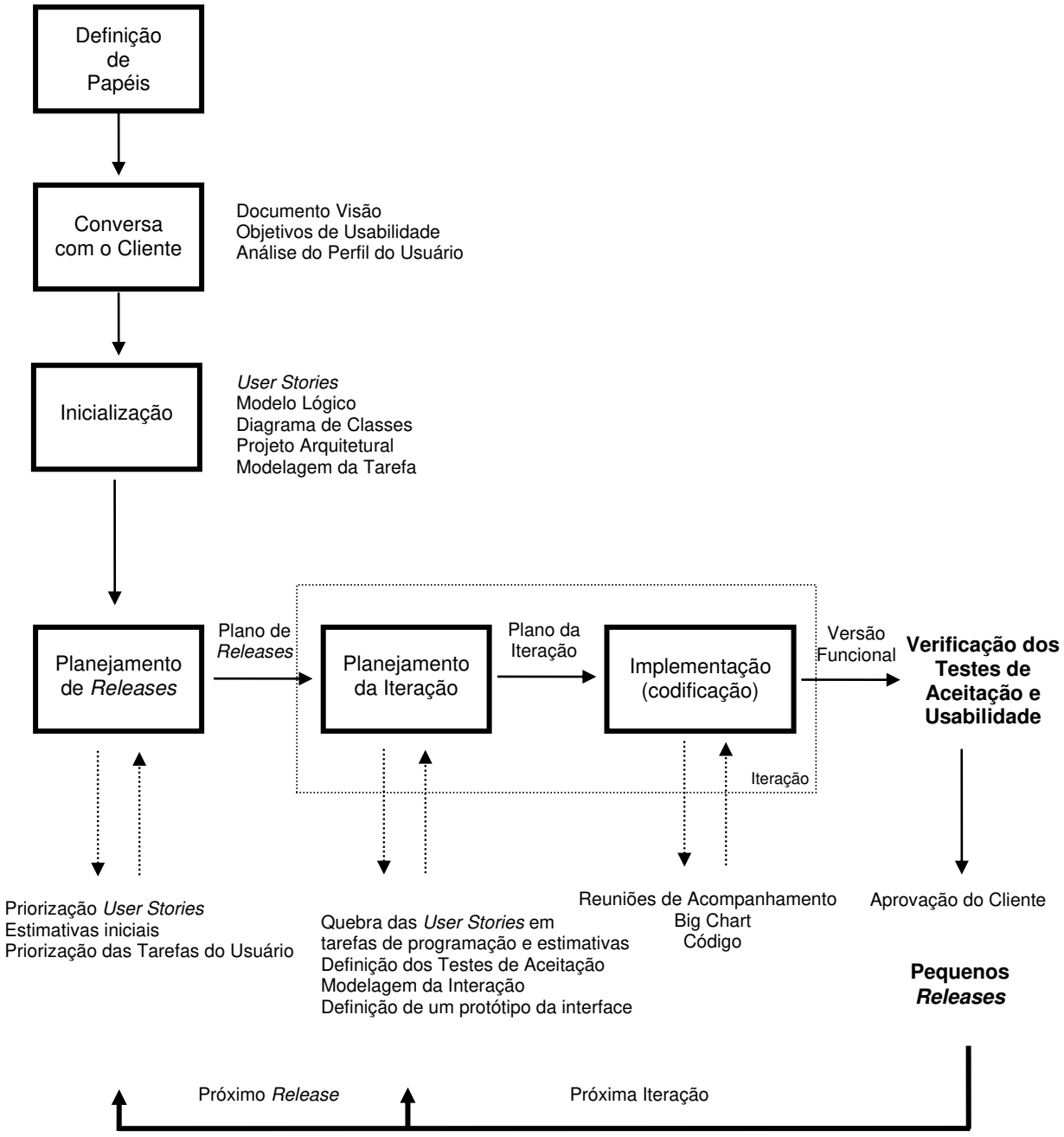

Figura 6: Estrutura do processo XPU.

A seguir, será apresentado um detalhamento acerca de cada uma das fases do processo XPU. Este detalhamento dará ao leitor a oportunidade de se conhecer determinadas particularidades relacionadas ao processo, seus princípios fundamentais, artefatos e também suporte ferramental.

4.4 Detalhamento das fases

Nesta seção, será detalhada cada uma das fases propostas por XPU. Inicialmente, será apresentada uma descrição geral da fase com o objetivo de contextualizá-la dentro do processo. Em seguida e, em cada fase, serão abordados os artefatos, bem como o suporte ferramental sugerido pelo processo para a sua confecção (preferencialmente, ferramentas de código livre).

4.4.1 Fase de definição de papéis

4.4.1.1 Descrição

Ao observar as fases e princípios básicos do processo XP, presentes no Capítulo 3, percebe-se que esta metodologia de desenvolvimento já fornece alguns indicadores relacionados não apenas aos papéis, mas também, de forma específica, a algumas tarefas (ou responsabilidades) que devem ser alocadas à cada um dos membros do time de desenvolvimento. Isto porque é fundamental, em um projeto de software, que cada membro do time saiba exatamente que atividade deverá desempenhar. Dentre estes papéis sugeridos por XP, destacam-se: (1) cliente, (2) desenvolvedor (ou programador) e (3) gerente de projeto.

Entretanto, apesar de mencionar estes papéis de maneira clara, o processo XP não faz referências explícitas a respeito de um momento específico (ou fase) que vise à alocação destes comportamentos a cada um dos membros do time, ainda no início do processo. O processo YP [EAS 03][PET 03], uma outra metodologia ágil de desenvolvimento de software que também se apóia intensamente em práticas oriundas do XP, traz uma recomendação bastante útil: uma fase para alocação de papéis ainda no início do projeto. XPU também adota esta prática, porém recomenda que novos papéis sejam acrescentados.

Foi visto que, em equipes XP, os programadores escrevem primeiro os testes (testes unitários) para só então desenvolver o software que atenda aos requisitos destes testes. Além disso, em XP, o cliente é chamado para auxiliar na descrição de testes (testes de aceitação) para ter certeza de que os recursos necessários ao sistema estão

sendo fornecidos. Em XPU, porém, a supervisão, geração e aplicação de testes em cima do código de outro desenvolvedor serão realizadas por um outro ator dentro do time: o "testador". Mais adiante, serão detalhadas outras tarefas relacionadas a este papel.

Uma vez que se pretende incorporar técnicas de usabilidade ao modelo de desenvolvimento de software pretendido, faz-se necessário incluir um novo papel dentro do processo: o do "especialista em usabilidade". De forma geral, este indivíduo deverá acompanhar a introdução das práticas bem como a geração dos artefatos de usabilidade sugeridos pelo MCI [TUR 02]. Mais adiante e, especificamente, serão comentadas outras tarefas relacionadas a este papel.

Em XPU, como em algumas outras metodologias da ES, é aconselhado que cada indivíduo possa desempenhar um único papel dentro do processo. Obviamente, porém, no caso de equipes formadas com um número pequeno de pessoas, uma mesma pessoa deverá desempenhar mais de um papel, simultaneamente. A respeito deste assunto, vale ressaltar que esta prática deve ser realizada com bastante cautela, para que não haja atrasos nos cronogramas.

De maneira semelhante ao processo XP, o papel do cliente também é importante dentro de XPU. Maurer e Martel [MAU 02], ao abordar os doze princípios básicos de XP, informam que um dos principais fatores que contribuem para o sucesso desta metodologia ágil se deve à participação ativa do cliente, tornando-o parte integrante da equipe de desenvolvimento ("*on-site customer*") e disponível durante todo o projeto. Porém, em muitos ambientes de produção de software, esta realidade nem sempre é possível e o cliente poderá estar indisponível durante parte do desenvolvimento ou ainda pior: não integrar o time. Assim sendo, caso o cliente não possa participar ativamente ao longo do projeto, é imprescindível que o mesmo indique um representante ainda no início do projeto.

Ainda, além da participação do cliente poder ser diferenciada de um projeto de software para outro, em alguns casos, também, nem sempre o cliente será o próprio usuário final do sistema. Em alguns projetos, o cliente (normalmente, o contratante) e o usuário final do sistema são indivíduos distintos, possuindo cada um, portanto, características, habilidades e preferências também diferentes. Assim sendo, XPU sugere

uma divisão explícita destes papéis e, portanto, o aparecimento de um novo membro na equipe de desenvolvimento: o "usuário" do sistema.

Em síntese, XPU recomenda a presença de seis papéis: (1) cliente, (2) usuário, (3) gerente, (4) desenvolvedor, (5) testador e (6) especialista em usabilidade.

A seguir, serão descritas e detalhadas as tarefas alocadas em cada um destes papéis:

- **Cliente:** conforme visto no Capítulo 3, XP cita algumas tarefas que são de responsabilidade do cliente do projeto, entre elas: (1) elaborar e priorizar as *user stories* do sistema, (2) ser especialmente dedicado e integrar a equipe de desenvolvimento, (3) acompanhar a geração do plano de *releases,* (4) elaborar e validar os testes de aceitação, (5) aprovar (ou não) um *release* gerado e (6) ser um indivíduo comunicativo e que tome decisões rapidamente (para que o cronograma do projeto seja preservado).
- **Usuário**: em XPU, o usuário do sistema é o responsável por: (1) ajudar a definir e validar os testes de aceitação e de usabilidade, (2) acompanhar a modelagem da tarefa, (3) ajudar a levantar o perfil do usuário e (4) fornecer *feedback* contínuo não somente a respeito do sistema de um modo geral, mas, sobretudo, sua interface.
- **Gerente:** em XPU, o gerente do projeto é o responsável por: (1) alocar as competências e acompanhar o progresso do time, (2) obter os recursos necessários ao projeto, (3) prever, identificar e mitigar riscos, (4) gerar os planos de *release* e iteração junto aos desenvolvedores e o cliente e (5) resolver problemas internos.
- **Desenvolvedor:** em XPU, o desenvolvedor é o responsável por: (1) ajudar a gerar as *user stories* junto ao cliente do projeto, (2) gerar testes unitários, (3) refatorar código, (4) manter a integração contínua de código, (5) ajudar a gerar um protótipo da interface, (6) estimar tempo ideal de desenvolvimento e (7) ajudar o gerente a gerar os planos de *release* e iteração.
- **Testador:** em XPU, o testador é o responsável por: (1) supervisionar e gerar de testes de unidade em cima do código de outro desenvolvedor, (2) supervisionar os testes de usabilidade junto ao especialista de usabilidade e (3) gerar os testes de aceitação junto ao cliente e usuário do projeto.

- **Especialista em usabilidade:** em XPU, o especialista em usabilidade é o responsável por: (1) levantar os objetivos de usabilidade junto ao cliente, (2) levantar o perfil do usuário, (3) realizar a modelagem da tarefa do usuário e da interação, (4) elaborar testes de usabilidade, (5) definir um protótipo da interface do sistema e (6) avaliar continuamente a interface do sistema junto aos usuários.

4.4.1.2 Artefatos a serem produzidos

Nesta fase, sugere-se apenas a construção de uma tabela simples contendo o nome do membro da equipe, juntamente com o papel que este último deverá assumir durante o projeto.

4.4.1.3 Suporte ferramental

Para a confecção da tabela citada há pouco, um processador de textos comum pode ser utilizado.

4.4.2 Fase de conversa com o cliente

4.4.2.1 Descrição

Algumas metodologias densas (*heavyprocesses*) de desenvolvimento de sistemas da ES [JAC 99][KRU 00] dispõem de artefatos específicos que devem ser gerados pelo time para coletar as informações necessárias do ponto de vista do negócio, determinando escopo e viabilidade de um projeto de software.

Entretanto, conforme se observou no Capítulo 3, XP não menciona (de forma explícita) um único artefato que possa contemplar, ainda no início do projeto, estas duas informações, simultaneamente: visão de negócio e escopo do projeto. Considerando a relevância de tal artefato, XPU recomenda a elaboração do Documento de Visão [EAS 03][PET 03]. Este documento deve, preferencialmente, ser gerado com as presenças do cliente do projeto juntamente com todo o time de desenvolvimento.

Além do documento de visão citado há pouco, no âmbito da usabilidade, dois artefatos devem ser produzidos: o documento contendo o perfil do usuário e uma lista contendo os objetivos de usabilidade (a serem mais adiante confrontados com testes de

usabilidade). De acordo com Turnell [TUR 02], de forma geral, o Documento de Perfil do Usuário deve definir o perfil dos usuários potenciais em termos das tarefas a serem realizadas: seu potencial, limitações, preferências, interesses, aptidões e os conhecimentos necessários para realizá-las. Por sua vez, segundo a mesma autora, os objetivos de usabilidade devem se fundamentar não somente no conhecimento do perfil do usuário, mas, sobretudo, na inspeção de produtos similares. Isso porque o levantamento de problemas em outros produtos com propósitos similares auxilia a proposição de um conjunto de objetivos de usabilidade que visem superar os problemas encontrados, favorecendo, assim, a qualidade do produto final.

4.4.2.2 Artefatos a serem produzidos

- **Documento de Visão:** (vide Anexo A).
- **Lista dos Objetivos de Usabilidade:** (vide Anexo B).
- **Documento de Perfil do Usuário:** (vide Anexo C).

4.4.2.3 Suporte ferramental

Para a confecção do Documento de Visão e da Lista de Objetivos de Usabilidade, um processador de textos comum pode ser utilizado. Entretanto, para se conhecer os perfis dos usuários, é apresentado, no Anexo C, um questionário de Levantamento do Perfil do Usuário adotado no MCI e incorporado ao XPU para este fim.

4.4.3 Fase de inicialização

4.4.3.1 Descrição

Uma vez com a equipe de desenvolvimento formada, identificados o escopo do problema e viabilidade do projeto, levantados o perfil do usuário e os objetivos de usabilidade, é o momento de dar início a algumas atividades de planejamento. Nesta fase de inicialização, além de se gerar alguns artefatos já conhecidos e utilizados por boa parte da comunidade de ES (por exemplo, diagrama de classes, projeto arquitetural), outros artefatos oriundos de XP e IHC (por exemplo, modelo da tarefa) devem ser produzidos.

Conforme visto no Capítulo 3, para se levantar os requisitos funcionais do sistema a ser produzido, o time deve, com o cliente do projeto, gerar as *user stories* do sistema. Beck [BEC 00] cita que o cliente (ou representante, como foi dito anteriormente) é quem deve priorizar quais *user stories* serão implementadas primeiro (e não o time de desenvolvimento, como acontece em alguns projetos de software). Em XPU, além dos artefatos citados no parágrafo anterior, o time deve gerar um modelo lógico de dados [SIL 99] (caso haja um banco de dados envolvido no sistema).

Por fim, no âmbito da usabilidade, deve-se realizar a modelagem da tarefa do usuário. O artefato modelo da tarefa é conhecido e utilizado por boa parte da comunidade de IHC. Em XPU, este artefato pode complementar a eficácia das *user stories*, uma vez que alguns estudos [COR 03] mostram a adequada utilização das tarefas do usuário resultantes da atividade de modelagem da tarefa da IHC para levantar os requisitos funcionais de uma aplicação computacional. Segundo Turnell [TUR 02], modelar a tarefa do usuário consiste em realizar um estudo detalhado desta, a fim de determinar sua natureza, propósito, as partes que a compõem (subtarefas ou ações básicas) e a ordem em que estas partes devem ser executadas.

Mais adiante, é apresentado um detalhamento em torno de cada artefato sugerido:

4.4.3.2 Artefatos a serem produzidos

Nesta fase do projeto, deve-se gerar os seguintes artefatos:

- ***User stories*:** a equipe de desenvolvimento deve definir junto ao cliente todas as *user stories* do sistema [BEC 00][MAU 02][JEF 02].
- **Modelo lógico de dados:** se houver um banco de dados envolvido no projeto, este artefato deve ser produzido. De uma forma geral, a modelagem de dados baseia-se na percepção de um universo constituído por um grupo básico de objetos chamado *entidades* e por *relacionamentos* entre estes objetos [SIL 99][LAR 00]. O modelo lógico foi desenvolvido a fim de facilitar o projeto do banco de dados permitindo a especificação de um *esquema*. Tal esquema representa a estrutura lógica global do banco de

dados. A partir daí, pode-se então implementar esta estrutura no SGBD[6] e, posteriormente, "povoá-la" com informações.

- **Projeto arquitetural:** segundo Sauvé [SAU 03], este artefato tem como objetivo descrever o funcionamento do sistema num alto nível de abstração. Este é útil quando se deseja explicitar como as partes do sistema a ser desenvolvido interagem entre si ou com outros sistemas [SHA 96]. Em geral, o projeto arquitetural contém informações que sofrem pouca ou nenhuma alteração ao longo do processo.
- **Diagrama de classes:** uma das principais atividades realizadas durante um projeto de software diz respeito à representação bem definida, na forma de um artefato, dos objetos que irão compor o sistema. Tal artefato é conhecido como diagrama de classes. Segundo Larmam [LAR 00], normalmente, cada uma das classes (e seus relacionamentos) descrita neste diagrama possui um conjunto de operações (ou métodos) que, por sua vez, permitem que os objetos possam interagir entre si na realização de uma dada funcionalidade da aplicação.
- **Modelo da tarefa:** o resultado da análise da tarefa do usuário consiste na descrição detalhada e hierarquizada do trabalho, com um diagnóstico das situações problemáticas, graus de dificuldades e as soluções possíveis, e/ou recomendações ergonômicas para a concepção da interface do futuro sistema. Um exemplo deste artefato pode ser encontrado no Anexo D desta dissertação.

4.4.3.3 Suporte ferramental

Para a construção do projeto arquitetural, é aconselhado o uso de uma ferramenta *case* que esteja de acordo com a linguagem *Unified Modeling Language* (UML). No momento da realização deste trabalho de dissertação, foram encontrados alguns bons ambientes de software disponíveis no mercado, como, por exemplo: *Poseidon for UML* [POS 03] e *BlueJ* [BLU 03].

No momento da definição deste trabalho, o MCI sugeria apenas o uso do ambiente *Euterpe* [EUT 03] para a confecção do modelo da tarefa. Assim sendo, o XPU também sugere a utilização de tal ambiente. Trata-se de uma ferramenta para análise e

[6] Sistema Gerenciador de Banco de Dados.

descrição de tarefas do usuário de código livre, fácil aprendizado e que pode ser baixada sem ônus algum para a equipe de desenvolvimento. Porém, é muito importante frisar que, atualmente, existe um interessante trabalho [SCH 03] em andamento na academia no tocante ao estudo e levantamento de um suporte ferramental mais adequado ao método MCI onde serão apontadas, ao final, outras ferramentas acadêmicas ou comerciais que melhor se adequem a tal método.

4.4.4 Fase de planejamento de releases

4.4.4.1 Definição

Concluídas as atividades da fase anterior, deve ser iniciado o planejamento de *releases* do projeto.

No Capítulo 3, onde foram abordadas as principais fases da metodologia XP, foi visto que este processo sugere a construção de um plano de entrega para as novas versões do produto. XP [BEC 00][MAU 02][JEF 02] nos diz ainda que o conjunto de *user stories* levantadas devem, agora, ser priorizadas pelo cliente do projeto e agrupadas nos *releases*. Vale lembrar ainda que, em XPU, cada *release* é formado por duas iterações de duas semanas cada uma, totalizando uma janela de tempo de 30 (trinta) dias. Conforme pode-se notar, os ciclos de desenvolvimento de XPU são bastante curtos. E isto não é por acaso: Maurer e Martel [MAU 02], ao abordar o processo XP, citam que uma das doze práticas chaves de XP consiste em estabelecer ciclos curtos de desenvolvimento, de modo que o time possa entregar rapidamente software valioso ao cliente, num ritmo constante.

Ainda, segundo Beck [BEC 00], um dos objetivos da reunião para a criação do plano de *releases* do produto é o de permitir ao time de desenvolvimento estimar cada *user story* (priorizada pelo cliente) em termos de tempo ideal de programação (em semanas).

Assim, de forma análoga ao processo XP [BEC 00][MAU 02][JEF 02], o planejamento geral de XPU consiste de dois planejamentos: o de *releases* e o da iteração.

4.4.4.2 *Artefatos a serem produzidos*

Conforme visto no Capítulo 3, XP sugere uma reunião formada entre o cliente do projeto juntamente com o time de desenvolvimento, onde deve ser elaborado um plano de *releases.* Este plano deve conter as *user stories* que foram priorizadas pelo cliente e estimadas pela equipe de desenvolvimento [BEC 00][MAU 02][JEF 02]. No âmbito da usabilidade, deve-se também priorizar as tarefas do usuário a serem implementadas (a partir do modelo da tarefa já construído na fase anterior).

Para facilitar o entendimento do leitor, é fornecido um breve cenário de como este plano de *releases* pode ser elaborado. Para isso, é utilizado um exemplo reduzido de um projeto criado para um *site* de divulgação do Laboratório de Interface Homem-Máquina da Universidade Federal de Campina Grande (LIHM/UFCG). Em tal projeto, apenas duas *user stories* foram priorizadas pelo cliente para compor o primeiro *release* - uma primeira (chamada de US1) para o *cadastro e agendamento de congressos* e uma segunda *user story* (US2) para o *cadastro e agendamento de seminários*. Mais adiante, no âmbito da usabilidade, um modelo da tarefa parcial deste exemplo é apresentado. Por questões de otimização do artefato modelo da tarefa, pode-se observar, na Figura 7, que apenas a sub-árvore "*Conhecer o Laboratório de IHM"* do modelo da tarefa foi trabalhada.

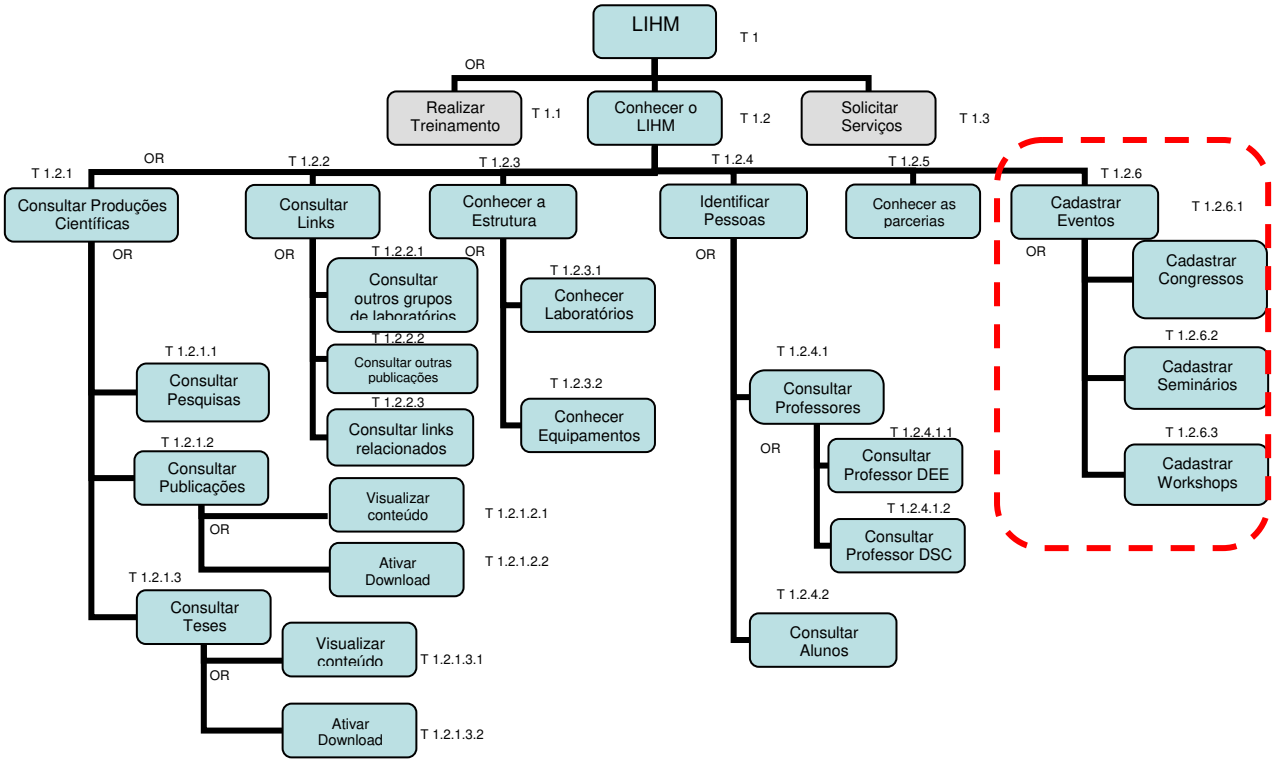

Figura 7: Exemplo de um artefato Modelo da Tarefa.

Realizar uma alocação adequada no *release* entre as *user stories* que foram priorizadas pelo cliente do projeto, juntamente com as tarefas do usuário já levantadas na análise e modelagem da tarefa não é uma atividade trivial (e não há uma regra explícita para este fim). Contudo, para esta atividade, XPU apenas pode sugerir uma combinação das *user stories* e tarefas do usuário que possuam algum tipo de relação entre si. Note, na Figura 7 acima, que a sub-árvore *Cadastrar Eventos* (aparece destacada na Figura 7) é uma boa candidata a combinação com as *user stories US1 e US2* citadas há pouco e pode, então, ser alocada. Finalmente, o plano *de releases* gerado pode ser visto na Tabela 3, adiante:

Release I: 1/11/2003 à 30/11/2003		
Iterações	**Período**	**User Stories**
Iteração 1	1/11 à 15/11	Sub-árvore T1.2.6, US1
Iteração 2	16/11 à 30/11	Sub-árvore T1.2.6, US2

Tabela 3: Exemplo de um Plano de Releases.

Em XPU, é importante ressaltar que, durante a alocação das *user stories*, o tempo de um *release* (ou iteração, conforme veremos na fase posterior) é fixo. O escopo, ou seja, a quantidade de tarefas e/ou *user stories* que serão implementadas, é que deve variar. Em síntese, as atividades que fazem parte do planejamento de *releases* são:

- **Priorização de *User stories*:** XP sugere que a equipe de desenvolvimento deve definir junto ao cliente quais *user stories* do sistema serão implementadas no *release* corrente [BEC 00][MAU 02][JEF 02];
- **Priorização das tarefas do usuário*:*** XPU sugere que a equipe de desenvolvimento (de forma mais específica, o especialista em usabilidade) deve definir junto ao cliente quais tarefas do usuário presentes no modelo da tarefa podem ser combinadas com as *user stories* na constituição do *release* corrente;
- **Construção do plano de *releases*;**

4.4.4.3 Suporte ferramental

Para a confecção do plano de *releases*, um processador de textos comum pode ser utilizado.

4.4.5 Fase de planejamento da iteração

4.4.5.1 Descrição

De acordo com XP [BEC 00][MAU 02][JEF 02], todas as *user stories* estimadas pelo time de desenvolvimento e que foram priorizadas pelo cliente no *release* devem, agora, ser "quebradas" em tarefas atômicas. Além disso, XP sugere que testes de aceitação sejam realizados para cada *user story* junto ao cliente e cada *user story* deve possuir pelo menos um teste de aceitação [BEC 00][MAU 02][JEF 02].

No âmbito da usabilidade, as tarefas do usuário, anteriormente citadas no planejamento de *releases*, também devem ser alocadas na iteração. Porém, devido à própria natureza hierárquica do modelo da tarefa, estas tarefas do usuário não precisam ser "quebradas" em tarefas menores (note o destaque pontilhado na Tabela 4 adiante).

Por fim, a modelagem da interação do usuário com o sistema também deve ser realizada nesta fase. Segundo Turnell [TUR 02], a partir do modelo da tarefa, deve-se construir uma tabela na qual são listados os objetos e ações envolvidos nas tarefas do usuário. Nesta tabela, são listadas as ações elementares, os objetos sobre as quais são realizadas e uma respectiva identificação oriunda do modelo da tarefa. Dessa forma, a modelagem da interação poderá associar os objetos e ações levantados no modelo da tarefa a objetos e ações no domínio da interface do sistema com o usuário.

4.4.5.2 Artefatos a serem produzidos

Nesta fase, sugere-se a geração, por parte do time de desenvolvimento, de um artefato interessante recomendado em YP [EAS 03]. Este é conhecido pelo nome de Tabela de Alocação de Tarefas [EAS 03].

O objetivo desta tabela proposta por YP [EAS 03] é o de agrupar, num único local, os seguintes itens: (1) as tarefas de programação geradas a partir da quebra de *user stories,* (2) o responsável pela realização desta tarefa, (3) o tempo estimado para a conclusão da tarefa, (4) o tempo real consumido para a conclusão e (5) uma coluna contendo o status da tarefa selecionada para a iteração, ao final.

XPU adota este artefato, entretanto, recomenda que não somente as tarefas atômicas de programação, mas também as tarefas do usuário (presentes no modelo da

tarefa) devem ser inseridas nesta tabela. É possível ver, na Tabela 4, um exemplo deste artefato:

TAT – Iteração 1					
Tarefa	**Descrição**	**Responsável**	**Estimativa em Horas**	**Tempo Real em Horas**	***Status*[7]**
TU1.2.6.1[8]	JSP Formulário para Cadastro de Congressos	João	2		
TP1.1	Funcionalidade Cadastro de Congressos (*script* do banco de dados)	Maria	4		
TP1.2[9]	Tratamento de mensagens de erro emitidas	José	3		
TU1.2.6.2	JSP Formulário para Cadastro de Seminários	João	2		
TP2.1	Funcionalidade Cadastro de Seminário (*script* do banco de dados)	Maria	4		

Tabela 4: Tabela de alocação de tarefas [EAS 03].

Além da TAT [EAS 03] indicada por YP, outros artefatos são sugeridos: o modelo da interação [TUR 02] (vide Anexo E) e o protótipo da interface do usuário. Em síntese, as atividades que fazem parte do planejamento da iteração são:

- **Desdobramento das *User stories*:** conforme visto no Capítulo 3, XP sugere que a equipe de desenvolvimento "quebre" as *user stories* do *release* em tarefas de programação atômicas menores. Estas devem ser alocadas na TAT [EAS 03] juntamente com as tarefas do usuário previamente levantadas no modelo da tarefa [TUR 02].
- **Definição dos testes de aceitação:** estes testes devem ser explicitados pelo cliente com a ajuda do usuário do sistema.
- **Modelagem da interação:** segundo [TUR 02], esta atividade prima por levantar os objetos e ações da tarefa do usuário a partir do modelo da tarefa já produzido. Em seguida, há uma associação de cada um destes objetos e ações com objetos e ações do domínio da interface com o usuário. Vide Anexo E desta dissertação para uma melhor compreensão.
- **Definição de um protótipo da interface:** uma vez concluída a modelagem da interação, o time pode montar um protótipo da interface.

[7] Valores sugeridos para este campo: C – concluída, D – em desenvolvimento e A – abortada.
[8] A notação TU1.2.6.1 indica que esta é a tarefa do usuário 1.2.6.1 presente no modelo da tarefa.
[9] A notação TP1.2 indica que esta é a tarefa de programação 2 da *user story* 1.

4.4.5.3 Suporte ferramental

Para a construção da TAT [EAS 03] e das duas tabelas propostas na modelagem da interação do usuário com o sistema [TUR 02] (vide Anexo E), um processador de textos comum pode ser utilizado.

4.4.6 Fase de implementação

De posse do plano da iteração definido na fase anterior, é o momento de se iniciar a codificação das funcionalidades. Entretanto, XPU ainda sugere algumas práticas interessantes defendidas por XP [BEC 00][MAU 02][JEF 02] para favorecer a produtividade da equipe de desenvolvimento nesta fase:

- **Integração Contínua:** a integração contínua de software é uma entre as doze práticas fundamentais de XP [MAU 02]. Fowler [FOW+ 03] nos diz que um dos principais benefícios de se implantar a integração contínua em um ambiente de projeto é o de minimizar erros durante a reunião de partes do sistema geradas pela equipe de desenvolvimento. Isto porque, uma vez que esta integração acontece continuamente, e envolvendo pequenas partes de código (testadas e sem erros), não somente o cliente poderá dispor, ao final do dia, de uma nova versão funcional do sistema, mas também evita-se a complexidade de integrar grandes módulos do sistema, tardiamente. Vale lembrar que esta atividade deve ser realizada com a ajuda de algumas ferramentas, as quais são citadas mais adiante.

- **Propriedade Coletiva:** a propriedade coletiva de código também é uma entre as doze práticas fundamentais de XP [MAU 02]. Basicamente, significa que um código produzido não é de propriedade exclusiva do programador que o fez, mas sim de toda a equipe de desenvolvimento. Segundo Beck [BEC 00], isso melhora bastante a qualidade do código gerado e economiza tempo para detectar e reparar erros no sistema (se um programador encontra uma oportunidade para corrigir, simplificar e melhorar o código gerado por um outro programador, ele pode fazê-lo). Vale lembrar,

entretanto, que a cada correção efetuada no código de outra pessoa, testes devem ser aplicados.

- **Boas Práticas de Programação:** XP cita que, uma vez que o código desenvolvido é compartilhado, é fundamental a utilização de boas práticas de programação por parte de todos os membros da equipe de desenvolvimento. Isso provê não somente uma compreensão melhor e mais rápida do código gerado por outro desenvolvedor, mas também facilita sua correção e simplificação. Ainda, sugere-se aplicar ao projeto as práticas de *design* simples, padrões de codificação, padrões de projeto e refatoramento [GAM 00][JEF 02][WEL 02].

- **Testes:** a atividade de testes em um projeto de software também constitui uma dentre as doze práticas fundamentais de XP [MAU 02][JEF 02][WEL 02]. De acordo com Beck [BEC 00], definir e elaborar testes antes da implementação é uma boa prática, pois auxilia a pensar mais no código antes mesmo que este seja desenvolvido. Desta forma, após o programador codificar algum componente do sistema, um teste específico deve ser aplicado (testes unitários). Caso não se detecte nenhum erro, só então o programador pode integrar esta nova funcionalidade ao sistema. Vale lembrar que, após esta etapa, todos os testes devem, ainda, ser rodados. Conforme visto no Capítulo 3, uma outra categoria de testes é ainda recomendada por XP com o objetivo de validar os requisitos funcionais do sistema junto ao cliente (testes de aceitação). No âmbito da IHC, testes de usabilidade são efetuados junto ao cliente e usuário visando confrontar os objetivos de usabilidade com a interface produzida.

- **Reunião de Acompanhamento:** conforme visto no Capítulo 3, XP sugere encontros (diários) com a equipe de desenvolvimento para que problemas e soluções possam ser discutidos. Essas reuniões orientam e encorajam o time

para o que deve ser feito. Nestes encontros, é sugerido ao gerente do projeto fazer uso de um *Big Chart*[10] [JEF 02] e da Tabela de Alocação de Tarefas (TAT) [EAS 03].

4.4.6.1 Suporte Ferramental

No momento da realização deste trabalho de dissertação, foram encontradas interessantes ferramentas de software disponíveis no mercado. Seguindo uma classificação em relação às principais atividades presentes na fase de implementação, podemos citar:

- **Ferramentas de Integração Contínua:** *Bacon* [BAC 03], *Gump* [GUM 03] e *AntHill* [ANT 03].
- **Ferramentas de Desenvolvimento:** *Ant* [ANT+ 03], *Eclipse* [ECL 03] e *Java* [JAV 03].
- **Ferramentas de Gerência:** *XPlanner* [XPL 03].
- **Ferramentas de Testes:** *JUnit* [JUN 03] e *Cactus* [CAC 03].
- **Ferramentas de Controle de Versões de Software:** *CVS* [CVS 03] e *WinCVS* [CVS+ 03].

4.4.7 Verificação dos testes de aceitação e de usabilidade

Nesta fase do projeto, é chegado o momento de finalizar a iteração. Além disso, deve haver o preenchimento completo da TAT [EAS 03] (campos: tempo real e status).

Ainda nesta fase, os testes de aceitação [BEC 00] que foram definidos junto ao cliente, bem como os testes de usabilidade (a serem vistos com os usuários do sistema) devem ser aplicados para verificar se os objetivos de usabilidade, listados no início do projeto, estão sendo alcançados na interface.

[10] Basicamente, este artefato permite monitorar o andamento do projeto em torno da coleta de algumas métricas (e.g., número de classes desenvolvidas, número de testes realizados com erro ou sucesso, número de linhas de código, etc.).

4.5 Considerações finais do capítulo 4

Foi apresentada, neste capítulo, a proposta de uma metodologia para a concepção de sistemas de software centrada no usuário, XPU. Da mesma forma que alguns dos principais processos de desenvolvimento de aplicações da ES, essa conduta de desenvolvimento de sistemas iterativa e incremental contempla alguns dos princípios e valores fundamentais da ES como, por exemplo: poder ser modificada ou ajustada de acordo com as necessidades da organização, ser extensível e de fácil compreensão. Entretanto, diferentemente de muitos outros, XPU propõe em sua composição algumas das melhores práticas do universo da IHC, evidenciando, portanto, a usabilidade como uma característica fundamental na qualidade dos produtos gerados por este processo.

Em sua estrutura, que abraça condutas ágeis de desenvolvimento de sistemas [BEC 00][MAU 02][JEF 02][EAS 03][MAN 03][SUT 01][SCW 95][COC 03a][COC 03b] e práticas de usabilidade [TUR 02], XPU é composto de fases que podem ser facilmente entendidas e aplicadas em um ambiente de projeto real. Nestas, é evidenciada a figura do usuário final como um parceiro ativo em todo o ciclo de desenvolvimento.

Inicialmente, não somente foi apresentada a estrutura geral deste processo, mas também seus fluxos gerais de trabalho. Em seguida, cada uma das fases do processo foi detalhada (descrição, atividades, artefatos e suporte ferramental sugerido) e analisada neste capítulo.

Capítulo 5

Uma Avaliação da Metodologia XPU

5.1 Considerações iniciais

No corrente capítulo é apresentada ao leitor uma proposta particular de validação para a metodologia de desenvolvimento de sistemas centrado no usuário XPU (*eXtreme Programming* + Usabilidade (MCI)). Inicialmente, serão expostos os objetivos gerais deste experimento de comprovação além de aspectos mais específicos que o nortearão. Em seguida, importantes informações relacionadas à amostra que foi utilizada, pessoal envolvido para participar deste experimento, mecanismos de abordagem para coleta dos dados e, sobretudo, a análise e significado das informações obtidas, serão apresentadas.

5.2 Introdução

É recomendado que o modelo proposto pudesse ser submetido a algum tipo de validação. Este procedimento tem como meta avaliar, ao final, se os objetivos e especificidades definidos no início do trabalho foram alcançados. Neste caso, foi planejado e preparado um ambiente de validação visando confirmar a eficácia da metodologia XPU proposta. Ademais, a validação foi centrada na medição do impacto da integração do Método para Concepção de Interfaces (MCI) ao processo baseado em *eXtreme Programming* (XP). Inicialmente, optou-se por implantá-la em um projeto de software real (com cronogramas e equipe de desenvolvimento reais) visando, assim, verificar seu comportamento, eficácia, e aceitação por parte dos envolvidos no projeto.

Outrossim, além da avaliação em si, um outro aspecto fundamental deve ser enfatizado: a participação e *feedback* contínuos dos envolvidos no experimento e a

análise dos resultados coletados resultaram numa oportunidade ímpar de aprimorar e, dessa forma, amadurecer a metodologia sugerida.

Conforme dito anteriormente, antes de implantar e validar a metodologia, alguns objetivos tiveram de ser definidos. Na próxima seção, serão explicitados os objetivos gerais e, na seguinte, apresentaremos aspectos mais específicos que nortearam o experimento de validação.

5.3 Objetivos gerais da validação

A proposta de validação teve como objetivos gerais não somente refinar o processo XPU, mas, sobretudo, verificar sua aceitação e mensurar sua eficácia em um ambiente real de projeto de sistemas de software.

5.4 Aspectos específicos a serem observados na validação

O primeiro passo, depois de definidos os objetivos gerais que se pretendiam atingir, consistiu no levantamento de alguns aspectos importantes a serem observados e perseguidos na aplicação da metodologia no cenário de validação. Tais aspectos, mais específicos, guiaram o experimento de comprovação e favoreceram o aparecimento de respostas adequadas para o tema principal de nosso trabalho – maximizar a integração entre as áreas de Interação Homem-Computador (IHC) e Engenharia de Software (ES).

Antes de expor estes aspectos, é crucial enfatizar que o levantamento destas especificidades foi inspirado através de evidências empíricas percebidas em um período anterior a este momento de validação, onde, de forma peculiar, houve a oportunidade de acompanhar equipes de desenvolvimento que já estavam aplicando um processo na concepção de sistemas computacionais. Tal oportunidade de acompanhamento e de observação, documentada na forma de um relatório [VAS 03c], foi concretizada através da disciplina Estágio de Docência[11], que faz parte da grade curricular do curso de Pós-

[11] A disciplina, que deve ser exercida por alunos regularmente matriculados em cursos e programas de pós-graduação, nos níveis de mestrado e doutorado, compreende atividades acadêmicas com participação no ensino supervisionado em uma disciplina dos cursos de graduação da universidade relacionada com a estrutura curricular do curso de pós-graduação ou programa ao qual pertença o aluno.

graduação em Informática da Universidade Federal de Campina Grande. Naquela ocasião, a disciplina de graduação relacionada ao Estágio de Docência e cursada por mim foi **Laboratório de Engenharia de Software** (LES).

Em particular, é adequado frisar que as atividades da disciplina Estágio de Docência para alunos de mestrado devem ser desenvolvidas sob a responsabilidade de um professor (o professor encarregado da disciplina de graduação envolvida) e supervisionada pelo orientador do aluno. Estas duas funções - professora e supervisora - foram exercidas por uma de minhas orientadoras; a Professora Francilene Procópio Garcia, do Departamento de Sistemas e Computação da Universidade Federal de Campina Grande.

Finalmente, foram definidos alguns aspectos a serem perseguidos no momento da validação da metodologia, como, por exemplo:

1) **Eficiência e eficácia da metodologia sugerida**: a equipe de desenvolvimento, incluindo a participação do cliente e do usuário, devem se posicionar acerca da eficácia do processo proposto, da qualidade da interação com o sistema e da interface produzida.
2) **Esforço, complexidade e pertinência dos artefatos de usabilidade**: deve-se relatar tais aspectos coletados junto à equipe de desenvolvimento sobre os "novos" artefatos de usabilidade sugeridos por XPU.
3) **Modificações realizadas nos protótipos iniciais gerados:** uma vez iniciada a aplicação da metodologia, esta etapa tem como meta identificar se as atividades desenvolvidas têm tido algum impacto em versões geradas do sistema de software antes do conhecimento do novo processo.
4) **Complementações trazidas pelos artefatos de usabilidade:** esta atividade analisa os benefícios da integração de artefatos de usabilidade quando comparados com os processos tradicionais de ES e, em particular, com o processo XP.
5) **Aceitação da metodologia sugerida:** uma das principais metas da organização deste cenário de validação é observar o comportamento do cliente e membros da equipe de desenvolvimento - detectar suas reações e opiniões de forma espontânea, no decorrer da rotina de atividades.

6) **Eventuais desvios:** este aspecto visa identificar algum impacto negativo após a inclusão dos artefatos de usabilidade no ambiente de projeto (e.g., "estouro" no cumprimento dos prazos de entrega inicialmente acordados com o cliente).

5.5 Descrição do cenário de validação

Nesta seção, é apresentada uma descrição detalhada do cenário de validação escolhido para implantação da metodologia XPU.

Passada a fase de definição dos aspectos mais específicos a serem considerados no experimento de validação, foi necessária a escolha de um local adequado onde a metodologia proposta pudesse ser implantada e observada. Diante de tal necessidade e, com base em atividades expostas na seção anterior, optou-se pela definição de um cenário de validação na mesma disciplina de graduação citada anteriormente: LES.

A disciplina citada não somente se apresenta como uma das mais importantes da grade curricular do curso de Ciência da Computação da Universidade Federal de Campina Grande, mas é também a primeira a possibilitar aos seus alunos o desenvolvimento de um projeto real de software. Diante disso, a disciplina tornou-se, automaticamente, uma boa candidata para o experimento, isso porque os grupos de alunos poderiam experimentar não apenas um processo para concepção de sistemas, mas um que contemplasse práticas de IHC em seu fluxo de atividades.

A pertinência da disciplina de LES pode ser confirmada na seção seguinte, onde serão apresentadas: sua descrição geral, relevância e características.

5.5.1 Laboratório de Engenharia de Software

Nesta seção, é apresentada uma descrição da disciplina Laboratório de Engenharia de Software, destacando algumas de suas características relevantes ao experimento.

A disciplina Laboratório de Engenharia de Software é obrigatória para os graduandos em Ciência da Computação do Departamento Sistemas e Computação da

Universidade Federal de Campina Grande. A disciplina é de 30 (trinta) horas (o que corresponde a dois créditos) e tem como pré-requisito a disciplina Engenharia de Software I (onde os alunos recebem a fundamentação teórica necessária acerca de processos de desenvolvimento).

O objetivo da disciplina de LES é aplicar um processo iterativo e incremental completo para a construção de aplicações computacionais. Normalmente, a conduta de desenvolvimento utilizada nos projetos da disciplina resulta de uma adaptação bem definida de XP, incluindo etapas de planejamento, análise de requisitos, projeto, codificação e testes. No âmbito da estrutura do processo já adotado em LES, sugere-se um processo com três *releases* de duas iterações cada, o qual deve ser documentado e medido. Diversas atividades são planejadas e definidas em LES: artefatos a serem produzidos, métricas a serem colhidas, verificação e garantia de integridade do sistema.

Em síntese, LES é uma disciplina orientada à execução de projetos concretos de desenvolvimento de software. Uma das maiores contribuições da disciplina é permitir que as equipes de alunos implantem um processo de desenvolvimento em projetos com clientes reais.

5.5.2 Público-alvo envolvido na validação

Definido o local do experimento, foi o momento de considerar o público-alvo da validação. Nesta população, inicialmente, foram incluídos todos os grupos de alunos que estivessem matriculados na disciplina de LES no período 2003.1. Entretanto, para que fosse possível observar o impacto da adoção de XPU sem que o processo original adotado na disciplina fosse substituído por XPU completamente, foi selecionado um grupo de projeto composto de três alunos (além do cliente do projeto) escolhido de forma aleatória, sem limitações de sexo ou faixa etária.

O projeto desenvolvido pelo grupo de alunos tinha o objetivo de conceber uma ferramenta *web* de suporte ao planejamento e gerenciamento de um projeto de software – a ferramenta *eXtreme Management* (XMan). Tal ferramenta é destinada a desenvolvedores e, sobretudo, gerentes de projetos de sistemas. Basicamente, todo o projeto foi orientado à luz de tecnologias *Java* [JAV 03]: *Ant* [ANT+ 03] e *Eclipse* [ECL 03] no desenvolvimento, *JUnit* [JUN 03] e *Cactus* [CAC 03] dando suporte às

atividades de testes e *CVS* [CVS 03] como principal ferramenta de controle de versões. No âmbito do porte do sistema desenvolvido, o mesmo foi finalizado com um total de 105 (cento e cinco) classes[12] e mais de 60 (sessenta) páginas JSP[13].

A aplicação do experimento foi comunicada aos alunos da disciplina e, nesta ocasião, foram observadas se ocorreriam mudanças nos comportamentos destes, indicando algum tipo de insatisfação ou rejeição, o que não foi verificado. Outrossim, durante toda a aplicação da metodologia, houve sempre conversas informais com os alunos, como forma de identificar possíveis dificuldades, sucessos, esclarecimentos, enfim, os sentimentos que os cercavam com relação à introdução de práticas de usabilidade em seu projeto.

5.5.3 Abordagem utilizada na coleta de informações

O próximo passo no planejamento do experimento consistiu em definir os mecanismos de abordagem para a coleta dos dados. Nos casos do cliente e da equipe de alunos, optou-se pela aplicação de dois questionários com perguntas subjetivas e objetivas. Antes de aplicá-los, os questionários foram analisados e melhorados, sofrendo alterações para potencialização do experimento de validação. Somente após várias alterações realizadas nos questionários originais, novos modelos, mais objetivos e sucintos, foram produzidos. A partir daí, começaram a ser aplicados com a amostra definitiva da pesquisa.

Os dois modelos de questionários: um destinado à equipe de desenvolvimento e outro ao cliente do projeto, no formato final, constituem o Anexo F desta dissertação.

Com tais questionários, a participação no projeto da equipe e a abordagem junto aos membros do time e cliente para coleta de dados foram favorecidas. Este contato direto proporcionou uma relação mais estreita entre o investigador e o fenômeno estudado.

[12] Blocos de construção fundamentais presentes na tecnologia *Java*.
[13] *Java Server Pages* - uma tecnologia *Java* para construção de páginas *web* de conteúdo dinâmico.

5.6 Análise dos resultados

Após a coleta dos dados, foi realizada uma estratificação - momento em que foi feita a análise quantitativa e, em seguida, para uma melhor visualização do leitor, estes resultados foram expressos em gráficos e tabelas a serem analisados de forma qualitativa, visando, portanto, compreender o significado de cada resultado obtido.

A partir dos procedimentos descritos, foi possível coletar dados e analisar situações de suma importância dentro dos objetivos propostos que evidenciaram o fenômeno estudado e comprovaram os aspectos esperados. Sendo assim, o estudo apresentou-se relevante na área dos profissionais de ES e IHC, bem como no amadurecimento da metodologia XPU.

5.6.1 Análise sobre a compreensão da equipe de desenvolvimento

Iniciou-se com a análise dos dados apurados a partir da aplicação do questionário da equipe em cada um dos membros do time de desenvolvimento. O modelo do referido questionário encontra-se no Anexo F.

Os alunos que constituíam a equipe de desenvolvimento eram em número de 3 (três) e as perguntas foram direcionadas para identificar informações relacionadas aos artefatos de usabilidade, bem como sua compreensão geral com relação ao projeto do sistema.

Os dois modelos de questionários aplicados com a equipe e com o cliente do projeto constam de uma primeira parte introdutória que trata da assiduidade de tais envolvidos nas reuniões em todo o projeto; neste caso, os dados a seguir mostrados tratam das respostas do time de desenvolvimento. Esta informação introdutória foi utilizada, com efeito, na análise das respostas dadas pela equipe nas perguntas seguintes (que, por sua vez, abordaram diretamente o objeto da pesquisa). Isso porque, caso um membro não tivesse participado ativamente do projeto, a veracidade de suas respostas poderia ser afetada.

A primeira pergunta do questionário se subdividiu em perguntas 1.1 e 1.2. A pergunta 1.1 levantava a participação do membro no projeto. Na pergunta 1.2, o

respondente deveria indicar a participação do cliente durante as reuniões de projeto. Conforme citado na seção 5.5.1, devido ao arranjo do processo na disciplina (onde um *release* apresentava uma janela de tempo de quatro semanas), foram acordadas doze reuniões entre a equipe e o cliente do projeto. Os dados relacionados à primeira subdivisão são demonstrados na Tabela 5, a seguir:

Membro da equipe	Freqüência de respostas	Participação
1	1	12
2	1	12
3	1	11

Tabela 5: Tabela Demonstrativa da Assiduidade da Equipe no Projeto.

A partir da Tabela 5, observou-se que dois dos três membros da equipe indicaram participação integral nas doze reuniões de projeto com o cliente e somente um membro, diferenciando-se dos demais, indicou participação em onze dos doze encontros realizados. Julgou-se, portanto, que a maior parte da equipe participou integralmente em todas as fases projeto e que a veracidade das respostas dadas pelo time foi favorecida.

Conforme já foi dito, a segunda subdivisão da questão objetivou identificar, na opinião da equipe, a assiduidade do cliente nas reuniões semanais de projeto. Os dados são demonstrados na Tabela 6:

Membro da equipe	Freqüência de respostas	Participação do Cliente
1	1	12
2	1	12
3	1	12

Tabela 6: Tabela Demonstrativa de Assiduidade do Cliente na Visão da Equipe.

A partir da Tabela 6, observou-se que três membros da equipe informaram uma participação integral do cliente. Concluiu-se que, segundo opinião da equipe, o cliente foi ativo durante o processo de desenvolvimento.

Considerando que a mesma pergunta foi feita para a equipe e o cliente do projeto, foi estabelecida uma comparação entre as respostas dadas por ambos na análise do questionário de clientes. O resultado desta comparação será apresentado mais adiante, quando serão apresentados os dados do questionário de clientes.

Concluídas as análises feitas nas tabelas anteriores, passou-se à segunda questão, que buscou avaliar a eficiência da metodologia. Ao contrário da questão anterior, foram oferecidos quatro tipos de alternativas (*ótima, boa, regular e ruim*) e as que foram assinaladas estão expostas na Tabela 7:

Membro da equipe	Freqüência de respostas	Eficiência do modelo
1	1	Boa
2	1	Boa
3	1	Boa

Tabela 7: Tabela da Eficiência da Metodologia XPU.

De acordo com a tabela acima, foi possível constatar uma distribuição homogênea dos dados, havendo um consenso total da equipe para uma boa eficiência da metodologia. Não obstante o não aparecimento nesta amostra de uma ótima eficiência, é importante frisar, neste momento da validação, que nenhum membro do time de desenvolvimento apontou uma eficiência regular ou mesmo insatisfatória.

As opções não assinaladas não foram contabilizadas na Tabela 7.

Visando aprofundar mais as informações, foi questionado, na terceira questão, sobre o esforço na execução de cada uma das atividades de usabilidade propostas em XPU. Atribuindo pesos de 1 a 6, os membros podiam classificar as atividades de usabilidade quanto a seu tempo de conclusão. Foi atribuído o peso 1 à atividade de menor duração. Foram apurados os seguintes dados, mostrados, abaixo, na Tabela 8:

	Atividades de Usabilidade e os Esforços na Execução (pesos)					
Equipe	**Necessidades do Cliente**	**Objetivos de Usabilidade**	**Perfil do Usuário**	**Modelagem da Tarefa**	**Modelagem da Interação**	**Testes de Usabilidade**
Membro 1	2	1	3	6	5	4
Membro 2	1	2	4	6	3	5
Membro 3	1	4	2	5	6	3

Tabela 8: Tabela da Indicação de Esforços nos Artefatos de Usabilidade.

Com base em tais dados, foi gerado um gráfico do esforço na execução das atividades de usabilidade, conforme pode-se ver no Gráfico 1:

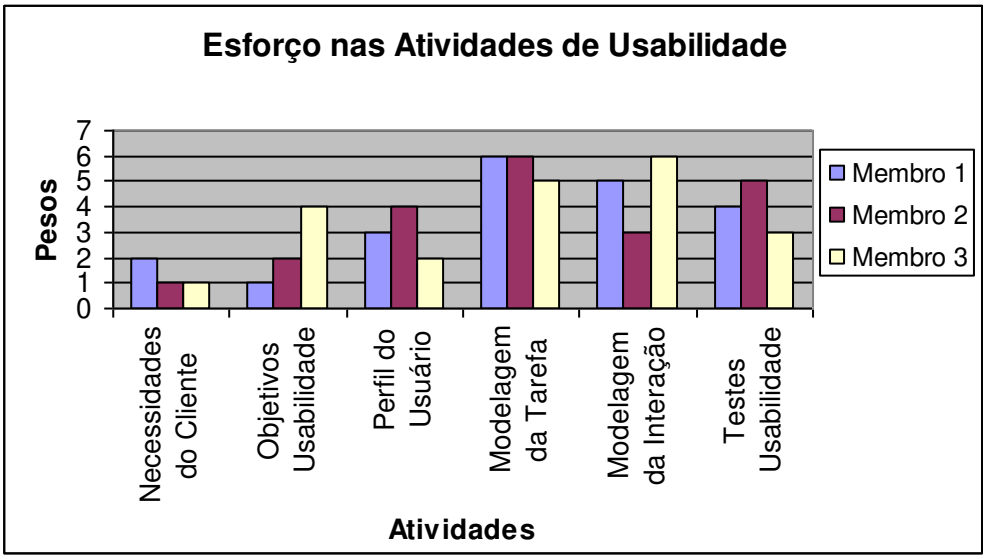

Gráfico 1: Esforço na Execução das Atividades de Usabilidade.

A disposição dos dados no gráfico acima indica que, na opinião da equipe, dentro das atividades de usabilidade apresentadas, a modelagem da tarefa do usuário foi a que consumiu o maior tempo de execução. Diante deste fato, foi pertinente a sugestão e inclusão de um suporte ferramental no processo para a realização de tal atividade. Já o levantamento das necessidades do cliente através de conversas iniciais refletiu a atividade de menor esforço.

Perguntou-se aos membros da equipe, na quarta questão, sua opinião sobre a complexidade presente em cada uma das atividades de usabilidade sugeridas no processo XPU e obteve-se os seguintes resultados:

	Complexidade na Execução das Atividades de Usabilidade					
Equipe	**Necessidades do Cliente**	**Objetivos de Usabilidade**	**Perfil do Usuário**	**Modelagem da Tarefa**	**Modelagem da Interação**	**Testes de Usabilidade**
Membro 1	1	2	4	6	5	3
Membro 2	1	3	2	4	6	5
Membro 3	1	2	3	5	4	6

Tabela 9: Tabela da Complexidade na Execução das Atividades de Usabilidade.

Da mesma forma que a questão anterior, foram atribuídos pesos de 1 a 6, sendo o peso 1 indicando a menor complexidade. Os dados apresentados na tabela acima foram mapeados para um gráfico, conforme pode-se ver no Gráfico 2:

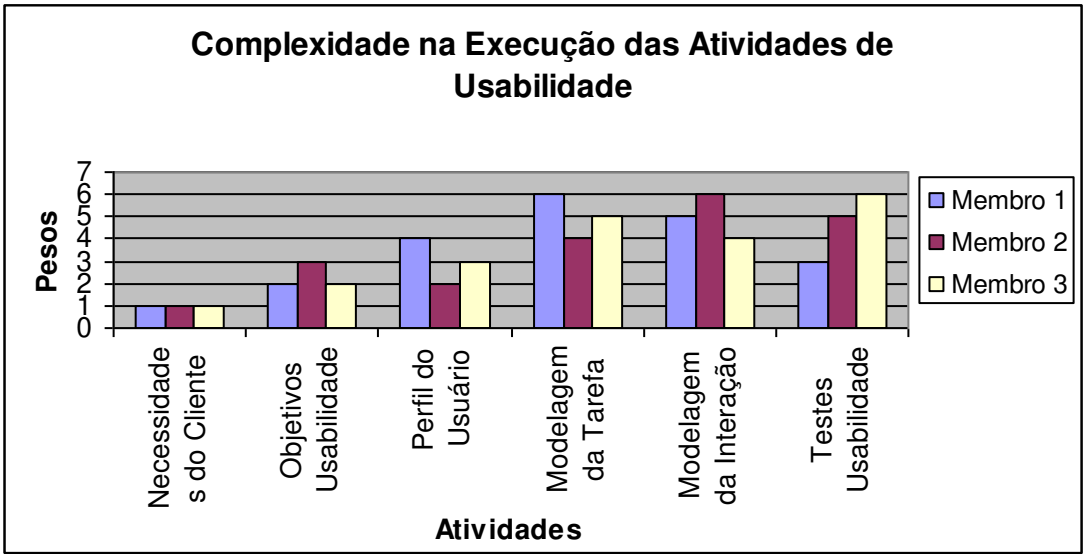

Gráfico 2: Gráfico da Complexidade na Execução das Atividades de Usabilidade.

Analisou-se, a partir do Gráfico 2, que há uma predominância da complexidade tanto na modelagem da tarefa do usuário, quanto na da interação. Considerando a disposição dos dados, observou-se um empate entre tais atividades, pois, em um somatório realizado dos pesos atribuídos às mesmas, obtêm-se o valor de quinze pontos. Concluiu-se, portanto, que a amostra delimitada indicou ambas, modelagem da tarefa e interação, como as duas atividades de maior complexidade quando comparadas às demais da mesma área sugeridas no processo XPU. Também neste quesito pode-se afirmar que a sugestão e inclusão de um suporte ferramental no modelo proposto para a realização da modelagem da tarefa foi pertinente. Porém, é importante enfatizar que a complexidade relatada pela equipe na atividade de modelagem da interação poderia ser diminuída, caso houvesse um suporte ferramental específico no método MCI para a realização de tal atividade.

A quinta pergunta indagou à equipe sobre a completude e benefícios de duas atividades de IHC propostas em XPU: Questionário do Perfil do Usuário e a conversa para Levantamento das Necessidades do Cliente. Mais especificamente, esta questão queria confirmar se o Documento de Visão (artefato indicado por processos típicos de ES) poderia ser complementado com tais atividades de usabilidade. Apuraram-se os seguintes dados mostrados na Tabela 10:

Membro da equipe	Freqüência de respostas	Complementação
1	1	Sim
2	1	Sim
3	1	Sim

Tabela 10: Tabela de Perfil do Usuário e Levantamento das Necessidades do Cliente versus Documento de Visão.

Conforme evidenciado na Tabela 10, os membros da equipe foram unânimes em indicar que as duas atividades de usabilidade mencionadas na questão complementam o artefato de ES. Além de tal complementação, este fato sugere que o Documento de Visão, por si só, não é suficiente para levantar o perfil do usuário e as necessidades iniciais do cliente de forma adequada.

A sexta questão é considerada das mais importantes do questionário, já que busca levantar o impacto das atividades de usabilidade sobre o novo protótipo gerado pela equipe de alunos a partir do conhecimento do XPU. Tal pergunta foi formulada para o time com o objetivo de avaliar o impacto das atividades de usabilidade no protótipo inicial da interface com o usuário, gerado antes do conhecimento da metodologia. Conforme já foi dito no início deste capítulo, uma vez que o experimento foi iniciado na segunda fase de desenvolvimento do projeto, já havia um primeiro protótipo da interface produzido. Conforme podemos observar, na Tabela 11, os seguintes dados foram colhidos:

Membro da equipe	Freqüência de respostas	Modificações
1	1	Muitas alterações
2	1	Alterações foram feitas
3	1	Muitas alterações

Tabela 11: Tabela da Análise da Equipe Sobre Modificações no Protótipo.

Tomando como base a tabela acima, gerou-se um gráfico que apresenta a opinião da equipe sobre as modificações realizadas em tal protótipo, conforme pode-se ver no Gráfico 3:

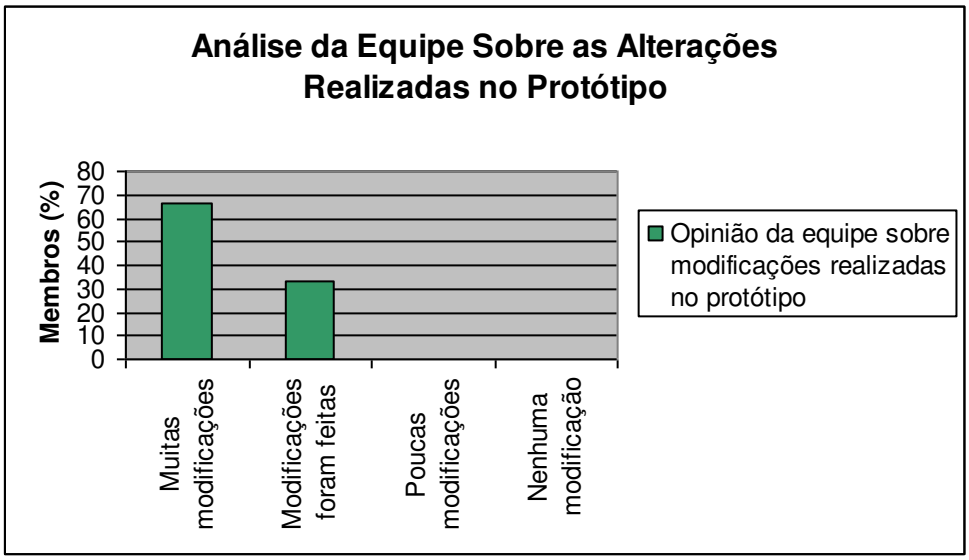

Gráfico 3: Gráfico da Análise da Equipe sobre as Alterações no Protótipo.

A disposição das informações mostra que dois dos três membros da equipe realizaram diversas alterações no protótipo, enquanto que um membro, diferenciando-se dos demais, informou um número normal de modificações feitas em tal protótipo.

Diante deste fato, podemos comprovar o impacto positivo da metodologia (em particular, de todos os artefatos de usabilidade sugeridos) no *modus operandi* da equipe, a qual, em sua maioria, afirmou realizar um número expressivo de modificações corretivas no protótipo da interface.

O próximo passo foi avaliar a aprovação e aceitação geral de uma ferramenta da área de IHC por parte da equipe de desenvolvimento. Conforme foi citado no Capítulo 4, com o objetivo de facilitar a construção do modelo da tarefa, é indicada uma ferramenta na metodologia XPU: o *Euterpe*. Na sétima questão, a pertinência de tal ferramenta foi indagada, conforme pode-se observar os dados apurados na Tabela 12:

Membro da equipe	Freqüência de respostas	A ferramenta foi útil?
1	1	Sim
2	1	Sim
3	1	Sim

Tabela 12: Tabela da Aceitação da Ferramenta Euterpe no Processo.

Conforme evidenciado na tabela acima, a equipe foi unânime, nesta questão, em confirmar os ganhos de produtividade e economia de tempo que o ambiente *Euterpe*

proporcionou na geração do artefato modelo da tarefa. Além de tal confirmação, este fato demonstrou a sua facilidade de utilização e obtenção.

Na oitava questão, a equipe era questionada sobre as tarefas do usuário levantadas durante a modelagem da tarefa. De forma não tão abrangente, tal questão tinha o objetivo de não somente confrontar tais tarefas com as tarefas de programação da ES ("quebradas" a partir das *user stories* na fase de planejamento da iteração), mas descobrir se estas tarefas do modelo da tarefa podem ser utilizadas para levantar os requisitos funcionais de um sistema de software. Os dados podem ser visualizados na Tabela 13:

Membro da equipe	Freqüência de respostas	Confronto
1	1	Sim
2	1	Sim
3	1	Sim

Tabela 13: Tabela da Modelagem da Tarefa versus Requisitos Funcionais.

Considerando a disposição dos dados, pode-se notar que os membros da equipe foram unânimes em afirmar haver um mapeamento entre as tarefas do usuário e as de programação obtidas a partir das *user stories.* Diante deste fato, concluiu-se ser adequada a utilização das tarefas do usuário resultantes da atividade de modelagem da tarefa da IHC para levantar os requisitos funcionais de uma aplicação computacional. Este fato também foi recomendado no estudo de Cordeiro [COR 03].

A penúltima pergunta do questionário das equipes também trata da utilização da modelagem da tarefa no projeto. Entretanto, sob outra perspectiva: a questão visa confirmar se a modelagem da tarefa pode ajudar os desenvolvedores no projeto a elaborar um *pré-contrato das classes envolvidas* no sistema. Antes de apresentar os resultados obtidos, é importante que o leitor tenha conhecimento deste último termo frisado.

É sabido pela comunidade de ES que uma das principais atividades realizadas durante um projeto de software diz respeito à representação bem definida, na forma de um artefato, dos objetos que irão compor o sistema. Tal artefato é conhecido como Diagrama de Classes. Normalmente, segundo Larmam [LAR 00], cada uma das classes descritas neste diagrama possui um conjunto de operações (ou métodos) que, por sua vez, permitem que os objetos possam interagir entre si na realização de uma dada

funcionalidade do sistema. A este conjunto de operações é dado o nome de *contrato* ou *interface* de uma classe.

Neste sentido, de maneira mais específica, a nona questão quis saber da equipe se as tarefas-folhas, encontradas na modelagem da tarefa, podiam ser mapeadas para métodos a serem enxertados nos contratos das classes. Apuraram-se os seguintes resultados, mostrados, abaixo, na Tabela 14:

Membro da equipe	Freqüência de respostas	Existe mapeamento?
1	1	Sim
2	1	Sim
3	1	Sim

Tabela 14: Tabela da Modelagem da Tarefa versus Contrato de Classes.

Toda a equipe afirmou haver um mapeamento entre as tarefas do usuário e os contratos das classes envolvidas. Com base neste fato, julgou-se que, após a construção do modelo da tarefa, o time pode mapear as tarefas-folhas para operações (ou métodos) no sentido de facilitar a criação do artefato diagrama de classes e, assim, favorecer a produtividade no projeto.

Quando se passou à análise do **espaço reservado para comentários e sugestões adicionais** no questionário da equipe, observou-se, de forma geral, que "*XPU serviu como um importante aliado na melhoria da interface",* uma vez que esta *"apresentava baixa qualidade antes do processo e foi melhorada"*. Como sugestão, a equipe informou ainda que o experimento de validação poderia ter sido feito desde a primeira fase do projeto, fazendo dessa forma, com que o time tivesse tido *"mais tempo para conhecer e gerar os artefatos"*.

5.6.2 Análise sobre a compreensão do cliente

Passada a análise das informações colhidas a partir dos questionários da equipe de desenvolvimento, foi o momento de entregar uma série de questões ao cliente do projeto. O questionário do cliente seguiu o mesmo padrão daquele aplicado com os membros do time, onde foram feitas as perguntas que abordaram diretamente o problema de pesquisa. O modelo de tal questionário também se encontra no Anexo F.

A amostra utilizada no experimento, neste caso, consistiu de um cliente experiente em gerência de projetos de software. Este, opinou sobre o projeto e, de forma específica, sobre a interface do sistema e seu relacionamento com o time de desenvolvimento.

Da mesma forma que no questionário da equipe, a primeira pergunta se subdividiu em perguntas 1.1 e 1.2. A pergunta 1.1 questionou sobre a participação do cliente projeto. Na pergunta 1.2, tal cliente deveria indicar a participação da equipe durante as reuniões de projeto. Os dados relacionados à primeira subdivisão são demonstrados na Tabela 15, a seguir:

Cliente	Freqüência de respostas	Participação
1	1	12

Tabela 15: Tabela da Assiduidade do Cliente no Projeto.

A partir da tabela acima, observa-se uma participação do cliente em doze reuniões do projeto. Ao considerar a assiduidade de tal cliente indicada pelo time de desenvolvimento, no início da seção anterior, observou-se uma consistência a respeito desse dado.

A segunda subdivisão da questão visa identificar, na opinião do cliente, a assiduidade dos membros do time nas reuniões de projeto. Apuraram-se os seguintes dados:

Cliente	Freqüência de respostas	Participação
1	1	12

Tabela 16: Tabela de Assiduidade da Equipe na Visão do Cliente.

Conforme evidenciado na tabela acima, o cliente indicou assiduidade integral da equipe durante todo o projeto. Ao confrontar essa informação com a apresentada na Tabela 5, conclui-se, neste caso, uma conformidade de informações.

Concluídas as duas análises feitas acima, passou-se a segunda pergunta do questionário. Tal questão buscou avaliar se a metodologia, de forma específica a inclusão dos artefatos de usabilidade no projeto, afetou as datas de liberação dos *releases* ou estes últimos continuaram sendo entregues dentro do prazo inicialmente acordado.

Nesta pergunta, o cliente informou que as versões de software continuaram sendo entregues dentro dos prazos. Diante deste fato, conclui-se que, mesmo com a inclusão dos artefatos de usabilidade no processo XP, o volume do processo resultante se manteve adequado mantendo sua aplicação em um ambiente de projeto viável.

Perguntou-se ao cliente, na terceira questão, sobre sua aceitação geral da interface produzida pelo time. De forma específica, a questão buscava descobrir se houve conformidade da interface com o sistema entregue pelo grupo de alunos com o que foi conversado entre estas duas partes, ao longo de todo o desenvolvimento.

Verificou-se, com base na resposta dada pelo cliente, não apenas que o projeto liberado atendeu as expectativas do cliente, mas, no âmbito do processo XPU, a contribuição de todos os artefatos de usabilidade sugeridos.

A quarta questão indagou o cliente sobre a correspondência entre os objetivos de usabilidade levantados junto com os alunos e a interface final produzida. Conforme pudemos observar, no Capítulo 4, esta atividade de IHC sugerida na fase de conversa com o cliente deve ter a aprovação final deste último. Neste sentido, tal quesito desejava verificar se a lista dos objetivos levantados entre a equipe e o cliente realmente havia sido mapeada para a interface do sistema.

De acordo com a resposta dada pelo cliente, foi dada uma atenção adequada por parte do time nos objetivos de usabilidade, de forma que cada um dos itens da lista de tais objetivos foi corretamente mapeado para a interface.

O quinto quesito do questionário do cliente se mostrou o mais importante do questionário, já que fez uma abordagem direta à interface produzida pela equipe. Tal pergunta foi formulada para o cliente com o objetivo de não somente avaliar o impacto das atividades de usabilidade na interface, mas descobrir sua impressão geral acerca desta parte tão importante do sistema.

A questão acima mencionada subdividiu-se em perguntas 5.1, 5.2, 5.3 e 5.4. A pergunta 5.1 indagava se o cliente se encontrava satisfeito com a interface do sistema; a 5.2 tratava da facilidade de uso do produto de software através da interface projetada pelo time de desenvolvimento; o sub-item 5.3 perguntava ao cliente sobre como foi feita

a entrega da interface final e, finalmente, o 5.4 buscava descobrir se houve melhorias na interface ao longo das iterações no projeto.

A partir das respostas dadas pelo cliente, foi possível verificar:

- **no sub-item 5.1,** o cliente afirmou estar satisfeito com a interface do usuário final com o sistema. Diante deste fato, sugere-se que a metodologia XPU e, de modo particular, seus artefatos de usabilidade se mostraram eficientes e relevantes na produção de uma interface que agradasse ao cliente do projeto;
- **o sub-item 5.2** revelou que a interface final apresentada pelo time é de fácil utilização;
- **a pergunta do sub-item 5.3** expõe, segundo o cliente, que a interface com o usuário foi entregue funcionando adequadamente;
- **o sub-item, o 5.4** descreve que a interface apresentou melhorias ao longo do projeto.

Quando se passou a uma análise do **espaço reservado para comentários e observações adicionais** no questionário do cliente, observou-se, de forma geral, que "*as questões de usabilidade foram abordadas e o time de desenvolvimento dedicou-se bastante".*

5.7 Considerações finais do capítulo 5

Foi apresentada, neste capítulo, uma proposta particular de validação para a metodologia de desenvolvimento de sistemas centrada no usuário XPU. Primeiramente, foram esclarecidos alguns objetivos gerais deste experimento de validação e aspectos que nortearam este evento. Em seguida, foi fornecida uma descrição detalhada de tal momento de comprovação, descrevendo o local escolhido, amostra utilizada na coleta de informações, mecanismos de abordagem de tal coleta e análise de cada um dos resultados obtidos.

Capítulo 6

Conclusões

A utilização das técnicas de Interação Homem-Computador (IHC) está deixando de ser uma eventualidade no contexto de projetos computacionais para se tornar uma realidade, agregando valor competitivo à indústria de software. Fatores tais como: (1) aumento na produtividade do usuário, (2) aumento da satisfação do usuário, (3) diminuição nos custos de manutenção do sistema, (4) aumento da qualidade da interface, (5) redução no tempo de treinamento do usuário e (6) redução nas mudanças em fases avançadas de projeto; têm influenciado engenheiros e desenvolvedores de sistemas a reconhecerem a importância da inclusão de tais técnicas em seus projetos. Todavia, a ausência de um modelo de desenvolvimento que integre práticas de usabilidade com os processos de desenvolvimento de software adotados pela Engenharia de Software (ES) tem se constituído num desafio ao crescimento e consolidação das condutas de IHC em muitas organizações produtoras de software.

Este trabalho procurou, em primeiro lugar, apresentar um panorama acerca da utilização das técnicas de IHC na indústria de software, destacando não somente a visão tradicional da usabilidade ainda disseminada entre a comunidade de ES, mas, sobretudo, o impacto positivo que as atividades de IHC exercem na qualidade e aceitação de um sistema percebidas pelo seu usuário. Na seqüência, tomando-se como base a exposição e discussão de um conjunto de princípios e valores fundamentais defendidos pelas áreas de IHC e ES, buscou-se definir, alinhado com a realidade de mercado, um modelo para a concepção de sistemas de software centrado no usuário. Ao final, foi apresentado um experimento de validação desenvolvido com o intuito de avaliar a eficiência de tal modelo e sua aceitação em um ambiente de projeto de software real.

6.1 Contribuições

Foi apresentada, neste trabalho, a proposta de um modelo para o desenvolvimento de sistemas centrado no usuário, o XPU (*eXtreme Programming* + Usabilidade (MCI)). O seu desenvolvimento surgiu de um estudo inicial da utilização das técnicas de IHC em projetos de sistemas de software. Neste estudo, verificou-se algumas dificuldades e problemas em relação não apenas a aplicação de tais técnicas por boa parte da comunidade de ES, mas também, de forma não tão abrangente, a integração de algumas destas práticas de usabilidade com as modernas metodologias de desenvolvimento de software da ES.

Após uma análise mais detalhada de algumas metodologias da área de IHC, onde foram verificados os modelos utilizados, seus fluxos de trabalho e suporte ferramental (quando disponível), constatou-se que, de fato, tais metodologias poderiam agregar valor aos processos de desenvolvimento de software da ES, amadurecendo-os. Outrossim, a integração adequada entre os domínios de IHC e ES sob a forma de um modelo de desenvolvimento traria a reboque impactos positivos no tocante a qualidade da interface produzida e à interação do usuário com o sistema.

XPU é um modelo para o desenvolvimento de aplicações computacionais centrado no usuário. Trata-se de um processo de desenvolvimento baseado em práticas de metodologias ágeis e leves (*lightprocess*), acolhendo algumas condutas do *eXtreme Programming* (XP) em sua estrutura. Além disso, técnicas de IHC são sugeridas em sua composição, segundo uma adaptação ao Método para Concepção de Interfaces (MCI), uma metodologia da área de IHC proposta e disseminada pelo Grupo de Interfaces Homem-Máquina da Universidade Federal de Campina Grande (GIHM/UFCG).

No âmbito da ES, visto que a solução proposta se utiliza de uma adaptação bem definida de XP em sua estrutura, a documentação extensiva, a execução rígida de medições e a previsibilidade em um projeto de software abrem espaço para uma abordagem minimalista, com pouca burocracia envolvida no desenvolvimento e equipes ágeis, capazes de "abraçar" e reagir às mudanças rapidamente e de entregar software valioso ao cliente, num ritmo constante.

Uma outra contribuição fundamental do presente trabalho foi uma maior integração entre o GIHM e o Grupo de Engenharia de Software (GES), ambos da Universidade Federal de Campina Grande. Em particular, o MCI é um método do GIHM e foi, conforme citado anteriormente, adaptado e utilizado na construção da solução proposta. Além do aumento de sinergia entre os grupos, a elaboração do modelo sugerido possibilitou: (1) a continuidade (extensão) do trabalho iniciado com o desenvolvimento do MCI pelo GIHM e (2) posicionar melhor o MCI junto à indústria de software, ao combiná-lo com um processo de desenvolvimento da ES mais disseminado no mercado e entre os desenvolvedores.

Um diferencial encontrado em XPU é que o modelo possui seu foco nas pessoas ao invés do processo. Na solução proposta, em praticamente todas as suas fases, é evidenciada a figura do usuário como um parceiro ativo e tomador de decisões em todo o ciclo de desenvolvimento do produto e não como um indivíduo passivo. Conforme visto no Capítulo 2, o usuário pode ser visto como um "perito" em relação ao domínio da aplicação, ao passo que os engenheiros e desenvolvedores de software são porta-vozes do usuário ou cliente no mundo técnico. Portanto, exercer os papéis e atribuições de cada um no projeto é crucial.

Vale ressaltar também que um modelo de desenvolvimento de software que integra práticas de IHC e ES permitirá um maior contato por parte da comunidade de ES com algumas das principais técnicas do universo da IHC, de forma que tal comunidade poderá conhecer melhor a eficiência e os ganhos que alguns artefatos de usabilidade trazem aos seus projetos. Para corroborar o que foi dito e, de forma breve, se tomarmos como exemplo a análise realizada neste trabalho em cima de uma atividade de IHC (Modelagem da Tarefa do Usuário), apresentada no Capítulo 5, obtiveram-se resultados importantíssimos, como: (1) não só as *user stories*, artefatos consolidados no universo da ES, mas também o modelo da tarefa pode ser utilizado para levantar os requisitos funcionais de um sistema de software, (2) uma vez que as tarefas-folhas presentes no modelo da tarefa podem ser mapeadas para operações (métodos) do sistema, tal artefato de usabilidade se mostra bastante útil na definição dos contratos das classes envolvidas no sistema (uma atividade da ES) e (3) auxilia a equipe numa melhor especificação da interface do sistema.

Por fim, ao contrário das soluções utilizadas por diversas organizações produtoras de software que utilizam modelos extremamente focados na implementação das estruturas internas de uma aplicação computacional [CON 99], o modelo proposto torna-se uma alternativa atrativa para as equipes de desenvolvimento de software que desejam obter não apenas sistemas portáveis, fracamente acoplados ou reutilizáveis, mas, sobretudo, endereçar técnicas de IHC em seus projetos e obter interfaces que reflitam os objetivos, as características e as necessidades do usuário.

6.2 Restrições

Existem algumas restrições por parte do modelo de desenvolvimento indicado, as quais devem ser ressaltadas neste momento. Entre elas, destacam-se:

- a abordagem minimalista do modelo proposto enfoca a agilidade na forte comunicação e participação do usuário do sistema. Este último deve estar ciente de que é parte integrante da equipe de desenvolvimento, face que possui responsabilidades importantes em todas as fases do processo. Portanto, para que o modelo de desenvolvimento de sistemas centrado no usuário proposto funcione, é necessário que o usuário seja especialmente dedicado ao projeto e possa tomar decisões rápidas para garantir o cronograma do mesmo;
- haja vista que a usabilidade é tradicionalmente entendida por boa parte da comunidade da ES como sendo somente o processo de avaliação da interface com o usuário (atribuição adequada de cores, fontes, etc.), o modelo proposto sugere uma mudança cultural profunda por parte da equipe de desenvolvimento (e também do usuário, conforme restrição anterior) ainda nos primeiros estágios do processo, o que nem sempre é fácil de alcançar. Assim, a curva de aprendizagem pode se tornar um pouco mais acentuada em alguns ambientes de projeto.
- conforme visto no experimento de validação (Capítulo 5), a modelagem da interação do usuário com o sistema foi apontada como uma das atividades de IHC de maior complexidade. Segundo Turnell [TUR 02], o modelo da interação é um importante artefato a ser gerado durante o processo de

desenvolvimento do sistema, uma vez que relaciona o domínio da tarefa com o domínio da interface. Neste sentido, percebe-se que tal complexidade poderia ser diminuída, caso houvesse um suporte ferramental específico no modelo proposto para a realização de tal atividade de IHC.

6.3 Trabalhos futuros

A conclusão deste trabalho permite identificar alguns encaminhamentos futuros, no sentido de aperfeiçoar a especificação do modelo de desenvolvimento de software centrado no usuário. Além disto, percebe-se também a necessidade da realização de outros trabalhos de modo a incluir novas características na solução proposta. A seguir, são apresentadas algumas recomendações de trabalhos futuros:

- ao observar a eficácia dos artefatos de IHC, emanada de análises comparativas dentre alguns artefatos de IHC e ES, descritas no Capítulo 5, novas adaptações no modelo proposto podem ser realizadas. Poderão ser investigadas formas de diminuir a quantidade de artefatos do universo de ES envolvidos em tais análises, à medida que o modelo for utilizado. Para isto, será necessária a elaboração de um novo experimento de validação capaz de verificar não apenas a completude destes artefatos de IHC, mas, sobretudo, a eficiência do modelo face ao novo arranjo de artefatos e a realidade do cenário onde será utilizado. Pretende-se, com isto, evoluir um pouco mais em termos de processo de desenvolvimento para que se possam identificar novas formas de se configurar os artefatos disponibilizados, até o momento.
- deverá ser investigado, ainda, o impacto da incorporação de novos ambientes computacionais ao modelo sugerido. Atualmente, há uma proposta interessante de trabalho em andamento [SCH 03] no tocante ao estudo e levantamento de um suporte ferramental ao método MCI, onde serão apontadas ferramentas acadêmicas ou comerciais que se adequem a tal método. Dessa forma, a possibilidade de inclusão de novas ferramentas ao modelo abrirá novos caminhos em direção a uma maior automatização, facilitação do uso do MCI e, conseqüentemente, do modelo sugerido.

- outro trabalho a ser realizado para melhoria do modelo proposto é a coleta e análise de dados de novas amostras em um outro cenário de validação. Faz-se necessário, sobretudo, validar a metodologia com um número mais significativo de usuários. No momento da conclusão deste trabalho, o modelo foi novamente (e amplamente) adotado pela disciplina Laboratório de Engenharia de Software. Assim, após o término desta, seria interessante abordar as equipes de alunos e seus respectivos clientes que participaram dos projetos de tal disciplina com o intuito de verificar se a aplicação do modelo foi (ou não) realizada com sucesso.
- é necessária a implantação do modelo proposto em um ambiente externo à academia - na indústria, de preferência. Uma vez que o experimento de validação definido no trabalho foi elaborado e implantado no ambiente acadêmico, deverá ser investigado ainda, em um outro momento, a eficácia e aceitação do modelo sugerido em um projeto de sistemas de software de maior porte.

6.4 Considerações finais

A integração harmoniosa entre algumas das melhores práticas da ES e IHC não alcançada pelos demais processos de desenvolvimento da ES, um endereçamento adequado às principais técnicas de usabilidade e a inclusão do usuário final como um parceiro ativo em todo o ciclo de concepção de um sistema são características que diferem o modelo proposto dos demais modelos de desenvolvimento de sistemas existentes.

Ao se propor um novo modelo para a concepção de aplicações computacionais, percebeu-se que é possível trazer para a comunidade de ES técnicas já consolidadas da comunidade de IHC, evidenciando a usabilidade nos ambientes de projeto, agregando valor às atividades do universo da ES e favorecendo a qualidade e aceitação dos produtos de software gerados. A estrutura do modelo proposto e suas partes atendem as características listadas, como foi discutido em função dos resultados apresentados no Capítulo 5. Acredita-se, portanto, que os objetivos propostos no trabalho tenham sido alcançados.

Referências Bibliográficas e Bibliografia

Referências Impressas

[AND 01] ANDERSON, J.; FLEEK, F.; GARRITY, K.; DRAKE, F. *Integrating Usability Techniques into Software Development*. IEEE Software, vol.18, no.1. (January/February 2001), pp. 46-53.

[ANT 01] ANTUNES, H.; SEFFAH, A.; DJOUAB, R. Comparing and Reconciling Usability-Centered and Use-Case Driven Requirements Engineering Processes. In *IEEE: 2nd Australian User Interfaces Conference (AUIC'01),* January 29 - February 01, 2001, Gold Coast, Queensland, Australia.

[BAC 94] BACH, J. *Enough about process: what we need are heroes.* IEEE Software, vol. 12, no. 2, (March 1994), pp. 96-98.

[BAR 02] BARBOSA, S. D. J.; de SOUZA, C. S.; SILVEIRA, M. S. Design de sistemas de ajuda on-line baseado em modelos. *IHC - V Simpósio sobre Fatores Humanos em Sistemas Computacionais*, Fortaleza, 2002.

[BEC 00] BECK, K. ***Extreme programming explained: Embrace Change***. Reading: Addison-Wesley, 2000.

[BOL 99] BOLLINGER, T.; NELSON, R.; SELF, K. M.; TURNBULL, S. J. *Open source methods: Peering through the clutter.* IEEE Software, vol.16, no.4, (July/August 1999), pp. 8-11.

[BOO 00] BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. ***UML - Guia do Usuário.*** Editora Campus, 2000.

[CON 99] CONSTANTINE, L. L; LOCKWOOD, L. A. D. ***Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design.*** Addison -Wesley, 1999.

[COR 03] CORDEIRO, P. B. *Projeto e Implementação do Módulo TAME da Ferramenta ITAOS para Análise e Modelagem da Tarefa.* Dissertação de Mestrado, Departamento de Sistemas e Computação, UFCG, Campina Grande, Paraíba, Fevereiro de 2003.

[COS 01] COSTABILE, M.F. ***Usability in the Software Life Cycle.*** Handbook of Software Engineering and Knowledge Engineering. ed. S.K. Chang World Scientific Publishing, Singapore, 2001. 179-192.

[DEM 95] DEMARCO, T. ***Mad about measurement in: Why does Software Cost so Much?*** New York: Dorset House, 1st. ed., 1995.

[FER 01a] FERRÉ, X. Incorporating Usability into an Object Oriented Development Process. *In INTERACT 2001, 8th IFIP TC. 13º Workshop on Human-Computer Interaction*, Waseda University Conference Centre, Shinjuku Tokyo, Japan, July 9-13, 2001.

[FER 01b] FERRÉ, X.; JURISTO, N.; WINDL, H.; CONSTANTINE, L. *Usability Basics for Software Developers.* IEEE Software, vol.18, no.1, (January/February 2001), pp. 22-29.

[FER 03] FERRÉ, X. Approaches to HCI Integration into Software Engineering Processes: How Much We Still Need to Do. *WIHC-ES 2003: Integrating Human-Computer Interaction and Software Engineering Models and Processes*, Rio de Janeiro - RJ, Brazil, August 2003.

[FUR 97] FURTADO, M. E. S. *Mise en oeuvre d'une méthode de conception d'interfaces adaptatives pour des systèmes de supervision à partir de spècifications conceptualles.* PhD thesis, Doctorat de productique et Informatique à l'Université Aix Marseille III, France, 1997.

[FUR 99] FURTADO, M. E. S., Integrando Fatores Humanos no Processo de Desenvolvimento de Interfaces Homem-Computador Adaptativas, *II Workshop sobre Fatores Humanos em Sistemas Computacionais*, Campinas, SP, Brasil, 1999.

[GAM 98] GAMBOA, F. R. *Spécification et Implémentation d'ALACIE: Atelier Logiciel d'Aide à la Conception d'Interfaces Ergonomiques.* Thèse de Doctorat, Paris XI, Octobre, 1998.

[GAM 00] GAMMA, E.; HELM, R.; JOHNSON, R. & VLISSIDES, J. ***Padrões de Projeto: Soluções Reutilizáveis de Software Orientado a Objetos.*** Ed. Bookman, Porto Alegre, RS, 2000.

[GAU 91] GAUSE, D. M.; WEINBERG, G. M. ***Explorando requerimentos de sistemas.*** São Paulo, Makron, McGraw-Hill.

[GLA 01] GLASS, R. *Agile versus traditional: Make love, not war!* Cutter IT Journal, Vol. 14, No. 12, December 2001, pages 12-18.

[GUE 01] GUERRERO, C. V. S.; LULA, B. Jr. Model-guided and task-based approach to UI design centered in a unified interaction and architectural model, *CADUI'2002 - 4th International Conference on Computer-Aided Design of User Interfaces*, p. 107 - 119, Valenciennes, FRANCE, May 2002.

[GUE 01a] GUERRERO, C. V. S.; LULA, B. Jr. *Um tutorial na Web: obtenção do modelo da interação a partir do modelo da tarefa com o auxílio de regras ergonômicas.* Relatório Técnico RT- DSC-002/2001 (55 páginas), UFPB/DSC/CCT Campina Grande, PB, Outubro de 2001.

[GUE 01b] GUERRERO, C. V. S.; LULA, B. Jr. MEDITE: uma abordagem baseada em modelos para a concepção de interfaces ergonômicas*, IX Encuentro*

Chileno de Computación 2001- Jornadas Chilenas de Computación 2001, Atas (CD-ROM) Seção: Interfaz humano-computador y computación gráfica / Trabalho nº 4 (10 páginas), Punta Arenas - CHILE, Novembro de 2001.

[GUE 02] GUERRERO, C. V. S. *MEDITE: Uma Metodologia Orientada a Modelos para a Concepção de Interfaces Ergonômicas.* Dissertação de Mestrado, Departamento de Sistemas e Computação, UFCG, Campina Grande, Paraíba, Fevereiro de 2002.

[HAA 92] HAAN G.; VAN der VEER, G. C; VAN VLIET, J.C. *Formal Modeling Techniques in Human-Computer Interaction.* Acta Psychologica, 78, nos. 1-3, 26-76, North-Holland, Amsterdam, 1992.

[HAA 00] HAAN, G. *ETAG - A Formal Model of Competence Knowledge for User Interface Design.* Ph.D. Thesis, Vrije Universiteit, Amsterdam, October, 2000.

[HAM 95] HAMMOUCHE, H. *De la modélisation des tâches utilisateurs au prototype de l'interface homme-machine.* Thèse de Docteur, Université Paris VI, Décembre, 1995.

[IEE 90] IEEE. *IEEE Std. 610.12-1990. IEEE Standard Glossary or Software Engineering Terminology*. IEEE, New York NY, 1990.

[IEE 97] IEEE. *IEEE Std. 1074-1997. IEEE Standard 1074 for Developing Software Life Cycle Processes*. IEEE, New York NY, 1998.

[ISO 95] ISO/IEC. *International Standard: Information Technology. Software Life Cycle Processes, ISO/IEC Standard 12207:1995*. ISO, Geneva, Switzerland, 1995.

[ISO 98] ISO. *Ergonomic Requirements for Office Work with Visual Display Terminals, ISO 9241-11*, ISO, Geneva, 1998.

[ISO 99] ISO. *International Standard: Human-Centred Design Processes for Interactive Systems, ISO Standard 13407: 1999*. ISO, Geneva, Switzerland, 1999.

[ISO 02] ISO/IEC. *International Standard: Information Technology. Software Life Cycle Processes. Amendment 1. ISO/IEC Standard 12207:1995/Amd.1:2002*. ISO, Geneva, Switzerland, 1995.

[JAC 94] JACOBSON, I. ***Object-Oriented Software Engineering: A Use Case Driven Approach.*** Addison-Wesley Object Technology Series, Reading, Massachusetts: Addison Wesley, 1994.

[JAC 99] JACOBSON, I.; BOOCH, G.; RUMBAUGH, J. ***The Unified Software Development Process.*** Addison Wesley, 1999.

[JOH 88] JOHNSON, P.; JOHNSON, H.; WADDINGTON R; SHOULS A. *Task-Related Knowledge Strutures: Analysis, Modelling and Aplication.* Queen Mary College, University of London 1988.

[KRU, 00] KRUCHTEN, Phillipe. ***The Rational Unified Process, An Introduction***. (The Addison-Wesley Object Technology Series), 2000.

[LAR 00] LARMAN, C. ***Utilizando UML e Padrões - uma introdução à análise e ao projeto orientados a objetos.*** Bookman, 2000.

[LEH 88] LEHMAN, M. M. Some Reservations on Software Process Programming. *In: Proceedings of the IEEE/ACM Software Process Workshop*, vol. 4, 1988. p. 111-112.

[LIN 00] LINDVALL, M.; RUS, I. *Process diversity in software development.* IEEE Software, vol. 17, no. 4, (July/August 2000), pp. 14-18.

[LUL 92] LULA, B. Jr. *Elaboration d'un Environnement de Génération Interactive d'Interfaces à Manipulation Directer pour le Language OBJLOG.* Thèse de Docteur, Universidade de Droit d'Economie et des Sciences d'Aix-Marseille III, Faculté des Sciences et Techniques de Saint – Jérôme, França, 1992.

[MAU 02] MAURER, F.; MARTEL, S. *Extreme Programming: Rapid Development for Web-Based Applications.* IEEE Internet Computing, vol. 6, no. 1, (January/February 2002), pp. 86-90.

[MED 95] MEDEIROS, H; ROUSSELOT F. Un Outil D'Aide à la Modélisation de Concepts Dynamiques: Le Système TAME; *Journées Acquisition - Validation - Apprentissage, JAVA'95*, 04/95, Grenoble, França, 1995.

[MED 03] MEDEIROS, F. P. A. de. *Projeto e Implementação do Módulo TAOS-GRAPH da Ferramenta ITAOS para Análise e Modelagem da Tarefa.* Dissertação de Mestrado, Departamento de Sistemas e Computação, UFCG, Campina Grande, Paraíba, Fevereiro de 2003.

[MED 00] MEDEIROS, H., KAFURE, I. M e LULA, B. Jr., TAOS: a Task-and Action Oriented Framework for User's Task Analysis in the Context of Human-Computer Interfaces, *In: Proceedings of SCCC 2000 – XX International Conference oh he Chilean Computer Science Society*, Santiago, Chile, November 2000.

[MOR 98] MORAES, M. A. *Informática, a angústia do usuário.* Revista Developers, n. 31, ano 3, p.66, Março, 1998.

[PRE 95] PRESSMAN, R. ***Engenharia de Software.*** Makron Books, 1995.

[RAC 98] RACCOON, L. B. S. Toward a tradition of software engineering. ACM SIGSOFT Software Engineering Notes, vol.23, no.3, (May 1998), pp.105-110.

[RAD+ 01] RADDING, A. *Simplicity, But With Control* - Extreme programming promises speed and flexibility for those who can stomach it. (Technology Information) – Information Week, April 2, 2001 p71.

[RAD 01] RADLE, K.; YOUNG, S. *Partnering Usability with Development: How Three Organizations Succeeded.* IEEE Software, vol.18, no.1, (January/February 2001), pp. 38-45.

[SCA 88] SCAPIN, D. L. *Vers des outils formels de description des taches orientes conception d'interfaces*, Rapports de Recherche, Unité de Recherche, INRIA, Rocquencourt, France, 1988.

[SCA 89] SCAPIN, D. L.; PIERRET-GOLKBREICH, C. *Towards a Method for Task Description: MAD*, Unité de Recherche, INRIA, Rocquencourt, France, 1989.

[SCH 03] SCHERER, D. *Proposta de Dissertação de Mestrado: Proposta de Suporte Computacional ao Método MCI.* Coordenação de Pós-graduação em Informática, COPIN, Departamento de Sistemas e Computação, UFCG, Campina Grande, Paraíba, Fevereiro de 2003.

[SCW 95] SCWABER, K. SCRUM Development Process. *In: Proceedings of OOPSLA*, 1995. Springer-Verlag.

[SHA 96] SHAW, M.; GARLAN, D. ***Software Architecture: Perspectives of an Emerging Discipline.*** Englewood Cliffs, NJ: Prentice Hall, 1996.

[SIL 99] SILBERSCHATZ, Abraham; KORTH, Henry F. e SUDARSHAN, S. ***Sistemas de Banco de Dados.*** São Paulo: Makron Books do Brasil Editora Ltda, 1999.

[SOU 99] SOUSA, M. R. F. *Avaliação Iterativa da Especificação de Interfaces com Ênfase na Navegação.* Tese de Doutorado, Coordenação de Pós-Graduação em Engenharia Elétrica - COPELE, UFPB, Campina Grande, Dezembro de 1999.

[STA 97] STAPLETON, J. ***DSDM - Dynamic Systems Development Method.*** Addison-Wesley, 1997.

[STÁ 01] STÁBILE, S. *Um Estudo sobre a Desconexão entre Usuários e Desenvolvedores de Sistemas de Informação e sua Influência na Obtenção de Informação pelo Decisor.* Dissertação de Mestrado, Escola de Engenharia de São Carlos da Universidade de São Paulo, EESC/USP, São Paulo-SP, Julho de 2001.

[TAU 88] TAUBER, M. J. *On Mental Models and the User Interface.* In: van der Veer, G. C.; Green, T. R. G.; Hoc, J. M.; Murray, D. M. *Working with Computers: theory versus outcome.* Academic Press, London. pp. 89-119, 1988.

[TAU 90] TAUBER, M. J. ETAG: Extended Task Action Grammar - a language for the description of the user's task language. In: Diaper, D.; Gilmore, D.; Cockton, G.; Shackle, B. In *Proceedings of Interact'90*, pp. 163 - 168. Elseviers, North-Holland, Amsterdam, 1990.

[TUR 02] TURNELL, M. F. Q. V. *Notas de Aula da disciplina Projeto de Interface Homem-Máquina.* Departamento de Engenharia Elétrica, UFCG, Campina Grande, Paraíba, 2002.

[VAS 03] VASCONCELOS, C. R.; GARCIA, F. P.; TURNELL, M. F. Q. V. Integrando Usabilidade e Engenharia de Software: um modelo para o desenvolvimento de sistemas centrado no usuário. *WIHC-ES 2003: Integrating Human-Computer Interaction and Software Engineering Models and Processes*, Rio de Janeiro - RJ, Brazil, August 2003.

[VAS 03a] VASCONCELOS, C. R.; GARCIA, F. P.; TURNELL, M. F. Q. V. Integrando Usabilidade e Engenharia de Software: um modelo para o desenvolvimento de sistemas centrado no usuário. *II SBQS – Simpósio Brasileiro de Qualidade de Software / II WTQS - Workshop de Teses em Qualidade de Software* – Fortaleza, Ceará, Brasil, Setembro 2003.

[VAS 03b] VASCONCELOS, C. R.; GARCIA, F. P.; TURNELL, M. F. Q. V. Integrando Usabilidade e Engenharia de Software: um modelo para o desenvolvimento de sistemas centrado no usuário. *II WDCopin – Workshop de Dissertações, Departamento de Sistemas e Computação*, UFCG, Campina Grande, Paraíba, Brasil, Agosto 2003.

[VAS 03c] VASCONCELOS, C. R.; GARCIA, F. P. *Relatório de Atividades Realizadas Durante o Estágio de Docência.* Departamento de Sistemas e Computação, UFCG, Campina Grande, Paraíba, Brasil, Junho de 2003.

[WAN 95] WANG, C. B. ***O novo papel do executivo de informática.*** São Paulo, Markon Books.

[WHI 94] WHITAKER, K. ***Gerenciando maníacos por software.*** Rio de Janeiro, Editora IBPI Press, 1994.

Referências Eletrônicas

[ANT 03] ANTHILL BUILD MANAGEMENT SERVER. Disponível por WWW em http://www.urbancode.com/projects/anthill/default.jsp (12 de novembro de 2003).

[ANT+ 03] ANT - The Apache Ant Project. Disponível por WWW em http://ant.apache.org/ (14 de novembro de 2003).

[BAC 03] BACON - Build Automation Control. Disponível por WWW em http://www.dsc.ufcg.edu.br/~vinicius/bacon/ (15 de dezembro de 2003).

[BLU 03] BLUEJ - The Interactive Java Environment. Disponível por WWW em http://www.bluej.org/ (13 de dezembro de 2003).

[CAC 03] CACTUS - The Jakarta Cactus. Disponível por WWW em http://jakarta.apache.org/cactus/ (12 de dezembro de 2003).

[COC 03a] COCKBURN, A. Crystal Methodologies. Disponível por WWW em http://crystalmethodologies.org/ (04 de abril de 2003).

[COC 03b] COCKBURN, A. Crystal Clear: A Human-Powered Methodology for Small Teams, Addison-Wesley Disponível por WWW em http://members.aol.com/humansandt/crystal/clear (5 de agosto de 2003).

[CRI 00] CRISPIN, L. The Need for Speed: Automated Acceptance Testing in an XP environment. At Quality Week Europe, November, 2000. Disponível

por WWW em: http://www.soft.com/QualWeek/QWE2K/Papers/4T.html (12 de abril de 2003).

[CVS 03] CVS - Concurrent Versions System. Disponível por WWW em http://www.cvshome.org/ (4 de dezembro de 2003).

[CVS+ 03] CVSGUI. Disponível por WWW em http://www.wincvs.org/shots.html (5 de dezembro de 2003).

[CYB 98] CYBIS, W. de A.; PIMENTA, M. S.; SILVEIRA, M. C.; GAMEZ, L. Uma Abordagem Ergonômica para o Desenvolvimento de Sistemas Interativos. Atas IHC98. Disponível por WWW em http://www.unicamp.br/~ihc99/Ihc99/AtasIHC99/AtasIHC98.html (25 de janeiro de 2003).

[DIA 02] DIAS, G. H. A. A. *Análise de Sistemas I, Capítulo 1: Introdução à Engenharia de Software.* Disponível por WWW em http://www.di.ubi.pt/~ddg/asi/chapter1.pdf (30 de outubro de 2002).

[DSD 03] DSDM TOUR. The Essential Introduction to DSDM. Disponível por WWW em http://www.dsdm.org/ (05 de março de 2003).

[EAS 03] Easyprocess. Disponível por WWW em http://www.dsc.ufcg.edu.br/~pet/easyprocess/index.htm. (20 de maio de de 2003).

[ECL 03] ECLIPSE.ORG. Disponível por WWW em http://www.eclipse.org/ (18 de novembro de 2003).

[EUT 03] EUTERPE, A Task Analysis Tool. Disponível por WWW em http://www.cs.vu.nl/~martijn/euterpe.html (14 de agosto de 2003).

[FDD 03] FEATURE DRIVEN DEVELOPMENT. The Portal for All Things FDD. Disponível por WWW em http://www.featuredrivendevelopment.com/ (5 de agosto de 2003).

[FOW 03] FOWLER, M. *The New Methodology*. Disponível por WWW em http://www.martinfowler.com/articles/newMethodology.html (12 de Abril de 2003).

[FOW+ 03] FOWLER, M.; FOEMMEL, M. *Continuous Integration*. Disponível por WWW em: http://www.martinfowler.com/articles/continuousIntegration.html (25 de agosto de 2003)

[GAR 02] GARTNER GROUP. *Gartner Says No New Major IT Innovation Before 2005.* Disponível por WWW em http://banners.noticiasdot.com/termometro/boletines/docs/ti/gartner/2002/gartner_Before_2005.pdf (02 de setembro de 2003).

[GUM 03] GUMP. Disponível por WWW em http://jakarta.apache.org/gump/ (15 de dezembro de 2003).

[JEF 02] JEFFRIES, R. *XProgramming.com*: an Extreme Programming Resource. Disponível por WWW em http://www.xprogramming.com/. (20 de dezembro de 2002).

[JAV 03] JAVA TECHNOLOGY. Disponível por WWW em http://www.java.sun.com/ (18 de novembro de 2003).

[JOH 01] JOHNSON, J.; BOUCHER, K. D.; CONNORS, K.; ROBINSON, J. *Collaborating on Project Success*. Software Magazine - February/March 2001. Disponível por WWW em http://www.softwaremag.com/archive/2001feb/CollaborativeMgt.html (15 de outubro de 2003).

[JUN 03] JUNIT - Testing Resources For Extreme Programming. Disponível por WWW em http://www.junit.org/index.htm (15 de novembro de 2003).

[MAN 03] MANIFESTO FOR AGILE SOFTWARE DEVELOPMENT. Disponível por WWW em http://www.agilemanifesto.org/ (8 de novembro de 2003).

[MED 98] MEDEIROS, A. *Ferramentas de concepção de interfaces homem-máquina.* CTAIREVISTA. Volume I, Número 2, Dezembro de 1998. Disponível por WWW em http://www.ctai.rct-sc.br/revista/artigo3.html (24 de outubro de 2002).

[OSH 03] OSHIRO, A. K.; NOVELLI, A. D. P.; CASELI, H. M.; LUCENA, P. de. *Extreme Programming: um novo modelo para o desenvolvimento de software*. Disponível por WWW em http://talkagentfw.sourceforge.net/percival/artigos/ArtigoXP.pdf (12 de setembro de 2003).

[PET 03] Programa Especial de Treinamento. Disponível por WWW em http://www.dsc.ufcg.edu.br/~pet/ (15 de maio de 2003)

[POS 03] POSEIDON FOR UML COMMUNITY EDITION. Disponível por WWW em http://www.gentleware.com/products/download.php4 (15 de novembro de 2003).

[RAT 03] RATIONAL ROSE DEVELOPER. Disponível por WWW em http://www-306.ibm.com/software/awdtools/developer/rose/features/ (18 de outubro de 2003).

[SAU 03] SAUVÉ, J. *Métodos Avançados de Programação.* Disponível por WWW em http://www.dsc.ufpb.br/~jacques/cursos/map/html/arqu/resumo.html (3 de setembro de 2003).

[SWD 03] SOFTWARE DEVELOPMENT. The Lifecycle Starts Here. Disponível por WWW em http://www.sdmagazine.com/ (22 de setembro de 2003).

[SEF 02] SEFFAH, A. Human-Centered Software Engineering: Designing for and with Humans. *In Canadian Undergraduate Software Engineering Conference, March 7-9, 2002, Montreal, Canadá*. Disponível por WWW em http://www.cusec.ca/archives/cusec2002/keynoteSeffah.ppt (13 de dezembro de 2002).

[SGI 98] THE STANDISH GROUP INTERNATIONAL (1998): CHAOS - A Recipe for Success, Disponível por WWW em

http://standishgroup.com/sample_research/PDFpages/chaos1998.pdf (24 de março de 2003).

[SUT 01] SUTHERLAND, J. *SCRUM Software Development Process*. Disponível por WWW em http://jeffsutherland.com/scrum/index.html. (04 de abril de 2003).

[XPL 03] XPLANNER HOME. Disponível por WWW em http://www.xplanner.org/ (15 de dezembro de 2003).

[WEL 02] WELLS, D. *Extreme Programming: a gentle introduction.* Disponível por WWW em http://www.extremeprogramming.org (20 de dezembro de 2002).

Anexo A

O Documento de Visão

Com o objetivo de facilitar a compreensão do artefato Documento de Visão, será mostrado, no corrente anexo, um breve exemplo para que o leitor possa observar como este artefato deve ser apresentado e feito. Conforme já foi dito Capítulo 4, este artefato é uma recomendação do processo YP [EAS 03] e foi trazido para o XPU.

A.1 Exemplo de um documento de visão

[EAS 03] recomenda que, em sua estrutura, um documento de visão seja composto de: (1) uma descrição geral do projeto em uma linguagem natural, (2) uma lista contendo os requisitos funcionais do sistema, (3) uma outra lista contendo os requisitos não funcionais e (4) um perfil inicial do usuário.

Veja, abaixo, um exemplo [EAS 03] de um documento de visão para um projeto fictício:

1) **Descrição do projeto:**

"O Sr. Joaquim queria aumentar as vendas da mercearia e deixar os fregueses contentes. Como o pessoal gostava de comprar fiado, encomendou umas cadernetas, carimbando nas capas a sigla: "CRM - Caderneta de Registro Mensal". Era nelas que passou a controlar as contas dos fregueses. Logo a CRM ficou popular no bairro. "Anota aí na CRM dois quilos de tomate para a patroa", pedia a Josefina. "Minha mãe mandou perguntar quanto vai pagar de CRM este mês", dizia a Silvinha. Mas a caderneta não servia apenas para cobrar os fregueses. Era a sua bola de cristal. Nela o português enxergava muito mais que o total que iria receber no final do mês. Ele

identificava ciclos de comportamento do freguês, suas preferências, a associação dos produtos adquiridos e muitas outras coisas. A freguesa comprava sempre tomate? Dá-lhe campanha promovendo o macarrão e o queijo ralado. A última compra foi há muito tempo? Joaquim ligava avisando que a laranja estava em promoção. E o freguês ia buscar somente porque o Joaquim havia ligado. Até o Pepe, do açougue ao lado, criou sua própria caderneta CRM para acompanhar as preferências da freguesia. Logo Joaquim e Pepe traçavam informações de suas CRMs, para ganho mútuo. E o Manoel da padaria acabou aderindo ao sistema, assim como o João do boteco. Cada um passou a ser um agente de uma pequena rede de troca de informações. O Joaquim vendeu carvão e sal grosso para o Dr. Januário? O Pepe era logo avisado e ia preparando a carne que o doutor gostava. O Manoel aumentava a receita do pão e o João do boteco colocava mais cerveja para gelar. Cada comerciante sabia prever a próxima compra, para fazer a próxima oferta e exceder a expectativa dos fregueses. Todos prosperavam e os fregueses estavam contentes".

2) **Requisitos funcionais (deste projeto fictício)**:

- Cadastro de Clientes - O dono do estabelecimento cadastra os clientes.
- Registrar Vendas - O dono do estabelecimento cadastra as vendas efetuadas, associando ao cliente que efetuou a compra.
- Consultar Promoções - O dono do estabelecimento entra com as informações do cliente para verificar em quais as promoções que o cliente está inserido.
- Definir Promoção - O dono do estabelecimento escolhe uma categoria de clientes (Ouro, Prata, Bronze e Lata) e define as regras para promoções dentro de cada categoria.
- Cadastro de Produtos - O dono do estabelecimento cadastra os produtos do estabelecimento.
- Cadastro de Funcionários - O dono do estabelecimento entra com as informações dos funcionários do estabelecimento.
- Cadastro de Usuários do Sistema - O dono do estabelecimento cadastra *login* e senha para cada usuário do sistema.
- Criar Eventos - O dono do estabelecimento entra com as informações de um evento que estará associado a um cliente.

- Cadastro de Categorias - O dono do estabelecimento cadastra informações a respeito das categorias de clientes que ele deseja criar para o seu estabelecimento.
- Consultar Eventos - O dono do estabelecimento escolhe um tipo de Eventos e o sistema emite uma listagem dos eventos.
- Emitir Relatórios - O dono do estabelecimento escolhe um dos relatórios e o sistema gera o relatório.
- Consultar Financeiro - O dono do estabelecimento escolhe uma categoria de clientes (Ouro, Prata, Bronze e Lata) e define as regras para promoções dentro de cada categoria.

3) **Requisitos não funcionais**

- Interface – Gráfica tipo *desktop* de um sistema. O usuário poderá utilizar o mouse e o teclado interagir com a interface do produto. A interface deverá ser de fácil utilização, com o objetivo de facilitar a vida do dono do estabelecimento.
- Volume de Utilização - Sem restrição;
- Segurança - Haverá necessidade de *login* por parte dos donos do estabelecimento para utilizar o sistema;
- Qualidade / Robustez - A integridade dos dados é garantida pelo Sistema Gerenciador de Banco de Dados (SGBD);

4) **Perfil do usuário**

Proprietário de qualquer tipo de estabelecimento comercial, que queira manter um histórico das compras dos clientes. Funcionários dos proprietários devem ser capazes de interagir com a aplicação, dessa forma, o sistema deve ter uma interface para um público que não é usuário experiente em computação.

Anexo B

Os Objetivos de Usabilidade

Este anexo apresenta de forma breve como levantar a lista contendo os objetivos de usabilidade indicados em XPU. Primeiramente o leitor será motivado para a importância de se levantar os objetivos de usabilidade junto a um produto de software. Em seguida, serão citados alguns exemplos de objetivos de usabilidade bem a disposição destes em uma tabela.

B.1 Definição

Segundo Turnell [TUR 02], este artefato envolve o estabelecimento de um conjunto de objetivos, na forma de requisitos de desempenho, mensuráveis, que devem ser atingidos pelo produto. Estes objetivos devem ser colocados em uma escala de prioridades.

B.2 Por quê?

- Os objetivos de usabilidade apóiam as decisões de projeto e facilitam uma posterior avaliação dos atributos do produto.
- Os objetivos de usabilidade devem se fundamentar no conhecimento do Perfil do usuário e, sobretudo, na inspeção de produtos similares. Isso porque o levantamento de problemas em outros produtos com propósitos similares auxilia a proposição de um conjunto de objetivos que visam superar estes problemas encontrados.

B.3 Exemplos de objetivos de usabilidade

- reduzir a taxa de erros,
- reduzir o tempo de realização da tarefa,
- facilitar a memorização,
- aumentar a satisfação do usuário,
- clareza na estrutura,
- rapidez na interação,
- atratividade,
- conteúdo adequado,
- facilitar o aprendizado,
- identidade visual,
- atualização das informações.

B.4 Exemplo do artefato (em um projeto fictício)

Objetivos de Usabilidade	Mensuração de atributos de Usabilidade
Reduzir o tempo de realização da tarefa	Tempo de execução das tarefas: Desempenho inicial versus desempenho após uso prolongado
Facilitar o aprendizado do sistema	Quão rápido e fácil o usuário aprende a usar o sistema; Uso de recursos avançados.
Elevar o tempo de retenção de informações	Por quanto tempo o aprendizado é mantido
Aumentar a satisfação subjetiva o usuário	Primeira impressão e opinião após uso prolongado
Reduzir taxa de erros	Número de tarefas concluídas sem falhas
Facilitar a memorização	

Tabela 17: Tabela de Objetivos de Usabilidade.

Anexo C

Levantamento do Perfil do Usuário

O corrente anexo ilustra de forma resumida como efetuar o levantamento do perfil do usuário em um projeto de software. Inicialmente, serão discutidos alguns conceitos e definições importantes acerca desta atividade. Em seguida, serão apresentadas algumas características que são, normalmente, consideradas neste levantamento, um exemplo desta atividade para um sistema fictício e, por fim, será mostrado o questionário sugerido por XPU para tal levantamento.

C.1 Conceitos

De acordo com Turnell [TUR 02], levantar o perfil do usuário consiste em:

- representar o comportamento de um usuário individual ou de um grupo de usuários;
- definir o perfil dos usuários potenciais em termos das tarefas a serem realizadas; seu potencial, limitações, preferências, interesses; aptidões e os conhecimentos necessários para realizá-las.

O perfil do usuário pode ser levantado a partir da análise realizada pela equipe de desenvolvimento do produto com base nos seguintes aspectos:

- observação dos usuários;
- entrevistas com usuários;
- preenchimento de questionários pelos usuários.

No caso de XPU, optamos pelo preenchimento de um simples questionário, conforme modelo será apresentado mais adiante.

C.2 Características consideradas no documento de perfil do usuário

- idade;
- sexo;
- grau de instrução;
- motivações;
- objetivos;
- personalidade;
- aptidões;
- Freqüência de utilização: Ocasional, Regular;
- Experiência com o sistema: treinamento, tentativa e erro, etc;
- Habilidades motoras (Velocidade, precisão);
- Habilidades de percepção (visual, auditiva, táctil,etc.).

É possível ainda combinar as características acima com o conhecimento sintático e semântico do usuário necessário à interação, independentemente das características de uma plataforma específica. As definições destes tipos de conhecimentos podem ser encontradas, abaixo:

- Conhecimento sintático – variado, dependente de dispositivo, adquirido por memorização, facilmente esquecido.
- Conhecimento semântico (tarefa e ações) – estruturado, independente de dispositivo, adquirido por aprendizado, e estável na memória. (o usuário tem conhecimento semântico da tarefa, mas pode não ter o conhecimento sintático do comando/dispositivo específico).

C.3 Exemplo de um Levantamento de Perfil de Usuário

A seguir, no Quadro 2 abaixo, será apresentado um exemplo de um levantamento de perfil de usuário para um sistema de consulta de livros de uma biblioteca.

CONHECIMENTO SEMÂNTICO:	**Nível do conhecimento**
Função: **Bibliotecário**	Elevado
Tarefa realizada:. cadastramento de DVD	Médio
Método de realização: utilizando código de barras	Baixo
Ferramentas utilizadas: Leitora de código de barras	Baixo
Computadores	Médio
CONHECIMENTO SINTÁTICO:	**Nível do conhecimento**
teclado	Médio
dispositivos especiais de interação (caneta ótica)	Baixo
terminologia específica	Alto
ESTILO COGNITIVO:	
Aprendizado	por tentativa e erro
Capacidade de solucionar problemas	iniciativa própria
Retenção do aprendizado	facilidade
Nível de curiosidade	baixo
Nível de persistência	elevado
Nível de motivação	elevado
Nível de inovação	baixo
HABILIDADES NECESSÁRIAS PARA EXECUÇÃO DA TAREFA:	
Velocidade de digitação	Média
Precisão no uso de dispositivos especiais	Elevada
Níveis de percepção visual	médio

Quadro 2: Exemplo de uma análise do perfil de usuário [TUR 02].

C.4 Questionário para levantamento do perfil do usuário

Para que se colete dados a respeito do usuário do sistema, XPU considera a aplicação de um breve questionário[14], conforme mostrado abaixo:

Formulário para o levantamento do Perfil do Usuário
© 2001, Laboratório de Interfaces Homem-Máquina, UFCG

Características do usuário, escolhidas pelo projetista, de acordo com a relevância, para o projeto. Levantamento baseado em:

Fatos ☐ Opinião do usuário ☐ Dados medidos ou observados ☐

[14] Este questionário é aplicado pelo Grupo de Interfaces Homem-Máquina da Universidade Federal de Campina Grande – UFCG.

Parte I – Características Gerais

Faixa etária:
Sexo:
Habilidades necessárias para executar a tarefa:
    Níveis de percepção (ex. acuidade visual):
    habilidades motoras (ex. velocidade, precisão):
Grau de instrução (ex. técnico, superior):
Função desempenhada na Organização (ex. gerência, atendimento ao público):
Tarefas realizadas na Função (ex. cadastramento, supervisão):
Freqüência de execução das Tarefas na função (ex. diária, semanal, mensal):
Objetivos (o que pretende com o sistema):
Motivações (por que usaria o sistema):
Preferências (ex. uso do teclado, preenchimento do formulário):

Parte II - CONHECIMENTO CONCEITUAL necessário à execução das tarefas:

Conhecimento Semântico	Nível de experiência
Função	
Método	
Tarefa	
Computadores	
Ferramentas utilizadas na execução das tarefas (ou similares):	

Conhecimento Sintático	Nível de experiência
Uso de teclado e mouse	
Uso de dispositivos especiais de interação	
Uso de terminologia específica	

Parte III - ESTILO COGNITIVO:

Aprendizado (ex. treinamento):
Capacidade de solucionar problemas (sozinho, com ajuda):
Capacidade de reter o aprendizado(alta, média, baixa):

Personalidade:
Nível de curiosidade (baixo, médio, elevado)
Nível de persistência (baixo, médio, elevado)
Nível de inovação (baixo, médio, elevado)
Inovador ( ) Conservador ( )
Impulsivo ( ) Reflexivo ( )

Anexo D

Análise e Modelagem da Tarefa

O corrente anexo apresenta conceitos fundamentais para a realização da modelagem da tarefa do usuário – atividade sugerida pelo Método para Concepção de Interfaces (MCI) e XPU. Inicialmente, serão apresentados alguns conceitos fundamentais relacionados à análise e modelagem da tarefa. Em seguida, será exposto um exemplo de um artefato. Por fim, é indicado um suporte ferramental para automatizar a geração do modelo.

D.1 Definição

De acordo com Turnell [TUR 02], modelar a tarefa do usuário consiste em realizar um estudo detalhado desta com o propósito de determinar sua natureza, propósito, as partes que a compõem (subtarefas ou ações básicas) e a ordem na qual estas partes devem ser executadas.

Ao analisar a tarefa o projetista deve ser capaz de levantar dados sobre os seguintes aspectos:

- seqüências necessárias e porque são necessárias;
- papel do usuário no processo;
- freqüência de execução da tarefa;
- grau de dificuldade para realizá-la (como é percebido pelo usuário);
- criticalidade na execução da tarefa (gravidade da conseqüência dos erros);

- alocação de tarefas e funções - quais aspectos podem ser automatizados com o objetivo de aumentar a produtividade, a eficiência e a qualidade do sistema.

Ações de alto nível são decompostas em ações intermediárias as quais são refinadas em ações atômicas que o usuário executa com um único comando. Vale ressaltar que a descrição detalhada da tarefa, denominada Modelo da Tarefa, apóia as fases de:

- especificação do sistema, uma vez que muitos dos requisitos funcionais do sistema aparecem no modelo;
- projeto da interface e
- elaboração de material de treinamento.

D.2 Resultado da Análise da Tarefa

Turnell [TUR 02] afirma que o resultado da análise da tarefa do usuário consiste na descrição detalhada e **hierarquizada** do trabalho, com um diagnóstico das situações problemáticas, graus de dificuldades e as soluções possíveis, e/ou recomendações ergonômicas para a concepção da interface do futuro sistema.

D.3 Exemplo de um artefato

A decomposição de uma tarefa é hierárquica. Inicia com a tarefa objetivo (nível mais alto, ou seja, a raiz) até as tarefas elementares (nível mais baixo, ou seja, as folhas). As tarefas elementares são tarefas descritas por ações elementares.

Com o objetivo de facilitar a compreensão do artefato Modelo da Tarefa, será mostrado um breve exemplo para que o leitor possa observar como este artefato deve ser apresentado e feito. O modelo adiante foi criado para um *site* de divulgação do Laboratório de Interface Homem-Máquina da Universidade Federal de Campina Grande.

Pode-se observar abaixo, na Figura 8, que apenas a sub-árvore *Conhecer o Grupo de IHM* foi trabalhada. É possível notar também que as tarefas e subtarefas foram numeradas para uma melhor compreensão. Vale lembrar ainda que, quando o usuário **tiver de escolher** uma dentre um conjunto de atividades, ou ainda **tiver de percorrer uma de seqüência** de atividades, os operadores OR e SEQ devem ser adicionados. Na Figura 8, apenas o operador OR foi necessário.

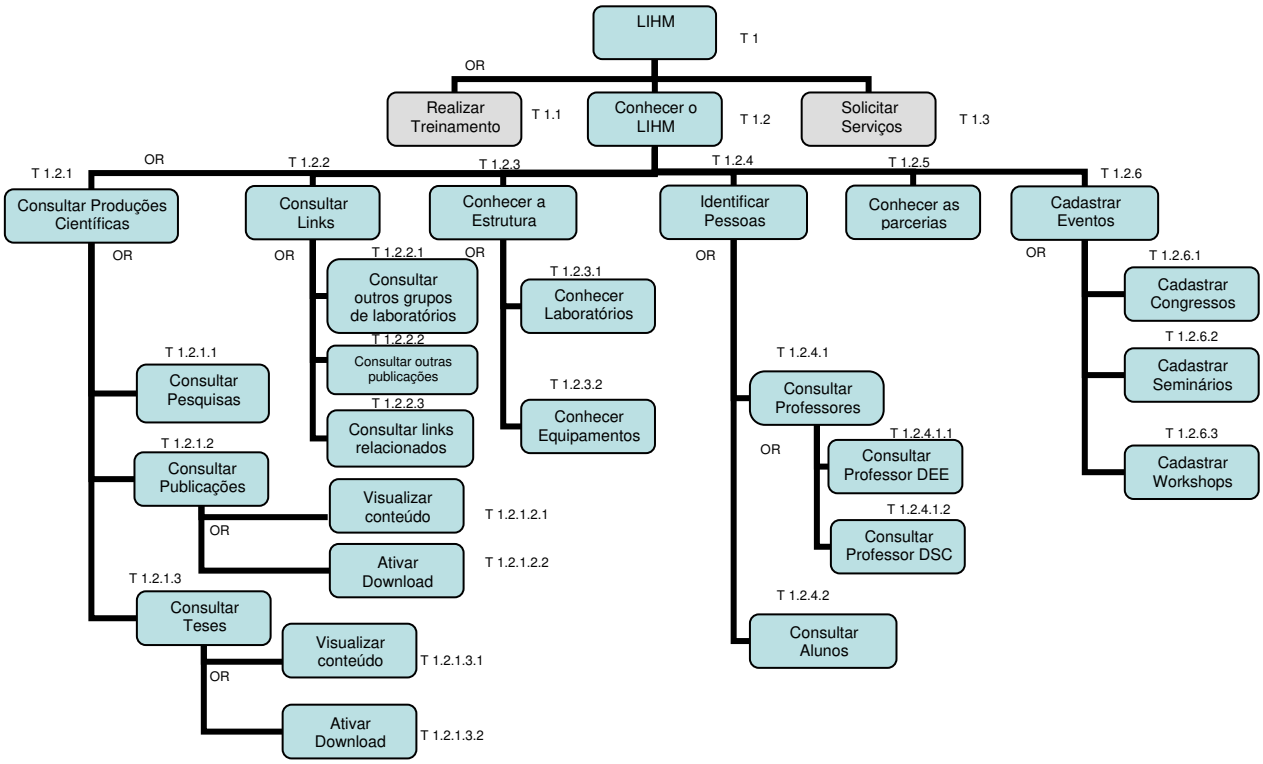

Figura 8: Exemplo do artefato Modelo da Tarefa.

Neste exemplo e, conforme citado anteriormente, ações de alto nível puderam ser decompostas em ações intermediárias as quais foram refinadas até o nível de ações elementares as quais o usuário executa com um único comando. Note que as ações-folhas podem ser mapeadas diretamente para um procedimento ou função nas linguagens de programação convencionais da Engenharia de Software. Por exemplo: a tarefa do usuário *Ativar Download* pode ser mapeada para a operação *ativarDownload();*

D.4 Ferramenta

Para a construção do modelo da tarefa, XPU sugere o uso da ferramenta Euterpe [EUT 03]. Trata-se de uma ferramenta para análise e descrição de tarefas do usuário de código livre, fácil aprendizado e que pode ser baixada sem ônus algum para a equipe de desenvolvimento.

Anexo E

A Modelagem da Interação

O corrente anexo apresenta de forma sucinta como realizar a modelagem da interação do usuário com o sistema sugerida na fase de planejamento da iteração de XPU. A modelagem da interação proposta em XPU é composta de duas fases: inicialmente, será mostrada, na primeira seção, como levantar os objetos e ações da tarefa do usuário a partir do modelo da tarefa. Em seguida, será mostrado como associar cada um desses objetos e ações com objetos e ações do domínio da interface com o usuário. Exemplos de artefatos serão mostrados para facilitar a compreensão desta atividade.

E.1 Levantamento dos objetos e das ações

Primeiramente, segundo Turnell [TUR 02], uma lista contendo todos os objetos e ações atômicos representados no Modelo da Tarefa deve ser preparada. Estes objetos e ações serão, em seguida, associados a metáforas propostas para a interação do usuário com o sistema.

A partir do modelo da tarefa, deve-se construir uma tabela na qual são listados os objetos e ações envolvidos nas tarefas do usuário. Nesta tabela, são listadas as ações elementares, os objetos sobre as quais são realizadas e a respectiva identificação oriunda do modelo da tarefa. O leitor pode notar que, conhecendo-se a tarefa, pode-se retirar as ações e os objetos relacionados. Tarefas: Ações (objetos). Ex.: *Identificar (aluno); Cadastrar (cliente); Selecionar (serviço).*

Com o objetivo de facilitar a compreensão desta atividade, observe, na Tabela 18 abaixo, um exemplo do artefato para a sub-árvore *Consultar Produções Científicas* presente no modelo da tarefa, apresentado no Anexo D:

Identificação da Tarefa do Usuário	Modelo da Tarefa	
	Ações	**Objetos**
T1.2.1.2.2	Ativar Download	Publicações
T1.2.1.3.2	Ativar Download	Teses
T1.2.1.1	Consultar	Pesquisas
T1.2.1.2.1	Visualizar Conteúdo	Publicações
T1.2.1.3.1	Visualizar Conteúdo	Teses

Tabela 18: Exemplo de uma Tabela de Ações e Objetos do Modelo da Tarefa.

E.2 Concepção das metáforas dos manipuladores para execução das ações

Após a análise da tarefa e, conhecendo-se os objetos e ações, a equipe pode então escolher dentre os estilos de interação[15]. De acordo com Turnell [TUR 02], esta atividade consiste na seleção de metáforas para representar os objetos de informação levantados e as estratégias para realização das ações representadas no modelo da tarefa.

De forma não tão abrangente, a criação de metáforas e manipuladores associados aos objetos e ações do modelo da tarefa levantados na seção anterior consiste em associar objetos e ações do modelo da tarefa a objetos e ações no domínio da interface com o usuário.

Com o objetivo de facilitar a compreensão desta atividade, observe, na Tabela 19 abaixo, um exemplo deste segundo artefato para a sub-árvore *Consultar Produções Científicas* presente no modelo da tarefa, apresentado no Anexo D:

[15] Seleção de um menu, Manipulação direta (ativação de um botão), Acionar um comando, Arrastar um item na tela, etc.

Id. Tarefa	Modelo da Tarefa		Modelo da Interação		
	Ação	Objeto	Ação	Objeto	Visualização
T1.2.1.2.2	Ativar Download	Publicações	Selecionar	Menu Principal > Opção Ativar Download	JanelaPrincipal
T1.2.1.3.2	Ativar Download	Teses	Acionar	Teclas CRTL + D	JanelaPrincipal
T1.2.1.1	Consultar	Pesquisas	Ativar	Botão Consultar	JanelaConsulta
T1.2.1.2.1	Visualizar Conteúdo	Publicações	Selecionar	Link Visualizar	JanelaResultado
T1.2.1.3.1	Visualizar Conteúdo	Teses	Selecionar	Ícone Visualizar	JanelaResultado

Tabela 19: Associação dos Objetos e Ações do Modelo da Tarefa a Objetos e Ações da Interface.

É possível observar, na tabela acima, que ações e estilos de interação (menus, teclas, botões, etc.) são definidos para auxiliar a execução das ações e especificar o modo como o usuário vai interagir com o sistema.

Anexo F

Questionários de Avaliação

O corrente anexo apresenta os dois modelos de questionários utilizados na abordagem de coleta de informações de alunos e clientes durante a validação da metodologia XPU. Inicialmente, será mostrado o modelo de questionário aplicado com a equipe de desenvolvimento. Em seguida, o questionário destinado aos clientes será exposto.

F.1 O modelo de questionário aplicado na equipe de desenvolvimento

UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE CIÊNCIAS E TECNOLOGIA
DEPARTAMENTO DE SISTEMAS E COMPUTAÇÃO
CURSO DE MESTRADO EM INFORMÁTICA

Aos Membros das Equipes de Desenvolvimento,

Solicitamos sua colaboração no sentido de responder ao presente questionário, acerca do processo adotado ao longo do seu projeto.

Salientamos que a precisão de suas respostas será fundamental na composição do modelo que estamos propondo. Asseguramos ainda, que as informações colhidas serão estritamente sigilosas, a fim de resguardar a privacidade daqueles que estão colaborando, garantido-lhes a liberdade necessária para responder as questões.

Certos de sua colaboração, agradecemos antecipadamente.

César Rocha Vasconcelos
Mestrando em Informática

1 - Assiduidade nas reuniões

1.1 - Como a equipe avalia sua assiduidade nas reuniões combinadas com o cliente?

Em _____ % das reuniões, a equipe esteve presente.

1.2 - E a assiduidade do cliente?

Em _____ % das reuniões, o cliente esteve presente.

2 - Eficiência do modelo sugerido

De um modo geral, como a equipe avalia a eficácia do modelo de desenvolvimento quanto aos resultados do ponto de vista da iteração do usuário com o sistema que foi produzido?

[ ] Ótima [ ] Boa [ ] Regular [ ] Ruim

3 - Esforço na geração dos artefatos de usabilidade

Atribuindo números de 1 a 6, classifique as atividades listadas abaixo, quanto a sua duração no processo; Atribua 1 a atividade de menor duração.

[ ] Levantamento das Necessidades do Cliente
[ ] Definição dos Objetivos de Usabilidade
[ ] Análise do Perfil do Usuário
[ ] Modelagem da Tarefa
[ ] Modelagem da Iteração
[ ] Testes de Usabilidade

4 - Complexidade na geração dos artefatos de usabilidade

Atribuindo números de 1 a 6, classifique as atividades listadas abaixo, quanto a sua dificuldade no processo. Atribua 1 a atividade de menor complexidade

[ ] Levantamento das Necessidades do Cliente
[ ] Definição dos Objetivos de Usabilidade
[ ] Análise do Perfil do Usuário
[ ] Modelagem da Tarefa
[ ] Modelagem da Iteração
[ ] Testes de Usabilidade

5 - Perfil do Usuário e Necessidades do Cliente

Na opinião da equipe, as atividades de Levantamento do Perfil do Usuário e das Necessidades do Cliente, ambos artefatos de Usabilidade, podem complementar o Documento de Visão?

[ ] Sim [ ] Não. Caso negativo, explique por quê ?_____

6 - Modificações no protótipo inicial da interface com o usuário

Como a equipe avalia o impacto das atividades de usabilidade no protótipo inicial da interface com o usuário, gerado antes do conhecimento do modelo?

[ ] Muitas modificações foram feitas no protótipo inicial
[ ] Modificações foram feitas no protótipo inicial
[ ] Poucas modificações foram feitas no protótipo inicial
[ ] Nenhuma modificação foi feita no protótipo inicial

7 - Uso da ferramenta EUTERPE

Como a equipe avalia a introdução da ferramenta Euterpe para a realização da Modelagem da Tarefa? A ferramenta foi útil?

[ ] Sim [ ] Não. Caso negativo, por quê?_____

8 - Modelagem da Tarefa versus Requisitos Funcionais

Observando o modelo da tarefa que foi construído, a equipe avalia que as tarefas levantadas podem ser confrontadas com os requisitos funcionais (tarefas de programação que foram "quebradas" a partir das user stories) do sistema?

[ ] Sim [ ] Não. Caso negativo, por quê?_____

9 - Modelagem da Tarefa versus Contrato das Classes

A equipe acredita que as tarefas-folhas encontradas na modelagem da tarefa poderiam ser mapeadas para métodos presentes nos contratos das classes envolvidas no sistema?

[ ] Sim [ ] Não. Caso negativo, por quê?_____

O espaço a seguir é reservado para seus comentários e sugestões sobre o modelo:

_____
_____
_____
_____
_____

Muito obrigado!!!

F.2 O modelo de questionário aplicado no cliente do projeto

UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE CIÊNCIAS E TECNOLOGIA
DEPARTAMENTO DE SISTEMAS E COMPUTAÇÃO
CURSO DE MESTRADO EM INFORMÁTICA

Aos Clientes,

Visando aumentar a eficácia do método de desenvolvimento de sistemas, adotado neste projeto, pedimos sua colaboração no sentido de responder algumas questões envolvendo as etapas de desenvolvimento de seu projeto. Salientamos que suas respostas são fundamentais para o sucesso de nosso trabalho e asseguramos que as informações fornecidas têm um caráter estritamente sigiloso.

Certos de sua colaboração, agradecemos antecipadamente.

César Rocha Vasconcelos
Mestrando em Informática

1 - Assiduidade nas reuniões

1.1 – Qual o seu grau de assiduidade nas reuniões combinadas com o grupo de alunos?

Eu estive presente.em _____ % das reuniões,

1.2 – E a assiduidade do grupo de alunos?

O grupo inteiro esteve presente em _____ % das reuniões,.

2 - Prazo de entrega

A entrega dos *releases* foi feita dentro dos prazos ?

[ ] Sim [ ] Não

3 - Relevância / Aceitação

Houve conformidade do projeto apresentado pelo grupo de alunos com o que foi acordado inicialmente?

[ ] Sim [ ] Não

4 - Objetivos de Usabilidade

Houve correspondência entre os objetivos de usabilidade propostos para o projeto e a interface do sistema produzida pelo grupo de alunos?

[ ] Sim [ ] Não

5 - Interface com o sistema

5.1 – Você está satisfeito com a interface do usuário com o sistema?

[ ] Sim [ ] Não

5.2 - Como você avalia a facilidade de uso do produto através da interface projetada?

[ ] Fácil utilização
[ ] Difícil utilização com a necessidade de suporte

5.3 – De um modo geral, como você avalia a entrega da interface pelo grupo de alunos:

[ ] Entregue e funcionando adequadamente
[ ] Entregue, mas apresenta problemas

5.4 – De um modo geral, a interface do sistema apresentou melhorias ao longo do andamento das iterações no projeto?

[ ] Muitas melhorias foram apresentadas.
[ ] Melhorias foram apresentadas.
[ ] Poucas melhorias foram apresentadas.
[ ] Nenhuma melhoria foi apresentada.

Este espaço é reservado para suas observações e comentários sobre o projeto e seus resultados:

_____
_____
_____
_____
_____

Muito obrigado!!!