MARIA ROSILENE FERREIRA LOPEZ

Estudo da Prototipação na Engenharia de Requisitos para Desenvolvimento de Softwares Interativos em Ciclo de Vida Acelerado

Trabalho final apresentado ao Instituto de Pesquisas Tecnológicas do Estado de São Paulo – IPT, para obtenção do título de Mestre em Engenharia da Computação
Área de concentração: Engenharia de Software

São Paulo

2003

MARIA ROSILENE FERREIRA LOPEZ

Estudo da Prototipação na Engenharia de Requisitos para Desenvolvimento de Softwares Interativos em Ciclo de Vida Acelerado

Trabalho final apresentado ao Instituto de Pesquisas Tecnológicas do Estado de São Paulo – IPT, para obtenção do título de Mestre em Engenharia da Computação
Área de concentração: Engenharia de Software

Orientadora: Prof[a]. Dra. Lúcia Vilela Leite Filgueiras

São Paulo

2003

Ficha Catalográfica

Lopez, Maria Rosilene Ferreira
Estudo da prototipação na engenharia de requisitos para desenvolvimento de softwares interativos em ciclo de vida acelerado/ Maria Rosilene Ferreira Lopez. São Paulo, 2003.
113 p.

Trabalho Final (Mestrado Profissional em Engenharia de Computação) – Instituto de Pesquisas Tecnológicas do Estado de São Paulo. Área de concentração: Engenharia de Software

Orientador: Prof. Dra. Lúcia Vilela Leite Filgueiras

1. Engenharia de software 2. Requisitos do usuário 3. Análise de requisitos 4. Prototipação 5. Ciclo de vida 6. Tese I. Instituto de Pesquisas Tecnológicas do Estado de São Paulo. Centro de Aperfeiçoamento Tecnológico II.Título

004.414.3/.32(043)
F383e

Dedicatória

A meu filho e meus pais, que me apoiaram com amor e carinho em todos os momentos deste trabalho.

Agradecimentos

Gostaria de expressar meus agradecimentos:

A professora Lúcia Vilela Leite Filgueiras pela orientação, paciência, dedicação e incentivo no desenvolvimento desta Dissertação.

Ao IPT, pela oportunidade concedida para a realização do Curso de Mestrado em Engenharia de Computação e pela possibilidade de utilizar os conhecimentos adquiridos ao longo destes 25 anos de trabalho nesta empresa, na elaboração desta dissertação.

Aos nossos clientes e em especial às empresas Alcatel Telecomunicações e Itautec–Philco e, ainda, à Secretaria da Fazenda do Estado de São Paulo e à Secretaria de Ciência, Tecnologia, Desenvolvimento Econômico e Turismo do Estado de São Paulo por terem permitido o uso de seus nomes e da experiência adquirida nos nossos trabalhos conjuntos.

Aos colegas da DITEL que de várias formas contribuíram, com seus conhecimentos, experiências e muito incentivo, para a realização desta pesquisa.

Ao Professor Mário Myiake que sempre me cobrou este trabalho, levando-me a sua realização.

Em especial aos meus pais, Jovelino e Ignêz, por estarem sempre ao meu lado incentivando minha formação.

A minha irmã, Sueli, sobrinhos e cunhado, pelo apoio na conclusão deste trabalho.

Ao meu filho, Rodrigo, pela compreensão e paciência, pelos finais de semana sem passeios, pelas férias sem viagem, sem os quais não chegaria até aqui.

A Deus, por tudo.

Sumário

Lista de figuras

Lista de abreviaturas

Sigla	Descrição
AST	Agrupamento de Software e Telecomunicações da DITEL
ATM	*Automated Teller Machines*, ou seja, Máquinas de Caixa Automatizadas
CAPS	Computer Aided Prototyping Systems
DFD	Diagrama de Fluxo de Dados
DITEL	Divisão de Informática e Telecomunicações do IPT
ERS	Especificação dos Requisitos de Software
ES	Engenharia de Software
GUI	"*Grafical User Interface*", ou seja, Interface gráfica do usuário
IHC	Interface Humano-Computador
IPT	Instituto de Pesquisas Tecnológicas do Estado de São Paulo
UCD	"*User Centered Design*", ou seja, "*Design* centrado no usuário"
URA	Unidade de Resposta Audível

RESUMO

Ferreira Lopez, Maria Rosilene. **Estudo da Prototipação na Engenharia de Requisitos para Desenvolvimento de Softwares Interativos em Ciclo Acelerado.** Trabalho final apresentado ao Instituto de Pesquisas Tecnológicas do estado de São Paulo – IPT, para a obtenção do título de Mestre Profissional em Engenharia da Computação, São Paulo, 2003.

Este trabalho estuda o uso da prototipação dentro do processo de desenvolvimento de software para projetos com ciclo de vida acelerado. A partir da análise da importância da prototipação na Engenharia de Requisitos, na Engenharia de Usabilidade e no processo adotado pela equipe do AST-DITEL-IPT, propõe-se um roteiro de recomendações que levam em conta a experiência prática, aliada aos conceitos levantados na revisão bibliográfica. Estas recomendações orientam equipes de desenvolvimento na utilização dos protótipos como ferramenta de levantamento, análise, apresentação, e validação dos requisitos.

Palavras-chave: Prototipação; ciclo de vida acelerado; processo de desenvolvimento de software; usabilidade; engenharia de requisitos.

ABSTRACT

Ferreira Lopez, Maria Rosilene. **Study of the Prototyping in the Requirements Engineering for Development of Interatives Softwares in accelerated life cycle**. Master´s Thesis in Computer Engineering, Instituto de Pesquisas Tecnológicas do Estado de São Paulo - IPT. São Paulo, 2003

This work studies the role of prototyping in accelerated software development projects. The study departs from the analysis of prototyping in Requirements and Usability Engineering and from a critical view over AST-DITEL-IPT interactive software development process and produces a set of prototyping guidelines based both on professional experience and on literature review. These guidelines should help development teams in using prototypes as tools for requirements elicitation, analysis, presentation and validation.

Key-words: Prototyping; accelerated lifecycle; software development processes; usability; requirements engineering.

Capítulo 1. Introdução

O processo de desenvolvimento de software envolve diversas etapas, independentemente do ciclo de vida adotado. No entanto, qualquer que seja o processo escolhido, o levantamento, validação e gerência de requisitos são fundamentais para o sucesso da empreitada. Existem diversas técnicas para se trabalhar com este assunto. Há evidências de que a prototipação, considerada como uma das técnicas para levantamento e validação de requisitos, possa também ser usada com papel estendido nos casos de desenvolvimento em ciclo acelerado.

O objetivo deste trabalho é estudar o papel da prototipação no desenvolvimento de software e discutir a ampliação de seu uso natural como instrumento de levantamento de requisitos para uma forma de registro e apresentação dos requisitos dos sistemas de software, especialmente nos projetos com ciclo de vida acelerado, onde o tempo de desenvolvimento é fator fundamental.

Além disso, a partir deste estudo e da vivência profissional, deseja-se sugerir um roteiro para o desenvolvimento de software utilizando protótipos, que leve em conta conceitos da gerência de requisitos e da engenharia de usabilidade, e que possa ser aplicado no desenvolvimento de software de sistemas interativos em ciclo acelerado.

1.1 Motivação

No desenvolvimento de um novo software, a questão do levantamento de requisitos é de fundamental importância. Por melhor que seja a equipe de desenvolvimento, se os requisitos levantados não forem completos, forem ambíguos ou não houver um entendimento correto sobre as necessidades dos usuários, certamente o projeto terá dificuldades, podendo mesmo ser abandonado.

Em muitos sistemas só se percebem os erros de especificação quando o sistema entra em produção. A falta de uma funcionalidade que o usuário considere importante ou sua implementação de forma errada pode causar o desencanto dos usuários que passam a considerar que o produto não atende às suas necessidades e, a partir daí, podem começar a criticar e “jogar contra” o mesmo.

Erros ocorridos em outras fases do projeto, principalmente na fase de programação, normalmente são muito mais fáceis de serem corrigidos do que erros de especificação, que podem afetar inclusive toda a modelagem realizada.

Outra grande dificuldade encontrada é a validação, junto aos usuários, do resultado da análise feita sobre os requisitos levantados, ou seja, o que o sistema a ser desenvolvido fará e como. A apresentação de um documento com a definição de todos os requisitos do sistema, normalmente é de leitura difícil para os usuários (Yourdon,

1990). Embora eles possam validar os requisitos, ainda assim, não há garantias de que a especificação está completa, sem erros de entendimento e que não é ambígua.

Outros documentos oriundos da análise, tais como o DFD (Diagrama de Fluxo de Dados) ou os documentos da análise orientada a objetos, são ilegíveis para a grande maioria dos usuários, principalmente quando o sistema será usado por usuários que não possuem tradição na utilização e desenvolvimento de novos sistemas (YOURDON, 1990). Os documentos e diagramas apresentados são muito técnicos para a grande maioria deles. Nem sempre o cliente tem um corpo técnico capacitado para ler os documentos e em última instância é o usuário final e não o técnico que detém os requisitos. Por exemplo, quando apresentamos um dicionário de dados para usuários novatos dificilmente eles lerão e entenderão este documento, principalmente se for usado todo o formalismo necessário.

Tem-se ainda o problema dos “usuários.programadores”. Com a grande demanda reprimida de softwares, muitos usuários resolveram desenvolver suas pequenas aplicações. Como eles não possuem a formação técnica de modelagem e não conhecem engenharia de requisitos, fica impossível exigir-se que eles providenciem a documentação adequada.

Na literatura clássica de Engenharia de Software, representada principalmente pelos trabalhos de Pressman (2002) e Yourdon (1990), a prototipação é citada como uma abordagem eficiente para o levantamento, apresentação e validação dos requisitos em várias situações, tais como: quando há dificuldades na definição dos requisitos ou para sistemas com muitas interfaces e poucos algoritmos.

Tradicionalmente a prototipação serve somente para o levantamento e a validação dos requisitos. Com esta técnica é possível validar, junto aos usuários, as funcionalidades que o sistema deverá implementar, bem como as características das informações tratadas pelo sistema. Além disso, pode-se avaliar, logo nas primeiras etapas do projeto, a interface humano-computador que será implementada. Certamente, correções nesta etapa do desenvolvimento causarão muito menos impacto do que se descobertas depois do sistema pronto.

Um outro ponto importante a ser levado em conta são os fatores humanos envolvidos. No ciclo clássico de desenvolvimento, o usuário vê o software apenas depois de pronto. Se o software não corresponder às suas expectativas isto causará um desencanto que, muitas vezes é irreversível, embora o analista faça as mudanças desejadas. Quando se usa a técnica de prototipação o usuário vê o software logo no início do projeto, verifica as mudanças que deseja, valida as interfaces e requisitos, criando com isso uma cumplicidade com a equipe de desenvolvimento pois ele sente que faz parte da equipe, que é ouvido e que é capaz de influenciar nos resultados. Isto tudo faz com que ele assuma o projeto ajudando para que seja levado a bom termo.

No entanto, neste trabalho, deseja-se discutir o uso da prototipação de forma mais abrangente, como forma de documentação de software. Para sistemas on-line de pequeno e médio porte, o protótipo se apresenta como alternativa eficiente para a substituição de alguns artefatos do ciclo de vida, principalmente nos casos de desenvolvimento em ciclo acelerado. Embora a prototipação possa eventualmente ser considerada um pouco mais demorada e cara do que apenas a validação do documento de requisitos, no processo de desenvolvimento como um todo, ela acaba trazendo um ganho de tempo e de recursos.

Deve-se considerar ainda que com o advento da micro-informática, os clientes passaram a exigir interfaces amigáveis e intuitivas. Hoje não basta que o sistema faça o que foi especificado. Ele tem que ser fácil de ser usado, ter telas agradáveis e seguir padrões comuns de uso. Então, mesmo que os requisitos tenham sido levantados com apuro e estejam totalmente corretos, resta ainda saber se a interface com o usuário estará de acordo com os princípios da Engenharia de Usabilidade. Ainda lembrando a importância das interfaces no processo de desenvolvimento de software, Nielsen (1993) relata que as interfaces representam 48% do código do software e 1/3 do esforço de revisão.

Nos últimos anos também ocorreu uma mudança importante nas exigências dos clientes. Os prazos admissíveis para que se obtenha um produto de software estão cada vez mais reduzidos. Nos anos 70 e 80 era natural que se levasse 1 ou 2 anos no desenvolvimento. Hoje não se admite mais do que seis meses para a obtenção de uma versão do produto. Este fator leva ao desenvolvimento em ciclo acelerado.

Correia (2003, p.1), define ciclo acelerado da seguinte forma:

> "Verifica-se que o prazo de conclusão de um projeto tem duas naturezas diferentes:
>
> a. **tempo de desenvolvimento (TD)**, que é o tempo estimado para o desenvolvimento e entrega do software, normalmente estimado por quem efetuará o desenvolvimento, a partir da complexidade do problema e da capacidade de realização da empresa;
>
> b. **O tempo de mercado (TM),** ou **janela de validade**, que é o tempo em que o software deve ser disponibilizado para operação, permitindo seu uso dentro de um prazo que atenderá às exigências do cliente ou do mercado.
>
> Considera-se desenvolvimento em ciclo acelerado a estratégia de organização da empresa e de seu processo de software de forma a permitir que o desenvolvimento seja executado em TM, sem perda das características de qualidade, reduzindo o ciclo de vida do software.
>
> Tem-se **ciclo acelerado** sempre que **TM** < **TD**."

> “Nota-se que a TI (Tecnologia da Informação) está apoiando os negócios de muitas organizações e que os projetos modernos devem ser implementados rapidamente, de modo que atendam à velocidade de mudança dos produtos e negócios.”

Como se pode ver, o desenvolvimento em ciclo acelerado não deve implicar na perda de qualidade mas sim numa racionalização das atividades de forma a cumprir a janela de validade do projeto. Para se alcançar estes objetivos o trabalho de Correia (2003, p. 31) propõe ações para o desenvolvimento de projetos em ciclo acelerado. Segundo o autor, para haver ciclo acelerado,

> “... ações e atividades devem ser executadas antes do início do projeto, durante o seu desenvolvimento, pelo gerente do projeto e pela instituição, e ainda, depois que o projeto é finalizado.”

Em suma, projetos de ciclo acelerado necessitam ter uma infra-estrutura institucional preparada para a sua realização, com atividades realizadas antes do início do desenvolvimento e atividades que serão executadas apenas após a obtenção do produto, de forma a acelerar o processo, gerando o produto dentro de sua janela de validade.

Considerando-se todas estas questões, vê-se que um grande desafio para as empresas de desenvolvimento de software, nos dias atuais, é desenvolver os seus produtos com qualidade e dentro de sua janela de validade.

Na empresa em que a autora trabalha, a maioria dos projetos tem sido desenvolvidos em ciclo acelerado devido a pressões de mercado. A prototipação, no seu sentido mais amplo, tem sido largamente usada e tem se mostrado eficiente, gerando muito pouco descontentamento, por parte do cliente, ao final do desenvolvimento. No entanto, este uso tem sido feito de forma empírica.

Este trabalho é um esforço para simplificar o processo de desenvolvimento de software tanto em aplicações de ciclo acelerado, como para software “descartável” - entendendo-se aqui “descartável” como o software que é desenvolvido especificamente para ser usado num determinado momento e realizar uma certa tarefa sendo abandonado depois, e nunca sujeito a manutenção.

Neste trabalho, será discutido o uso dos diversos tipos de protótipo no processo de desenvolvimento. Serão identificadas situações de uso ampliado (como documentação dos requisitos de software), que cuidados deve-se tomar e que documentação adicional deve ser gerada.

Como resultado deste trabalho apresenta-se um roteiro com recomendações para o desenvolvimento de software, baseado em protótipos e ressaltando-se sua utilização e ampliação de uso. Este roteiro serve de orientação para empresas e organizações de software e os chamados “usuários-programadores”, na elaboração de seus produtos.

1.2 Perspectivas de contribuição

Neste trabalho deseja-se resgatar o papel da prototipação no processo de desenvolvimento de software e criar um guia prático que contribua para a melhoria de qualidade no processo de desenvolvimento de software, especificamente para os sistemas de informação fortemente interativos desenvolvidos com ciclo acelerado, pois pretende-se que ele seja simples, de fácil aplicação e baseado tanto na teoria como na experiência, mostrando que a teoria ratifica a prática e consolida o processo.

1.3 Metodologia

A metodologia empregada no desenvolvimento deste trabalho, para atingir os objetivos propostos, compreende as seguintes etapas:

Etapa 1 – Revisão bibliográfica

Em primeiro lugar, foi realizada uma pesquisa bibliográfica onde foram obtidos subsídios para o entendimento de como a prototipação é abordada na literatura. As atividades desta etapa foram as seguintes:

- Revisão e síntese dos principais conceitos da engenharia de software sobre as atividades referentes à prototipação;
- Revisão dos conceitos de usabilidade e principais conceitos e técnicas para criação de interfaces gráficas, no padrão GUI[1];
- Revisão da área de Engenharia de Requisitos com relação ao levantamento, validação, documentação e gerência de requisitos.

Etapa 2 – Levantamento do processo atual e da importância da prototipação

Nesta etapa procurou-se documentar, analisar e revisar as atividades do processo empregado na DITEL (Divisão de Informática e Telecomunicações do IPT), executando-se as seguintes atividades:

[1] GUI – *Graphical User Interface*, ou Interface gráfica do usuário

- Documentação das atividades práticas atualmente empregadas no desenvolvimento de software na empresa da autora;
- Detalhamento da importância da prototipação no processo de desenvolvimento de software atualmente empregado;
- Revisão destas atividades levando-se em conta a revisão bibliográfica efetuada;
- Proposta de ampliação do uso dos protótipos como forma de documentação dos requisitos de software.

Etapa 3 – Criação de roteiro para prototipação

Nesta etapa, que é a principal contribuição deste estudo, desenvolveu-se o roteiro de recomendações e discutiu-se o uso ampliado dos protótipos. Para tal, foram executadas as seguintes atividades:

- Identificação das boas práticas, na literatura;
- Identificação das boas práticas, na prática;
- Consolidação e criação de roteiro de sugestões de boas práticas para a elaboração de protótipos.

1.4 Estrutura do trabalho

Este trabalho possui a seguinte estrutura:

- Capítulo 2 – Prototipação – conceitos e evolução - Apresenta a prototipação na visão clássica de Engenharia de Software, experiências, sua evolução no decorrer do tempo, tipos e classificação dos protótipos;
- Capítulo 3 – Engenharia de Usabilidade - Apresenta os conceitos relativos à usabilidade e ao ciclo de vida da Engenharia de Usabilidade enfocando a importância da prototipação;
- Capítulo 4 - A prototipação na Engenharia de Requisitos - Apresenta a área de Engenharia de Requisitos e como a prototipação se insere dentro desta área;
- Capítulo 5 - A prototipação nos projetos da DITEL - Apresenta o processo de desenvolvimento de software da DITEL, como a prototipação se insere dentro deste processo e uma discussão sobre a ampliação do uso dos protótipos como forma de especificação dos requisitos;

- Capítulo 6 - Roteiro para o desenvolvimento de software em ciclo acelerado utilizando protótipos - Apresenta um roteiro de boas práticas com recomendações relativas ao processo de desenvolvimento de software tendo a prototipação como ferramenta essencial e,

- Capítulo 7 – Conclusões - Apresenta as conclusões finais do trabalho e sugestões para novos trabalhos.

Capítulo 2. Prototipação – conceitos e evolução

A prototipação tem sido usada como forma de validação dos requisitos há bastante tempo. Com as mudanças ocorridas no ambiente de desenvolvimento de software ao longo do tempo, foi aumentando sua importância como ferramenta de levantamento, apresentação e validação dos requisitos.

Este capítulo apresenta algumas definições de prototipação, sua evolução e utilização ao longo do tempo, caracteriza os tipos de protótipo e algumas formas de classificação.

2.1 Definição de Prototipação

Buscando-se na literatura, algumas definições sobre prototipação são encontradas. Apresenta-se algumas delas a seguir.

Na literatura clássica de análise e projeto de sistemas, Yourdon (1990, p. 120) apresenta a seguinte descrição para a prototipação, devida a Boar (1984):

> "Uma abordagem alternativa para a definição de requisitos é obter um conjunto inicial de necessidades e implementá-la rapidamente com a intenção declarada de expandi-las e refiná-las iterativamente à proporção do aumento do conhecimento mútuo do sistema por parte do usuário e do desenvolvedor. A definição do sistema ocorre através da descoberta gradual e evolutiva em oposição à previsão onisciente... Este tipo de abordagem é chamada de prototipação. Ela também é conhecida como modelagem de sistemas ou desenvolvimento heurístico. Ela oferece uma alternativa atraente e funcional para métodos de pré-especificação para que se possa lidar melhor com a incerteza, a ambigüidade e a inconstância dos projetos do mundo real."

Pressman (1995, p.35), em seu trabalho sobre Engenharia de Software, apresentava a seguinte definição para a prototipação:

> "A prototipação é um processo que capacita o desenvolvedor a criar um modelo do software que será implementado. Este modelo pode assumir uma das seguintes formas: (1) um protótipo em papel ou modelo baseado em PC que retrata a interação homem–máquina de uma forma que capacita o usuário a entender quanta interação ocorrerá; (2) um protótipo de trabalho que implementa algum subconjunto da função exigida do software desejado; ou (3) um programa existente que executa parte ou toda a função desejada,

> mas que tem outras características que serão melhoradas num novo esforço de desenvolvimento.”

Uma outra definição para protótipo é dada por Bahn e Nauman (1997, p. 240):

> “Protótipo é o processo de construção e avaliação de modelos de trabalho de um sistema a fim de se aprender sobre certos aspectos dos requisitos do sistema e/ou suas soluções potenciais”.

Lichter *et al*. (1993, p. 222), apresentam seu entendimento sobre o termo prototipação e os princípios que estão por trás dele:

- “Prototipação é um enfoque baseado numa visão evolucionária de desenvolvimento de software, afetando o processo de desenvolvimento como um todo.”

- “A prototipação envolve a produção o mais cedo possível de versões de trabalho (‘protótipos’) do futuro sistema aplicativo e a experimentação com ele”.

> “A prototipação providencia uma base de comunicação para discussões entre todos os grupos envolvidos no processo de desenvolvimento, especialmente entre usuários e desenvolvedores. Além disso, a prototipação permite-nos adotar um enfoque para a construção de software baseado em experimento e experiência”.

Como pode ser visto, as definições acima se complementam imprimindo um sentido e uma importância maior ao uso dos protótipos dentro do processo. Neste trabalho, o termo protótipo será usado na acepção mais ampla fornecida por Lichter *et al*. (1993).

2.2 A prototipação na visão clássica de Engenharia de Software

Consolidando-se os trabalhos clássicos de Pressman (2002) e Yourdon (1990) tem-se que a prototipação é citada como uma abordagem eficiente para o levantamento, apresentação e validação dos requisitos em situações em que:

- o cliente não é capaz de identificar os requisitos de entrada e saída, mas somente os objetivos gerais do software;
- o usuário não é capaz, ou não deseja definir todos os requisitos do sistemas. Só é possível obtê-los através de tentativa e erro;
- o desenvolvedor não tem certeza da eficiência de um algoritmo, ou da forma que a interação homem–computador deve assumir.

- o usuário não é capaz de examinar os modelos em papel;
- o sistema será on-line, tendo muita interação homem-computador;
- o sistema não exige grande detalhamento algorítmico.

Pressman (2002, p. 28) propõe o seguinte paradigma para a prototipação cuja seqüência de eventos é a ilustrada na Figura 1, e pode ser descrita como:

> "O paradigma da prototipagem começa com a definição dos requisitos. O desenvolvedor e o cliente encontram-se e definem os objetivos gerais do software, identificam as necessidades conhecidas e delineiam áreas que necessitam de mais definições. Um 'projeto rápido' é então realizado. Esse projeto concentra-se na representação daqueles aspectos do software que vão ficar visíveis ao cliente/usuário (p. ex., abordagens de entrada e formatos de saída). O projeto rápido parte de um protótipo. O protótipo é avaliado pelo cliente/usuário e usado para refinar os requisitos do software que será desenvolvido. Interações ocorrem à medida que o protótipo é ajustado para satisfazer às necessidades do cliente, enquanto, ao mesmo tempo, permitem ao desenvolvedor entender melhor o que precisa ser feito."

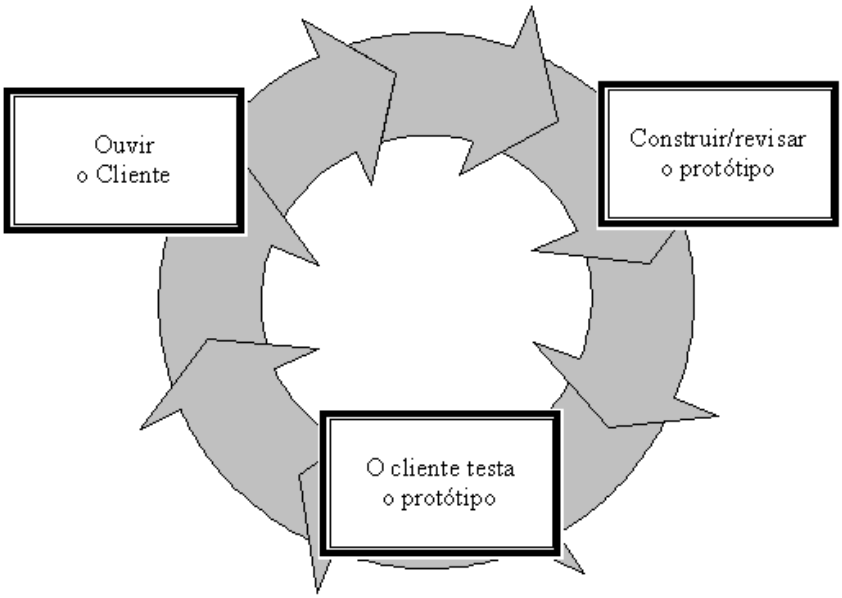

Figura 1. Paradigma da prototipagem, adaptada de Pressman (2002, p. 29)

Na década de 80, como as linguagens utilizadas levavam a uma programação demorada e difícil, sugeria-se que o desenvolvedor poderia utilizar, para facilitar e agilizar o desenvolvimento do protótipo:

- Pedaços de programas ou programas já existentes; e
- Ferramentas, como geradores de relatórios e gerenciadores de tela.

O protótipo era considerado um modelo que funcionava, ou seja, uma coleção de programas que simulavam algumas ou todas as funcionalidades do software que se desejava construir. Como os programas eram apenas um modelo, supunha-se também que, quando a modelagem terminasse, os programas seriam desprezados e substituídos por programas reais. O fato do protótipo ser descartado no final do levantamento dos requisitos levava a alguns problemas:

- O cliente podia não perceber que o protótipo não era o software final. Quando era informado de que o desenvolvimento iria iniciar, não aceitava a situação;
- No desenvolvimento do protótipo podia ser necessário utilizar ferramentas ou algoritmos que não eram os mais adequados, simplesmente para agilizar a criação do protótipo. Corria-se o risco do desenvolvedor esquecer os motivos que o fizeram adotar estas soluções para o protótipo e mantê-las no software final, prejudicando a qualidade do produto final;
- Se, por outro lado, o protótipo fosse convertido no sistema em produção corria-se o risco do software não conseguir tratar grandes volumes de informação, nem detalhes operativos como recuperação de erros, back-up etc,;
- Se o protótipo realmente fosse descartado, corria-se o risco de que o projeto acabasse sem um registro dos requisitos do sistema, dificultando, assim, a manutenção ao longo do tempo; portanto, sugeria-se que se trabalhasse conjuntamente com outros modelos.

Nesta mesma época, alguns estudos foram feitos para analisar a importância da prototipação dentro do processo de desenvolvimento de software. Um exemplo é o trabalho de Gomaa e Scott (1981), onde relatam uma experiência de uso de prototipação como ferramenta para a apresentação e validação dos requisitos do sistema. Neste caso foi usada uma linguagem específica para a prototipação (APL)[2] e foram criadas todas as interações com o usuário.

[2] A APL é uma linguagem de programação conhecida por seu uso de símbolos de não-ASCII, incluindo algumas letras gregas. É uma linguagem de alto nível criada por Ken Iverson e colegas na IBM nos 1960s. Ela foi projetada como uma notação executável poderosa, inspirada na matemática. Suas funções primitivas, que se aplicam a matrizes multidimensionais, têm os laços construídos nelas, desobrigando o programador de escrever muitos dos laços que são requeridos em outras linguagens. As operações tais como a classificação, pesquisa e a seleção são construídas na linguagem como primitivas. Os programas APL são muito mais curtos, mais fáceis de escrever, eliminar erros, e modificar. O ambiente interativo do APL incentiva a experimentação e facilita a prototipação e modificação rápidas das aplicações.

Os principais resultados obtidos nesta experiência foram:

- O protótipo revelou erros de entendimento entre os usuários e os desenvolvedores. Esses erros ocorriam com muita freqüência devido à diferença de conhecimento entre eles;
- Ambigüidades e inconsistências foram detectadas enquanto o protótipo estava sendo desenvolvido e puderam ser corrigidas;
- Omissões foram descobertas quando os usuários perguntaram por funções que eles consideravam essenciais, mas que não estavam disponíveis;
- Algumas funções não forneciam as informações que o usuário desejava, ou na forma que desejava;
- Terminologia confusa ou errada foi detectada;
- Os usuários foram capazes de dar um retorno valioso sobre as funcionalidades do sistema e se elas se apresentavam difíceis ou confusas para serem usadas;
- Em alguns casos o usuário não sabia como desejava a implementação de certas funções. O desenvolvimento do protótipo ajudou-os a tomar a decisão;
- O desenvolvimento do protótipo deu aos desenvolvedores uma visão valiosa de como o sistema deveria ser projetado, como os arquivos e dados deveriam ser estruturados e que algoritmos deveriam ser usados.

Como conclusão, os autores consideraram que o uso de um protótipo foi um método excelente para a avaliação da especificação dos requisitos. Além disso, o custo do desenvolvimento do protótipo foi menor do que 10% do custo total do projeto, o que foi considerado totalmente viável. Consideraram ainda que, com o uso do protótipo, vários erros foram encontrados e as mudanças foram feitas rapidamente. Se estas alterações tivessem que ser feitas numa etapa posterior do processo, causariam um gasto muito maior de tempo e dinheiro.

Em 1988 percebe-se a expansão do uso da prototipação, como no trabalho de Ribeiro e Buncker (1988). Era proposto, na época, que fossem usadas novas técnicas e ferramentas para o desenvolvimento de protótipos de software para auxiliar na análise estruturada de sistemas. Os autores propunham que os seguintes passos deviam ser realizados:

- Seguir as etapas da análise de sistemas estruturada até que o DFD do nível 1 e um dicionário de dados fosse gerados;
- Usar a informação do DFD e do dicionário de dados para direcionar o desenvolvimento de um protótipo do sistema. Na análise estruturada tradicional, os diagramas de nível inferior seriam feitos nesse ponto;

- Executar o protótipo na presença dos usuários do sistema e anotar onde o protótipo não correspondeu a suas expectativas; modificar o protótipo até que estivesse funcionando corretamente. O protótipo homologado podia também ser usado nas etapas da análise estruturada.

A principal vantagem deste método, segundo os autores, foi de que seria mais fácil verificar a exatidão do protótipo do que verificar a exatidão dos DFDs de um nível mais baixo, gerados durante as primeiras etapas da análise estruturada. A desvantagem desse método era que o protótipo tinha que ser programado em alguma linguagem de computador. Se esta tarefa fosse cansativa e consumisse muito tempo, isso anularia qualquer vantagem conquistada pelo uso do protótipo.

Para superar a desvantagem, foi usado o Prolog[3] que, segundo os autores, facilitou a tarefa de desenvolvimento de protótipo, diminuindo assim a complexidade e o tempo gasto no desenvolvimento do mesmo.

O objetivo do trabalho foi mostrar como o uso de protótipos podia, com sucesso, ser aplicado fornecendo um modelo que funcionava, em vez de um conjunto de gráficos. As entradas usadas para construir este protótipo foram os DFD´s (diagrama de contexto e DFD nível 1), e o primeiro esboço do dicionário de dados, o qual descrevia as especificações iniciais do sistema. Segundo os autores, as vantagens da prototipação incluíram:

- a habilidade para tentar novas idéias e ajudar nas decisões dos usuários;
- a rápida entrega para o usuário de um modelo do sistema funcionando, o qual mostrou o impacto do sistema proposto no ambiente da organização;
- uma oportunidade para que os usuários validassem os requisitos na etapa inicial, compreendendo suas necessidades de informação e como o sistema podia ou não ajudar a resolver tais necessidades;
- diminuição de riscos do projeto porque foi possível estabelecer como o sistema iria trabalhar antes de iniciar o projeto propriamente dito;
- redução de custos e tempo.

Os autores chegaram à conclusão de que os protótipos podiam ser vistos como ferramentas de melhoria de desempenho e que ajudavam no desenvolvimento de sistemas de informação. Eles forneciam uma boa base para validação e verificação dos diferentes passos para a construção de sistemas e também uma previsão do que se podia esperar do sistema pedido. Além disso, avaliaram que a prototipação era, na época, uma

[3] Prolog – esta linguagem foi concebida por Colmerauer e Roussel em 1972, sendo sua primeira versão feita em Fortran por Battani e Meloni em 1973. Desde então tem sido utilizada para aplicações de computação simbólica, como banco de dados relacionais, compreensão de linguagem natural, automação de projetos, análise de estruturas bioquímicas e sistemas especialistas.

técnica emergente que permitiria um rápido desenvolvimento do sistema completo, sendo uma técnica experimental excelente que poderia ser aplicada com sucesso na análise estruturada. Os autores advogavam que um protótipo poderia ser usado como uma alternativa para o tempo consumido na construção dos DFD's de nível mais baixo.

Ainda segundo os autores, o modelo de prototipação diminuiria os riscos do projeto pois seriam melhor controlados e minimizados, já que esta técnica proporcionava um aprendizado gradativo por parte dos desenvolvedores e usuários. Depois de um entendimento inicial do problema, uma tentativa foi feita para executar rapidamente o que foi entendido, então sucessivas iterações e refinamentos foram feitos até que um produto final ficou completo e aprovado pelos usuários. Os processos usados para construir DFD's e protótipos são parecidos, sendo que a principal diferença é que o protótipo é um modelo prático que mostra de forma clara o que o sistema deve fazer. Finalmente, os autores, na época, apontaram como as principais vantagens de um protótipo sobre um DFD que:

- um protótipo fornecia um modelo prático mostrando ao usuário exatamente como o sistema será executado;
- um protótipo podia ser usado para geração de testes;
- um protótipo facilitava a transição da análise para o projeto;
- um protótipo era um processo evolutivo de aprendizagem (característica heurística).

Não se pode esquecer que o posicionamento e trabalhos apresentados são da década 80, quando as ferramentas e o ambiente de desenvolvimento eram muito diferentes dos dias atuais. Nessa época os sistemas eram, em sua maioria:

- desenvolvidos para computadores de grande porte;
- as linguagens de programação eram procedimentais, sendo as mais usadas o Cobol, Fortran, Pascal, etc ;
- o foco do desenvolvimento era as funcionalidades e não a usabilidade. O importante era que o sistema fizesse exatamente o que deveria fazer;
- as interfaces com o usuário eram alfanuméricas. Os desenvolvedores não se preocupavam com a satisfação dos usuários em relação a essas interfaces;
- o analista desenvolvia o sistema e o usuário "tinha" que usá-lo.

No início da computação não havia grande preocupação com as interfaces dos sistemas. A grande preocupação dos analistas era a de que o software realizasse corretamente as tarefas para as quais foi proposto, mesmo porque a entrada de dados era feita através de dispositivos que não permitiam flexibilidade, tais como: leitoras de

cartões, de fitas de papel, de fitas magnéticas, etc. O resultado do processamento, por sua vez, era armazenado em dispositivos magnéticos ou impresso.

Com o aparecimento dos terminais alfanuméricos já começa a haver uma nova visão sobre a importância das interfaces. Mesmo assim ainda era uma preocupação secundária, visto que poucas empresas possuíam computador, os sistemas eram poucos e altamente especializados e existia uma equipe para operar os equipamentos. Conforme lembra Martinez (2002, p. 16):

> "Nos 80's o desenvolvimento de interfaces visava um usuário especializado que tinha acesso aos caros e raros sistemas de computação".

2.3 A prototipação na era da ubiqüidade

Nos início dos anos 90 houve uma grande mudança no ambiente de desenvolvimento de software iniciada com a disseminação do uso dos computadores pessoais. O uso extensivo dos microcomputadores provocou um aculturamento geral no uso da informática, fazendo com que os usuários passassem a exigir que os softwares não só realizassem as tarefas exigidas, mas também, da forma que eles desejavam e não mais como os analistas queriam. O aparecimento das linguagens visuais e da Internet tornou os usuários ainda mais exigentes quanto às interfaces dos sistemas. Este posicionamento dos usuários fez com que a classe dos desenvolvedores de software tivessem que vê-los como clientes, tendo assim que satisfazer suas exigências. Como citado por Martinez (2002, p. 14).

> "Quando o conceito de interface surgiu para sistemas computacionais, era geralmente entendido como o hardware e o software através dos quais ocorre a comunicação humano-computador. A evolução levou à inclusão dos aspectos cognitivos e emocionais do usuário durante a comunicação".

Todas essas mudanças fizeram com que nos últimos anos duas novas áreas de conhecimento fossem formadas dentro da engenharia de sistemas: a Engenharia de Usabilidade e a Engenharia de Requisitos.

Nielsen (1993) advoga que a prototipação é uma parte importante do processo de usabilidade. As avaliações da usabilidade podem ser baseadas em um protótipo do sistema final, que pode ser desenvolvido de forma mais rápida e muito mais barata, e que pode ser mudado várias vezes até que um melhor entendimento do projeto da interface com o usuário seja conseguido. Muitos aspectos do projeto das interfaces, especialmente os dinâmicos, são difíceis de serem documentados mas podem ser facilmente entendidos olhando-se um protótipo em funcionamento. A prototipação também ajuda a se alcançar a consistência do produto, já que o protótipo é uma amostra inicial do que o projeto está objetivando.

Ele ainda defende que nos modelos tradicionais de engenharia de software muito do tempo de desenvolvimento é gasto com a geração de vários produtos intermediários, sendo que os programas executáveis são produzidos no último momento possível.

Dentro da Engenharia de Requisitos, conforme citado no SWEBOK (2002)[4], os protótipos são considerados uma ferramenta valiosa para se esclarecer requisitos difíceis de serem entendidos. Eles são similares aos cenários na medida em que criam um contexto dentro do qual os usuários entendem melhor que informações precisam fornecer. O uso de protótipos para a validação dos requisitos permite uma melhor integração com os usuários. A prototipação é empregada, normalmente, tanto para a validação da interpretação feita pelo engenheiro de requisitos sobre os requisitos do sistema, como para o levantamento de novos requisitos. Uma das vantagens do protótipo é que ele permite a avaliação das ações dinâmicas de uma interface, o que não é possível ser feito com uma descrição textual ou modelos gráficos.

Como será visto nos Capítulos 3 e 4 deste trabalho, a prototipação desempenha um papel muito importante tanto na Engenharia de Usabilidade como na Engenharia de Requisitos. No entanto, seu papel é diferente nas duas abordagens. Na Engenharia de Usabilidade, como o foco é o usuário, o protótipo está no centro do processo, como ferramenta fundamental para a avaliação do desempenho humano sobre a interface visando as metas de usabilidade. Na Engenharia de Requisitos, a visão é a de implementação, e o protótipo é ferramenta fundamental para a captura, análise e validação dos requisitos do software.

Os próximos capítulos contemplam com maior detalhe a prototipação na Engenharia de Usabilidade e na Engenharia de Requisitos.

2.4 Tipos de protótipos

Existem na literatura várias classificações para os protótipos, dependendo das características que se quer abordar. Apresentam-se a seguir algumas das principais classificações encontradas na literatura.

É importante salientar que, na elaboração de protótipos, estas categorias não são usadas de forma excludente. Um protótipo pode pertencer a mais do que uma categoria,

[4] SWEBOK – *Software Engineering Body of Knowledge* - é um dos resultados do esforço conjunto da IEEE Computer Society e de ACM - Association for Computing Machinery no sentido de regulamentar a profissão de Engenheiro de Software. O SWEBOK é um guia para o corpo de conhecimento da Engenharia de Software atual e foi publicado em 2001, contemplando as seguintes áreas de conhecimento: Requisitos de Software, Projeto de Software, Construção de Software, Teste de Software, Manutenção de Software, Gerência de Configuração de Software, Gerência de Engenharia de Software, Processo de Engenharia de Software, Ferramentas e Métodos de Engenharia de Software e Qualidade de Software.

como por exemplo: o protótipo exploratório, normalmente, é de baixa fidelidade, implementa cenários e é descartável.

2.4.1 Classificação dos protótipos segundo seus objetivos

Lichter *et al.* (1993, p. 222-223), apresentam a classificação dos protótipos pelos seus objetivos. Nesta classificação pode-se distinguir os seguintes tipos:

- **Prototipação exploratória** é usada quando o problema em questão não está claro. As idéias iniciais são usadas como uma base para o esclarecimento dos requisitos dos usuários. Ela permite que muitas opções de projeto sejam examinadas não restringindo prematuramente as idéias. Os desenvolvedores ganham conhecimento na área de aplicação, nas tarefas dos usuários e nos problemas que vão encontrar.

- **Prototipação experimental** foca na implementação técnica de um desenvolvimento alvo. Através da experimentação, os usuários são capazes de avaliar se a solução está completa e especificar melhor suas idéias.

- **Prototipação evolucionária** não é meramente uma ferramenta no contexto de um único projeto de desenvolvimento; é um processo contínuo para a adaptação de uma aplicação às mudanças. Isto significa que o desenvolvimento de software não é mais visto como um projeto contido em si mesmo, mas um processo de acompanhamento contínuo da aplicação. Uma conseqüência disto é que os desenvolvedores se tornam consultores técnicos, trabalhando continuamente em estreita cooperação com os usuários para melhorar a aplicação. Este tipo de prototipação está relacionada com o ciclo incremental.

2.4.2 Classificação dos protótipos segundo o grau de fidelidade

Uma outra forma de classificação da técnica de prototipação é a citada por Hakim e Spitzer (2000, p. 49), onde se classifica pelo grau de fidelidade e interatividade. A fidelidade refere-se ao grau com que o protótipo reflete largura, profundidade e aperfeiçoamento do produto pretendido.

Protótipos de baixa fidelidade descrevem partes da aplicação pretendida e são construídos sem muito esforço em obter "aderência e aperfeiçoamento". Eles normalmente são construídos em papel, *cardboard mockup* ou "*Power Point"* com as interfaces, para revisão com pequenos grupos de usuários. Para minimizar os custos destes protótipos de baixa fidelidade, eles raramente incluem capacidade de interatividade.

Protótipos de alta fidelidade usualmente referem-se a versões incompletas do produto final, construído usando-se ferramentas, mas faltando a implementação completa de certas funcionalidades. Essas versões do produto são consideradas como protótipos, porque são demonstradas para os usuários e clientes tão cedo quanto possível no processo de desenvolvimento, para que se possa fazer ajustes baseados nelas.

2.4.3 Classificação dos protótipos pelo nível de funcionalidade

Pode-se classificar o tipo de prototipação pelo nível de funcionalidade implementada no protótipo. A idéia por trás da prototipação é economizar tempo e custo de desenvolvimento, obtendo-se alguma coisa que pode ser testada por usuários reais antes do desenvolvimento completo. Esta economia somente pode ser obtida se o protótipo for reduzido em comparação com o sistema completo, ou seja: cortando o número de funcionalidades no protótipo ou reduzindo o nível das características das funcionalidades, de tal forma que ele "parece fazer" mas, realmente, não faz nada. Estas duas dimensões são ilustradas na Figura 2, extraída de Nielsen (1993, p. 49).

A redução do número de funcionalidades é chamada **prototipação vertical,** já que o resultado é um sistema mais estreito que inclui as funcionalidades em profundidade, mas somente para um conjunto com poucas funcionalidades selecionadas. Um protótipo vertical pode testar apenas uma parte limitada do sistema inteiro, mas ele será testado em profundidade, em circunstâncias realistas e com tarefas reais do usuário, podendo-se avaliar o desempenho, inclusive.

A redução do nível de implementação das funcionalidades é chamada de **prototipação horizontal**, também chamada de protótipo "casca de ovo", já que o resultado é uma camada superficial que inclui toda a interface com o usuário, para todas as características do sistema, mas sem nenhuma funcionalidade subjacente. Um protótipo horizontal é uma simulação da interface, onde nenhum trabalho real pode ser realizado.

A prototipação horizontal torna possível o teste de toda a interface com o usuário, mesmo sabendo que o teste é menos realista, já que os usuários não podem realizar qualquer tarefa real num sistema sem funcionalidades. A principal vantagem do protótipo horizontal é que eles podem, freqüentemente, ser implementados rapidamente, com o uso de várias ferramentas de desenho de tela e prototipação e podem ser usados para avaliar quão bem a interface inteira trabalha e sentir o sistema como um todo.

Finalmente, pode-se reduzir ambos, o número de características e o nível das funcionalidades, para se chegar a um **cenário**, que é capaz apenas de simular as interfaces com o usuário, fazendo com que o usuário teste seguindo um caminho planejado previamente. Cenários são extremamente fáceis e baratos de construir, enquanto que ao mesmo tempo não são particularmente realistas. Rizzo *et al.* (1997, p. 305) dizem que:

> “cenários providenciam um contexto bem estabelecido para descrever modalidades de interação. Assim, cenários podem também ser vistos como dispositivos para facilitar a comunicação entre membros do time de projeto”.

O protótipo horizontal deve ser usado quando se deseja levantar e validar todas as interfaces que o produto terá. O protótipo vertical é indicado para o estudo e detalhamento de uma determinada parte do sistema, como por exemplo, quando queremos verificar a performance de uma determinada funcionalidade.

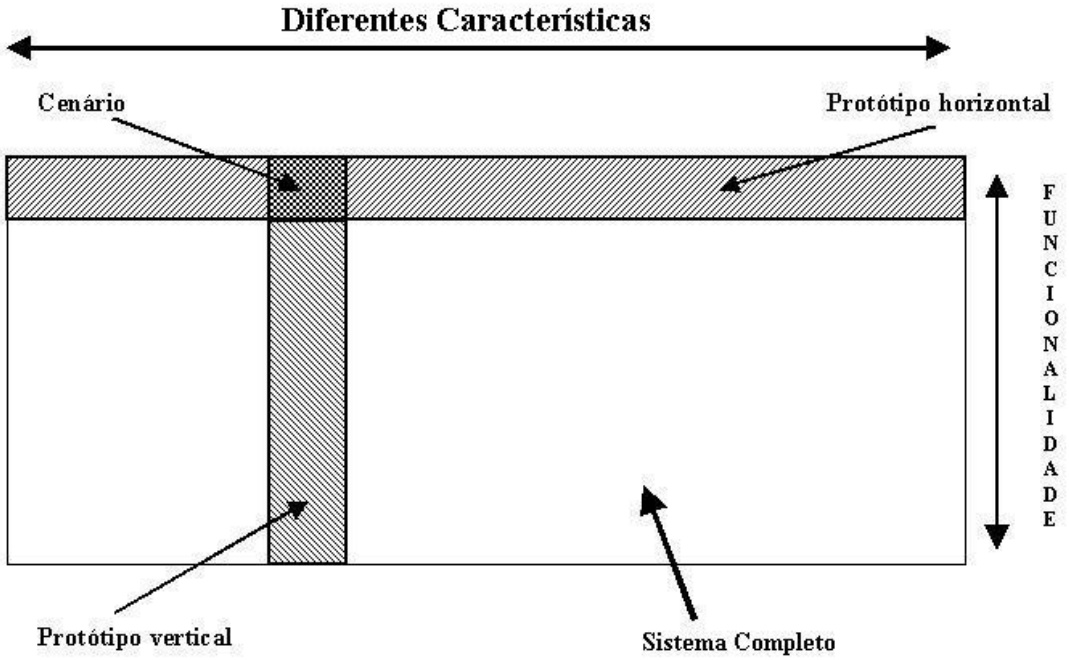

Figura 2. As duas dimensões da prototipação: horizontal e vertical.

2.5 Classificação dos protótipos pela técnica de construção

Uma outra forma de se classificar os protótipos é pela sua técnica de construção. Pode-se classificá-los em protótipo descartável e protótipo reutilizável.

Protótipo descartável é aquele que é desenvolvido e utilizado num único projeto, sendo abandonado no final do mesmo. Normalmente são desenvolvidos em papel ou em uma ferramenta de prototipação.

Protótipo reutilizável é aquele que é desenvolvido de forma a ser reaproveitado no desenvolvimento de uma família de produtos, ou até mesmo, em produtos de famílias diferentes. Os protótipos reutilizáveis, normalmente, são desenvolvidos numa linguagem que permite a componentização da interface, de preferência a linguagem que será utilizada para o desenvolvimento do software. Este tipo de protótipo ajuda a acelerar o ciclo de desenvolvimento além de garantir a consistência entre os produtos de uma mesma família.

Como vimos neste capítulo, o protótipo surgiu como uma ferramenta para o levantamento e a validação dos requisitos de software em casos bem específicos, desenvolvido com ferramentas de prototipação, numa linguagem diferente daquela que era usada no produto final e com o protótipo sendo descartado ao final da validação. Mesmo assim, era considerada uma ferramenta valiosa.

Na medida em que os ambientes e linguagens de desenvolvimento foram se modificando e que as interfaces passaram a ter maior importância, o uso do protótipo também ganhou impulso, na medida em que permite validar não só os requisitos funcionais, como também, os requisitos não funcionais, principalmente aqueles voltados às questões de usabilidade, que passaram a ser fundamentais na era da ubiqüidade.

Frente a isso, no Capítulo 3, serão tratadas as questões de usabilidade e como a prototipação se insere na Engenharia de Usabilidade.

Capítulo 3. Engenharia de Usabilidade

Nos ambientes visuais de desenvolvimento, as interfaces têm um papel muito importante na qualidade e aceitação do produto, fazendo com que os profissionais de Engenharia de Software se preocupem com as questões de usabilidade dos seus produtos.

Segundo Nielsen (1993), estudos feitos para analisar as causas que levaram projetos a ultrapassarem seus orçamentos apontaram como razões principais:

- requisições freqüentes de mudanças feitas pelos usuários;
- omissão de tarefas;
- falta de entendimento dos usuários dos seus próprios requisitos, e
- comunicação e entendimento insuficiente entre o usuário e o analista.

O uso da metodologia da engenharia de usabilidade, bem como da prototipação, poderia minimizar muitos desses problemas, reduzindo assim o estouro nos custos dos projetos de software. Nielsen (1993) cita que ainda existe um certo temor por parte dos gerentes em usar os princípios e técnicas da usabilidade, porque consideram que são caros e complexos. No entanto, num estudo sobre gastos com usabilidade em 31 projetos, obteve-se como resultado que apenas 6% dos orçamentos dos projetos foram gastos com atividades de usabilidade. Frente à melhoria do produto e diminuição com gastos de manutenção, as atividades de usabilidade trazem uma relação custo X benefício muito boa.

Neste capítulo são apresentados os conceitos de usabilidade e a área de engenharia de usabilidade. Isto se faz necessário na medida em que para se fazer protótipos coerentes e que possibilitem a criação de software que atenda às necessidades dos usuários, precisa-se utilizar os conceitos e técnicas sugeridas pela engenharia de usabilidade.

3.1 Usabilidade

Em primeiro lugar precisa-se caracterizar o que é Usabilidade. Panoui (1999, p. 1) define **usabilidade** da seguinte maneira:

> "De acordo com a norma ISO 9241 (93)[5] usabilidade é 'a efetividade e a satisfação com as quais usuários específicos conseguem alcançar objetivos

[5] ISO 9241 - Ergonomic Requirements for Office Work with Visual Display Terminals (VDTs) – norma composta por 17 partes. Não existe a norma da ABNT.

> específicos em ambientes particulares', onde efetividade é 'a precisão e completude com as quais usuários atingem objetivos específicos', eficiência é 'a precisão e completude dos objetivos em relação aos recursos gastos' e satisfação é 'o conforto e a aceitabilidade do sistema, e são usualmente chamados de atributos de usabilidade".

Panoui (1999, p. 1) ainda diz que:

> "Métodos de projeto de interfaces centrados no usuário têm que atender todos os tipos de necessidades do usuário (isto é, funcional, física e suas aspirações), para alcançar uma usabilidade maximizada no produto final."

O National Cancer Institute (2003) define usabilidade da seguinte forma:

> "Usabilidade é uma medida da qualidade de uma experiência do usuário quando interagindo com um produto ou sistema – seja um *Web site*, uma aplicação de software, tecnologia móvel, ou qualquer dispositivo operado pelo usuário".

Segundo Nielsen (1993), **usabilidade** não é uma propriedade simples e unidimensional de uma interface com o usuário. A usabilidade tem múltiplos componentes e é tradicionalmente associada com os seguintes cinco atributos:

- **Facilidade de aprendizagem**: O sistema favorecer a que o usuário possa rapidamente começar a trabalhar com ele.
- **Eficiência**: uma vez que o usuário tenha aprendido a usar o sistema, um alto nível de produtividade seja possível.
- **Facilidade de memorização**: o usuário casual deve ser capaz de retornar ao sistema depois de um período sem usá-lo, sem ter que aprender tudo de novo.
- **Erros**: o sistema deve ter uma baixa taxa de erros durante seu uso. Erros catastróficos não devem ocorrer. Deve-se prever mecanismos de recuperação de erros.
- **Satisfação**: os usuários devem ficar satisfeitos ao usar o sistema, e gostar dele.

Pela definição de conceitos abstratos de usabilidade em termos de seus componentes mais importantes e mensuráveis pode-se chegar a uma disciplina de engenharia onde a usabilidade é não só questionada mas também sistematicamente alcançada, melhorada e avaliada.

A avaliação de usabilidade é etapa central do processo e pode ser feita de duas formas. Em ambas, a usabilidade é medida através de testes feitos por um grupo selecionado de usuários, que usa o sistema para realizar um conjunto pré-estabelecido de

tarefas. No teste formal de usabilidade, os parâmetros são medidos pelos usuários reais, em campo, executando aquelas tarefas que eles têm que fazer normalmente. Na inspeção de usabilidade, especialistas nos atributos de usabilidade percorrem as tarefas em busca de defeitos. De qualquer modo, um ponto importante é que a usabilidade não é uma medida absoluta e sim uma medida relativa a certos usuários e certas tarefas. Pode-se então ter o caso em que o mesmo sistema seja avaliado como tendo diferentes características de usabilidade, se usado por diferentes usuários e para diferentes tarefas.

Nielsen (1993) sugere que algumas atitudes simples, tomadas pela equipe de desenvolvimento, podem melhorar a usabilidade dos produtos de software resultantes. Atividades simples e centradas no usuário, formam a base da engenharia de usabilidade:

- Conhecer as pessoas que serão os usuários de seu projeto;
- Conversar sobre como o sistema deve ajudar um usuário;
- Visitar o ambiente de trabalho dos usuários e observar quais são suas tarefas, como eles realizam estas tarefas e com que circunstâncias eles têm que lidar.

Nielsen (1993) lembra que existem métodos mais avançados mas advoga que uma simples ida ao campo, para observar o usuário no seu ambiente de trabalho, realizando sua tarefas cotidianas no mundo real, pode, freqüentemente, fornecer muitas idéias sobre usabilidade.

O uso dos métodos da engenharia de usabilidade pode gerar economia, tanto com a diminuição do tempo gasto pelos usuários na execução de suas tarefas, como pela descoberta de características não necessárias, mas que os desenvolvedores imaginavam ser necessárias.

A avaliação e comparação de usabilidade relativa entre produtos competitivos é bem conhecida na indústria de software e é usada pelos vendedores para facilitar a venda de seus produtos. Produtos com melhores características de usabilidade têm melhor aceitação por parte do cliente.

A engenharia de usabilidade é um conjunto de atividades que, idealmente, ocorrem através de todo o ciclo de vida do produto, com atividades significativas acontecendo nos estágios iniciais, antes que as interfaces do usuário tenham sido projetadas. A usabilidade não pode ser vista separadamente do contexto de desenvolvimento de produtos da corporação. Ao invés disso, a usabilidade se aplica ao desenvolvimento da família inteira de produtos, bem como das versões subsequentes deles.

Segundo definição obtida no site do National Cancer Institute (2003):

“Engenharia de Usabilidade é um enfoque metódico para produzir um Web site ou qualquer interface do usuário. Ela é um modo prático e sistemático para entregar um produto que trabalha para os usuários. A Engenharia de Usabilidade envolve vários

métodos, cada um dos quais aplicado para trabalhar para os usuários, cada um aplicado no tempo apropriado, incluindo captura dos requisitos, desenvolvimento e testes de protótipos, avaliação de *design* alternativos, análise de problemas de usabilidade, propostas de solução e testes do site (ou outras interfaces) com os usuários."

3.2 Prototipação no ciclo de vida da Engenharia de Usabilidade

No ciclo de vida da Engenharia de Usabilidade, proposto por Nielsen (1993), a prototipação aparece como uma das etapas. No entanto, os protótipos podem ser usados em diversas etapas deste ciclo de vida. Apresenta-se a seguir um resumo de cada uma das etapas, conforme o autor e uma análise crítica de como a prototipação pode ser usada em várias delas.

3.2.1 Conhecer o Usuário

O primeiro passo no processo de usabilidade é estudar os futuros usuários e o uso que farão do produto. Uma visita ao local onde o cliente trabalha, fornece aos desenvolvedores uma visão melhor de como o produto será usado. As características individuais dos usuários e a variabilidade nas suas tarefas são os dois fatores de maior impacto na usabilidade, assim eles precisam ser estudados com cuidado. O conceito de usuário usado aqui é mais amplo, incluindo todas as pessoas cujo trabalho é afetado de alguma forma pelo produto. Para se conhecer os usuários é necessária uma análise das: suas características individuais, tarefas a serem realizadas, razões funcionais para as tarefas e lembrar que existe uma evolução deles ao longo do tempo, conforme descrito a seguir. O conhecimento do usuário define as características de usabilidade a serem implementadas no protótipo.

Características individuais dos usuários

As características individuais dos usuários influenciam, de forma direta, na escolha dos parâmetros de usabilidade que serão prioritários no desenvolvimento do protótipo. Alguns fatores são muito importantes para antecipar dificuldades de aprendizagem e definir melhor os limites da complexidade das interfaces. O conhecimento sobre o trabalho, o nível de educação, a idade, a experiência prévia com computadores, o perfil de linguagem e leitura dos usuários influenciam no projeto das interfaces. O tempo que poderão dispor para aprender e se terão oportunidade para ter um mínimo de treinamento, também são fatores importantes. O ambiente de trabalho dos usuários e o contexto social também precisam ser conhecidos. Quando o produto está sendo desenvolvido para um determinado departamento, numa dada empresa, esta não é uma tarefa muito difícil, mas quando o produto é desenvolvido para um grande número de usuários ou para o mercado esta tarefa fica bem mais difícil.

No caso de existirem outros sistemas já em uso na empresa, deve-se levantar quais são os mais aceitos em termos da interação humano-computador. Isto trará informações sobre o gosto pessoal dos usuários e pode fornecer uma base para o desenvolvimento das interfaces mais apropriadas. Se não existirem outros softwares, pode-se apresentar vários modelos de interface e verificar qual será o mais aceito. Esta informação servirá de base para a construção do protótipo que melhor se adeqüe às características de seus usuários.

Análise das tarefas

A análise das tarefas a serem realizadas é essencial no projeto do sistema. Nesta atividade deve-se: estudar os objetivos gerais dos usuários, sua visão atual da tarefas e das informações necessárias, e como eles lidam com circunstâncias especiais e emergências. O modelo do usuário para suas tarefas deve também ser identificado, já que pode ser usado como uma fonte de idéias para a elaboração das interfaces. Deve-se também, buscar e observar os usuários realmente efetivos e os estratégicos e o andamento do trabalho, como sugestões do que o sistema deverá suportar. Finalmente, deve-se identificar os pontos fracos da situação atual: pontos onde os usuários falham na obtenção dos objetivos, gasto excessivo de tempo, ou que são feitos de forma desconfortável. Estas fraquezas apresentam oportunidades de melhoria no novo produto.

No caso de já existir um software que automatize as tarefas ou parte delas, ele pode servir como um protótipo inicial para a análise de tarefas. Analisar seus pontos fracos e fortes pode fornecer uma boa base para o novo produto.

Como resultado da análise de tarefas deve ser obtida uma lista de todas as coisas que os usuários desejam implementar no sistema, toda a informação que eles precisarão para alcançar seus objetivos, o fluxo de trabalho, todos os resultados e relatórios que precisam ser produzidos, os critérios usados para determinar a qualidade e aceitabilidade dos resultados e as necessidades de comunicação entre os usuários, tanto na realização das tarefas quanto na preparação delas. Protótipos exploratórios, desenhados em papel, também são uma boa ferramenta para o entendimento entre os usuários e os analistas sobre as tarefas a serem realizadas.

Para a documentação das tarefas de usuários, consultar Greiner (2000).

Análise Funcional

Na análise funcional deve-se estudar as razões funcionais implícitas para as tarefas, o que realmente precisa ser feito, e o que são procedimentos meramente superficiais e que podem e devem ser mudados. Muitas vezes, uma tarefa é realizada por costume ou tradição. Uma avaliação criteriosa de sua real necessidade e da possibilidade de simplificação pode resultar num processo de trabalho muito mais efetivo do que o

atual. Obviamente, há um limite de quão drasticamente se pode mudar o modo como os usuários fazem suas tarefas: assim, a análise funcional deve ser coordenada com a análise de tarefas.

O protótipo implementa as tarefas dos usuários. Quando o usuário vê como o protótipo funciona é capaz de ter idéias de como melhorar e otimizar o processo vigente, fazendo a reengenharia do processo.

Evolução do usuário

É importante não projetar um software apenas para a forma que os usuários o usarão logo depois de liberado. Na medida em que os usuários o usam, eles próprios são mudados pelo sistema, e já que eles mudam irão usá-lo de um novo modo. Um projeto mais flexível terá uma chance melhor de suportar estes novos usos.

3.2.2 Análise Competitiva

A análise competitiva usa produtos similares, já existentes, sejam da própria empresa ou de outro concorrente, para fornecer parâmetros de usabilidade para o produto a ser desenvolvido. Esses produtos são os melhores protótipos que se pode ter para o novo produto. Como eles já estão totalmente implementados é muito fácil utilizá-los para a realização de testes com os usuários. Os principais motivos são:

- Através de produtos similares pode-se realizar testes empíricos para estabelecer regras de usabilidade, com os usuários utilizando estes produtos;
- Os desenvolvedores dos sistemas existentes tiveram um razoável esforço, no seu processo de desenvolvimento, para que o produto pudesse trabalhar bem. Isto significa que os testes realizados pelos usuários com produtos existentes podem ser mais realistas do que testes com outros tipos de protótipo;
- Os usuários podem executar suas tarefas reais no sistema existente, permitindo que se aprenda, bem como suas funcionalidades e técnicas de interação suportam os tipos de tarefas que o novo sistema deve suportar.

Se existirem muitos produtos similares disponíveis, é possível fazer-se uma análise comparativa dos diferentes projetos de interface. Isto fornecerá idéias para o novo projeto e uma lista de regras sobre o que parece funcionar e o que deveria ser abandonado.

A análise competitiva proposta por Nielsen (1993) é uma ferramenta muito interessante para fornecer idéias sobre um novo produto. No entanto, existem pontos que dificultam esta atividade, tais como:

1. O software pode ser novo não existindo, portanto, outros produtos similares na empresa ou no mercado;
2. Dificilmente o usuário ou a equipe de desenvolvimento possui outros produtos concorrentes. Para que isso pudesse ser feito, seria necessária a compra e implantação desses produtos, o que seria muito dispendioso. Acredita-se ser difícil que isto possa ser realizado;
3. Somente no caso de se estar desenvolvendo uma nova versão de um produto existente é que seria factível. Neste caso, esta versão poderia ser usada como um "protótipo" para a avaliação das questões de usabilidade.

3.2.3 Estabelecimento de metas de usabilidade

A usabilidade compreende vários componentes que, algumas vezes, conflitam entre si. Em alguns projetos a facilidade de aprendizagem pode ser mais importante do que a eficiência, em outros um baixa taxa de erros pode ser fundamental. Em cada projeto deve-se estabelecer as prioridades de cada componente e atribuir pesos, tomando-se como base a análise dos usuários e de suas tarefas. É importante discutir-se as métricas de usabilidade que são de interesse para o projeto e especificar os objetivos da interface, em termos de medidas de usabilidade.

Paralelamente ao estabelecimento das metas de usabilidade, é uma boa idéia fazer uma análise do impacto financeiro da usabilidade do sistema. Esta análise envolve uma estimativa do impacto sobre a organização que o desenvolve, para ajudar a determinar o orçamento de usabilidade, e sobre a organização do usuário, relacionada com a economia de tempo dos usuários na realização de suas tarefas.

3.2.4 Projeto paralelo

O projeto paralelo consiste em se explorar diferentes alternativas de projeto, antes de assumir o enfoque definitivo, que será detalhado e desenvolvido. Esta técnica é especialmente útil no desenvolvimento de sistemas novos, onde não se tem nenhuma orientação de qual é a melhor interface. O projeto paralelo permite que se tenha diferentes visões dos modelos de interface de usuário e diferentes detalhes em seus projetos. Pode-se obter o novo projeto combinando as características dos projetos tentativos gerados.

Uma variante do projeto paralelo é chamado projeto paralelo diversificado que é baseado em se projetar as interfaces para diferentes aspectos do problema. Por exemplo: pede-se a um projetista para projetar a interface para usuários novatos, a outro para

desenvolver para usuários experientes e a outro para explorar a possibilidade de interfaces não verbais.

A criação de protótipos paralelos pode ser muito dispendiosa e consumir um tempo precioso, no caso do projetos com ciclo acelerado. Uma sugestão é criar protótipos paralelos apenas com as principais telas, de forma a avaliar diferentes possibilidades sem despender muito tempo.

3.2.5 Projeto participativo

A idéia do projeto participativo é incorporar, à equipe de projeto, um conjunto de usuários representativos, que irão colaborar fornecendo idéias e avaliando os projetos de interface propostos. Existem questões, relativas ao projeto, que somente os usuários conseguem responder. Eles freqüentemente levantam questões que a equipe de desenvolvimento não havia sequer pensado, tais como: potenciais erros nas tarefas reais e no modelo do processo de trabalho. Eles devem ser envolvidos no projeto de telas através de encontros regulares entre projetistas e usuários.

Os usuários sabem avaliar projetos concretos, sabem do que não gostam e o que é ou não é prático para trabalhar. Para conseguir bons resultados do envolvimento dos usuários, é necessário apresentar as sugestões de projeto do sistema numa forma que os usuários possam entender. Ao invés de volumosas especificações do sistema, protótipos, cenários ou protótipos exploratórios em papel são mais eficientes.

3.2.6 Projeto Coordenado da Interface Total

A consistência é considerada uma das características mais importantes da usabilidade, e deve ocorrer nos diferentes meios que formam a interface total do usuário, incluindo não apenas as telas da aplicação, mas também a documentação, o sistema de *help online*, tutoriais *online* ou em vídeo tape, bem como o treinamento tradicional.

A consistência deve ser mantida nas sucessivas versões de um produto, de tal forma que uma nova versão seja consistente com as versões anteriores e dentro da família inteira de produtos. Padrões corporativos de interfaces com o usuário são um modo comum de se conseguir este objetivo. A criação de *templates* das interfaces colabora na obtenção da consistência desejada.

A consistência também pode ser aumentada pelo compartilhamento de código ou de um ambiente de desenvolvimento restrito.

3.2.7 Diretrizes e avaliações heurísticas

Avaliação heurística é um método da engenharia de Usabilidade para encontrar problemas de usabilidade no projeto de interface com o usuário, de modo a que eles possam ser atendidos como parte de um processo de projeto iterativo. É um método informal e envolve especialistas de usabilidade para julgar se cada elemento do diálogo segue princípios de usabilidade estabelecidos, segundo Nielsen (1995a e 1995b).

Em geral, a avaliação heurística é difícil de ser realizada por um único indivíduo porque uma pessoa dificilmente poderá encontrar todos os problemas de usabilidade em uma interface. A experiência tem mostrado que pessoas distintas encontram diferentes problemas de usabilidade. Portanto, é possível melhorar significativamente a eficácia do método envolvendo múltiplos avaliadores.

Na avaliação heurística cada avaliador inspeciona a interface sozinho. Somente depois que todas as avaliações terminam é que os avaliadores podem se comunicar e agregar suas descobertas. Este procedimento é importante para assegurar avaliações independentes e não tendenciosas de cada avaliador. Os resultados da avaliação podem ser registrados como relatórios escritos de cada avaliador ou deixando que os avaliadores verbalizem seus comentários a um observador enquanto percorrem a interface.

Durante a sessão de avaliação, o avaliador percorre a interface diversas vezes e inspeciona os vários elementos do diálogo comparando-os com uma lista de princípios reconhecidos de usabilidade (as heurísticas). Além da lista geral é permitido que o avaliador considere todos os princípios adicionais de usabilidade. É possível, também, desenvolver-se heurísticas específicas para uma categoria de produtos.

O resultado do método é uma lista de problemas de usabilidade da interface com a referência aos princípios de usabilidade que foram violados, na opinião do avaliador.

As diretrizes listam princípios bem conhecidos de projeto de interfaces de usuário que devem ser seguidos no desenvolvimento dos projetos. Num projeto, vários níveis diferentes de diretrizes devem ser usadas: diretrizes gerais, aplicáveis a todas as interfaces de usuários; diretrizes específicas da categoria, para o tipo de sistema que está sendo desenvolvido; e diretrizes específicas do produto, que são criadas para o produto em particular.

As diretrizes podem ser usadas como base para a avaliação heurística. Por exemplo: uma diretriz geral poderia ser “prever retorno aos usuários sobre as ações e estados do sistema”.

A diferença entre um padrão e uma diretriz é que um padrão especifica como a interface deve aparecer para o usuário, enquanto que um conjunto de diretrizes fornecem recomendações sobre as características de usabilidade da interface.

3.2.8 Prototipação

A prototipação é um processo iterativo através do qual se desenvolve um protótipo do sistema final, de forma mais rápida e muito mais barata, podendo ser mudado várias vezes até que um melhor entendimento do projeto da interface com o usuário seja conseguido, isto é, a função do protótipo é permitir a avaliação de usabilidade antes mesmo que o software esteja pronto.

Os protótipos podem algumas vezes ser usados para uma forma especial de projeto participativo, chamado prototipação interativa, onde o protótipo é feito e modificado durante as sessões de teste com os usuários, a partir dos comentários feitos sobre seus pontos fracos. Na prática, entretanto, mesmo um programador experiente cometerá erros na alteração do código em tempo real. Os erros de programação e o tempo gasto atrapalharão a sessão de teste, tirando o foco da avaliação das tarefas do usuário e da interface. Além disso, normalmente os usuários não têm tempo para ficar esperando as correções. Sugere-se o uso de simulações em papel para sessões de prototipação interativa, permitindo aos usuários modificar os desenhos em papel. As mudanças propostas são implementadas depois, no ambiente de desenvolvimento.

Um protótipo é uma forma de projeto da especificação e é freqüentemente usado como o principal meio de comunicação dos desenvolvedores.

3.2.9 Avaliação da interface

É muito importante que sejam feitos alguns testes com os usuários para a avaliação das interfaces. Mesmo que não se use todos os métodos corretos para avaliação do projeto, os benefícios do emprego de alguns métodos razoáveis de engenharia de usabilidade para avaliar uma interface com o usuário, ao invés de liberá-la sem avaliação, são muito grandes.

Quando um método qualquer de avaliação é usado, o principal resultado é uma lista dos problemas de usabilidade da interface e sugestões para novas características que suportarão as expectativas dos usuários. Nem sempre é viável resolver todos os problemas e, nesse caso, eles deverão ser priorizados. A priorização será melhor se baseada em dados experimentais sobre o impacto dos problemas na performance do usuário, mas algumas vezes só se pode usar a intuição.

Nielsen (1993) sugere o envio da lista dos problemas de usabilidade descobertos na interface, a um grupo de especialistas de usabilidade, pedindo a eles que atribuam a taxa de severidade de cada problema. Na prática, em equipes pequenas, não existe um profissional especialista de usabilidade, quanto mais um grupo. Portanto, esta avaliação é feita em conjunto pelos usuários e desenvolvedores mais experientes.

3.2.10 Projeto Iterativo

A partir dos problemas de usabilidade e oportunidades descobertas pelos testes empíricos, pode-se produzir uma nova versão da interface. Em alguns casos, soluções alternativas potenciais precisam ser projetadas e testadas antes de se tomar uma decisão. A familiaridade com várias opções de projeto, o conhecimento adquirido na observação dos usuários e a criatividade são necessárias. O desenvolvimento em ciclo acelerado nem sempre permite que se estude várias opções de projeto, então, a existência de um conjunto de *templates* pré-existentes pode acelerar o processo.

Mudanças feitas para resolver problemas de usabilidade podem introduzir novos problemas de usabilidade. Isto é uma outra razão para se combinar projeto iterativo e avaliação. De fato, é muito comum que um novo projeto foque na melhoria de certos parâmetros de usabilidade, para descobrir depois que algumas mudanças têm impacto adverso sobre outros parâmetros de usabilidade.

Deve-se lembrar que novos problemas de usabilidade aparecem nos testes depois que os mais óbvios problemas foram corrigidos, portanto, não há necessidade de testar exaustivamente o projeto inicial, uma vez que ele vai mudar de qualquer modo. A interface deve ser alterada e re-testada tão logo um problema de usabilidade tenha sido detectado e entendido, assim aqueles problemas que tinham sido mascarados pelo problema inicial passam a ser encontrados.

A manutenção de uma trilha de auditoria durante o desenvolvimento iterativo é importante. Já que mudanças na interface freqüentemente terão que ser feitas, é útil conhecer as razões que levaram a adoção de uma determinada solução no projeto original, para que não se repitam erros anteriores e se tenha controle das decisões tomadas.

3.2.11 Acompanhamento de Sistemas Instalados

O principal objetivo do trabalho de usabilidade, depois da liberação de um produto, é obter dados de usabilidade para a próxima versão e para futuros produtos. Da mesma maneira que produtos existentes e competitivos são os melhores protótipos para o produto na fase inicial de análise competitiva, um produto recém liberado pode ser visto como um protótipo de futuros produtos. Estudos de campo do uso do produto avaliam como os usuários reais usam a interface para a execução das suas tarefas no seu ambiente de trabalho e podem, portanto, trazer muitas idéias que não se conseguiria num estudo de laboratório.

Algumas vezes, o retorno de campo pode ser obtido como parte de estudos de *marketing*, através, por exemplo, de informações de satisfação dos clientes. Pode-se também conduzir um estudo específico para se obter informações de acompanhamento sobre o uso dos produtos liberados. Basicamente, podem ser usados os mesmos métodos de estudos de campo e análise de tarefas, incluindo entrevistas, questionários e estudos de observação. Além disso, dados de *log* se tornam especialmente valiosos por sua habilidade de indicar como o software está sendo usado através de uma variedade de tarefas. Pode-se também obter informações sobre a usabilidade do produto através da análise das reclamações dos usuários, pedidos de modificação dos requisitos e chamadas ao suporte.

3.3 Outros ciclos de vida com foco em usabilidade

A literatura apresenta outros ciclos de vida de desenvolvimento de software centrados no usuário, além do proposto por Nielsen (1993). Baseados neste princípio pode-se citar o método de engenharia de usabilidade segundo Mayhew (1999) e o modelo estrela, brevemente descritos a seguir.

Método proposto por Mayhew (1999)

O método proposto por Mayhew (1999) é composto basicamente por três etapas: análise de requisitos, *design*/testes/desenvolvimento e instalação.

A etapa de **análise de requisitos** possui as seguintes atividades:

- **levantamento do perfil do usuário** – o perfil do usuário influencia fortemente o estabelecimento dos atributos de usabilidade a serem implementados no protótipo;
- **análise de tarefas contextual** – o contexto das tarefas executadas pelos usuários afeta o protótipo a ser desenvolvido;
- **análise de plataforma** – a plataforma define a tecnologia a ser utilizada no desenvolvimento do protótipo; e
- **definição dos princípios gerais de *design* –** os princípios gerais de *design* são a base para a elaboração dos protótipos. Se forem usados *templates* pode-se simplificar esta atividade pela escolha de padrões de interface já estabelecidos.

Ao final desta etapa tem-se o Guia de Estilo do produto, que documenta as quatro atividades acima descritas, e os princípios gerais de *design* a serem aplicados ao produto.

A etapa de *design*/testes/desenvolvimento é dividida em três níveis:

- **design de alto nível**, onde se faz a reengenharia do trabalho, são geradas as alternativas iniciais do *design* de alto nível, feitas maquetes do modelo conceitual e é feita a avaliação iterativa do modelo conceitual até que esteja relativamente estável. O protótipo é uma ferramenta eficaz para a atividade de reengenharia do trabalho e avaliação do modelo conceitual. O uso de *templates* simplifica a análise de alternativas de *design* já que existem várias opções prontas para serem analisadas;

- **definição de padrões**, onde se desenvolve um conjunto de padrões para a aplicação, atividade que poderia ser substituída pela escolha de *templates* de um repositório existente, implementa-se um protótipo com um conjunto selecionado das funcionalidades da aplicação e faz-se a avaliação iterativa dos padrões de *design* de tela, usando o protótipo desenvolvido. No final desta atividade acrescenta-se ao Guia de Estilo, obtido na etapa anterior, os resultados do *design* do modelo conceitual e dos padrões de *design* de tela; e

- **finalização do *design***, onde é feito o *design* detalhado do restante da interface do usuário e a avaliação iterativa deste *design*, nesta atividade um protótipo horizontal é concluído.

Na etapa de instalação, depois da aplicação estar em produção a algum tempo, busca-se a avaliação do usuário, em uso real, para melhoria do *design*, ou para novas versões ou outros produtos.

Ciclo de vida estrela

O ciclo de vida estrela é centrado em avaliação e visa principalmente o *design*. A avaliação é peça fundamental em todos os estágios deste ciclo de vida. O ciclo de vida estrela, conforme apresentado por Martinez (2002, p. 23) em seu trabalho retirado de Hix & Hartson (1989).

> “visa principalmente o *design* e é centrado em avaliação. A avaliação é relevante em todos os estágios do ciclo de vida e não somente no fim do desenvolvimento do produto como sugere o modelo em cascata.”.

O modelo estrela, segundo Martinez (2002, p. 23):

> “apresenta uma abordagem de desenvolvimento como ‘ondas alternantes’, entre atividades. Elas são similares às do modelo cascata mas não há um único ponto inicial e não há uma ordem rígida para as atividades de desenvolvimento; o método estabelece apenas que cada uma seja seguida por algum tipo de avaliação. A avaliação deve ser do tipo apropriado para a atividade avaliada”.

A Figura 3 apresenta o ciclo de vida do modelo estrela conforme apresentado no trabalho de Martinez (2002, p. 23), onde se percebe que a avaliação é a parte central do método.

Novamente a prototipação aparece como uma etapa deste ciclo de vida. Neste trabalho, propõe-se que o protótipo seja usado para a especificação dos requisitos, em projetos com ciclo acelerado, então é sub-produto da etapa de análise dos requisitos. Ele também por ser usado como artefato que implementa o *design* proposto. Quanto a especificação de usabilidade, o protótipo é a implementação destas especificações exceto quanto a questões de eficiência e desempenho.

O protótipo também é artefato de entrada para a etapa de implementação.

Portanto, nos projetos com ciclo de vida acelerado, sugere-se o uso do protótipo em todas as etapas do ciclo de vida estrela.

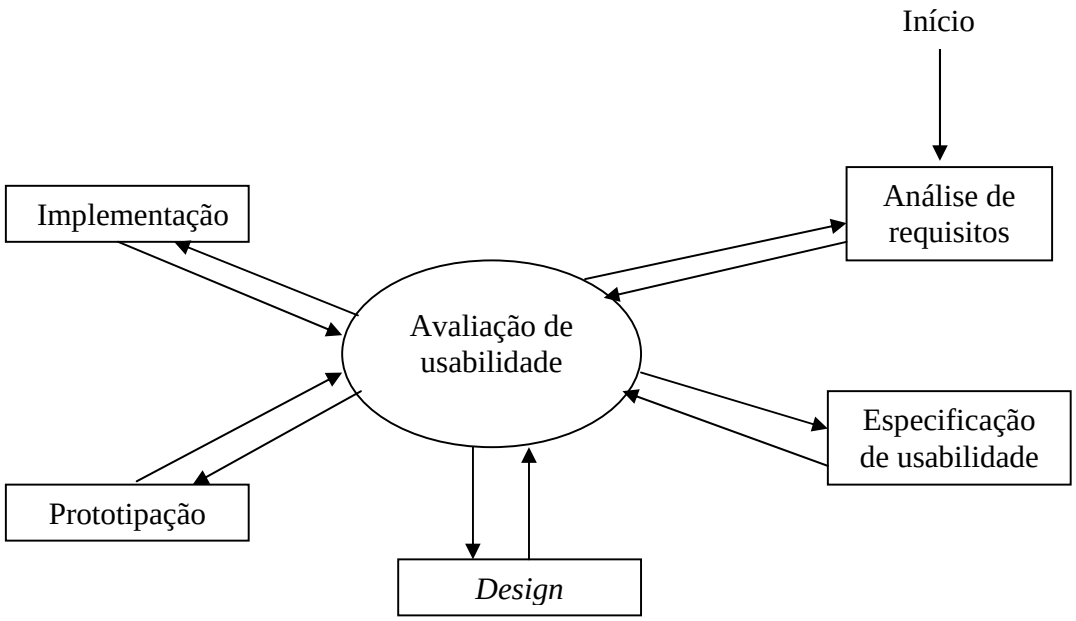

Figura 3. O ciclo de vida estrela, adaptada de Martinez (2002, p. 23)

3.4 Resumo

Em resumo, os métodos baseados em Engenharia de Usabilidade possuem algumas características comuns:

- O desenvolvimento é centrado no usuário e nas suas necessidades para a realização de suas tarefas;

- O processo de desenvolvimento não é mais uma seqüência de atividades que acabam num produto (como no modelo cascata) mas sim um processo iterativo que gera uma nova versão do software a cada iteração;

- Os interessados no software devem participar ativamente do processo, seja como avaliador ou como participante ativo das definições de projeto;

- A avaliação é extremamente importante no processo de desenvolvimento. Cada artefato produzido é avaliado;

- Os protótipos, sejam exploratórios, versões anteriores, softwares já existentes e que devem ser substituídos, ou um protótipo executável desenvolvido no projeto, passam a ter maior importância, visto que são uma ferramenta bastante adequada para o entendimento entre usuários e desenvolvedores.

Capítulo 4. A prototipação na Engenharia de Requisitos

Apresenta-se neste capítulo a caracterização da área de conhecimento de Engenharia de Requisitos, e como a prototipação se insere nas atividades propostas, facilitando e acelerando as suas etapas. O enfoque seguido é o apresentado no SWEBOK (2002).

Desde o início da criação de metodologias para desenvolvimento de software, a primeira etapa de qualquer ciclo de vida escolhido era conhecida como levantamento dos requisitos do software. Atualmente reconhece-se que o levantamento de requisitos é uma atividade mais geral e que ocorre durante todo o ciclo de vida do sistema. Frente a isso, tem-se usado o termo "engenharia de requisitos" para identificar o tratamento dos requisitos do sistema durante todo o processo de desenvolvimento. A área de conhecimento de requisitos de software trata do levantamento, análise, especificação, validação e gerência de requisitos de software. Os requisitos de software são um dos produtos do processo de engenharia de requisitos, ao lado de requisitos de hardware e dos operacionais.

Um dos principais objetivos da engenharia de requisitos é descobrir como particionar o sistema e identificar quais requisitos devem ser alocados para quais componentes do sistema. Em alguns sistemas, todos os componentes são implementados no software, em outros existe uma mistura de tecnologias, hardware, software e operacionais. Por causa disso a engenharia de requisitos é mais propriamente considerada uma atividade de engenharia de sistemas do que da engenharia de software. Neste trabalho o foco são os requisitos alocados ao software.

Um dos dogmas fundamentais de uma boa engenharia de software é que haja uma boa comunicação entre os usuários do sistema e os desenvolvedores do sistema. É o engenheiro de requisitos que conduz esta comunicação. Ele precisa fazer a intermediação entre o domínio dos usuários do sistema e outras partes envolvidas [6] e o mundo técnico do engenheiro de software. Isto requer que ele possua experiência técnica, habilidade para conhecer e entender o domínio da aplicação e conhecimento sobre relacionamento interpessoal para conseguir estabelecer um consenso entre grupos heterogêneos das partes envolvidas do sistema.

4.1 Definições

Antes de se caracterizar a área propriamente dita serão introduzidos alguns conceitos e definições fundamentais.

[6] Partes envolvidas é a tradução adotada neste trabalho para *Stakeholders*.

4.1.1 Requisitos

De forma simples, um requisito é uma propriedade que precisa ser exibida para resolver algum problema do mundo real. Pensando-se em software, pode-se dizer que um requisito é uma propriedade que precisa existir num sistema desenvolvido e adaptado para resolver um problema em particular.

A gama de problemas é infinita, como por exemplo, automatizar parte de uma tarefa de um usuário do sistema ou suportar processos de negócio da organização ou controlar um determinado dispositivo, etc. As tarefas a serem automatizadas podem ser muito complexas e, portanto, os requisitos correspondentes também o são, podendo ser uma combinação complexa de requisitos de diferentes pessoas, de diferentes níveis dentro de uma organização e de diferentes contextos, nos quais o sistema precisa operar.

Os requisitos variam em termos das propriedades que representam e de seus objetivos. Pode-se fazer uma distinção entre os requisitos do produto e os requisitos do processo.

Os requisitos do produto podem ser classificados em:

- requisitos funcionais, são as funcionalidades que o sistema deve possuir, e
- requisitos não funcionais, são as restrições sobre a solução. Alguns exemplos deste tipo de requisitos são: requisitos de desempenho, requisitos de segurança, requisitos de manutenibilidade, requisitos de confiabilidade, etc.

Um requisito do processo é essencialmente uma restrição sobre o processo de desenvolvimento do sistema, como por exemplo: metodologia, linguagem ou banco de dados a serem usados.

Os requisitos precisam ser levantados com clareza e sem ambigüidade e, onde apropriado, quantitativamente. No entanto, o SWEBOK (2002) diz que pode ser necessária a existência de requisitos vagos e não verificáveis, dependentes de interpretação subjetiva, como por exemplo: “O sistema será confiável”, “O sistema será amigável”. Algumas dessas características estão associadas as questões de usabilidade e podem ser avaliadas através de testes de usabilidade. Exemplificando: “O sistema será confiável” está associado a baixa taxa de erros e possibilidade de recuperação em caso de falha, enquanto que “O sistema será amigável” refere-se a satisfação do usuário. Estes requisitos tem se tornado a cada dia mais importantes na medida em que se adote ciclos de vida centrados no usuário.

Uma propriedade essencial de todos os requisitos é que eles deveriam ser verificáveis. No entanto, pode ser difícil ou caro verificar certos requisitos.

Outra coisa muito importante é que cada requisito precisa ser unicamente identificado, para que possa haver o controle de configuração e a gerência do requisito durante todo o ciclo de vida do sistema.

Os requisitos do sistema englobam os requisitos dos usuários e requisitos de outras partes envolvidas, tais como: clientes, auditores, analistas de mercado, desenvolvedores do sistema, etc.

Além dessas fontes humanas, importantes requisitos de sistema freqüentemente derivam de outros dispositivos ou sistemas com os quais o software em desenvolvimento deverá interagir.

O levantamento e a análise dos requisitos devem ser feitos com a meta de alcançar todos os objetivos do projeto.

4.1.2 Análise de requisitos

A análise de requisitos é um termo muitas vezes usado de forma genérica para significar as atividades de captura, análise e validação dos requisitos do sistema. Uma vez que os objetivos do projeto tenham sido estabelecidos, este trabalho pode começar. Estas atividades são fundamentais para se ter clareza sobre o problema para o qual o sistema deverá dar uma solução e se estimar o custo da solução.

O analista de requisitos precisa estar seguro de que todas as fontes relevantes dos requisitos sejam identificadas e consultadas. No entanto, nem sempre isto será possível, uma vez que em sistemas de grande porte ou que sejam voltados para uma área de negócios (produtos de prateleira) o número de usuários pode ser excessivamente grande. Nestes casos, deve-se buscar consultar usuários representativos de cada classe das partes envolvidas do sistema, não esquecendo categorias específicas.

O levantamento dos requisitos não é uma tarefa fácil. O analista de requisitos tem que aprender um conjunto de técnicas e habilidades para ajudar as pessoas a relatarem como elas fazem seus trabalhos e ajudá-las a verificar se não existe uma forma melhor de faze-lo. Há muitas questões políticas e sociais que podem afetar as partes envolvidas na hora de relatarem seus requisitos, além de sua inabilidade ou falta de disposição de relatá-los. Neste caso, ele deve ser sensível a estes fatores.

Em muitos casos, é necessário criar todo um contexto para facilitar a conversa. Uma das possibilidades é apresentar às partes envolvidas um protótipo, mesmo que ele não tenha alta fidelidade em relação ao produto final. Levá-las através de um número pequeno de cenários, representando seqüências de eventos no domínio da aplicação, também pode ajudar a verificar os fatores chave que afetam os requisitos.

Uma vez que os requisitos do sistema tenham sido identificados, eles devem ser validados e negociados com as partes envolvidas, antes que os recursos sejam alocados ao projeto. Para possibilitar a validação, os requisitos do sistema devem ser mantidos em alto nível, usando-se termos do domínio do problema e não em termos técnicos. Muitos detalhes técnicos, neste estágio, obscurecem as características essenciais do sistema para seus clientes e usuários. Sugere-se o uso do protótipo como forma de comunicação entre as partes envolvidas por ser uma linguagem comum a todos.

Não é sempre que se pode satisfazer todos os requisitos. Alguns fatores que influenciam a sua não satisfação são: requisitos tecnicamente inviáveis, requisitos tão custosos para serem implementados que se resolve abandoná-los, requisitos mutuamente incompatíveis. O analista de requisitos precisa analisá-los para entender como eles interagem e suas implicações. Além disso, ele deve priorizá-los e estimar o custo de cada um deles. Com isso é possível identificar o escopo do sistema e um conjunto básico de requisitos que serão viáveis e aceitáveis. Provavelmente ele necessitará negociar com as partes envolvidas, cujos requisitos conflitam, para chegar a um escopo aceitável. Os requisitos do sistema precisam ser analisados no contexto de todas as restrições aplicáveis. Restrições vêm de muitas fontes, tais como: ambiente de negócios, estrutura organizacional da empresa e ambiente operacional do sistema. No caso de ciclo acelerado, deve-se ainda considerar a questão da janela de validade na priorização dos requisitos.

Para ajudar a análise dos requisitos do sistema, modelos conceituais são construídos. Isto ajuda no entendimento do particionamento lógico do sistema, seu contexto no ambiente operacional e os dados e comunicações de controle entre as entidades lógicas. A prototipação pode ser uma ferramenta eficaz para a realização desta tarefa.

O trabalho do analista de requisitos não está restrito à captura dos requisitos das partes envolvidas, mas inclui fazer a avaliação de sua viabilidade. Aqueles que são claramente inviáveis devem ser rejeitados e a razão registrada. Os que são meramente suspeitos de serem inviáveis são mais difíceis de serem avaliados. Um estudo de viabilidade pode ser justificado se, por exemplo, um requisito duvidoso é fortemente defendido por uma das partes envolvidas. Nos projetos de ciclo acelerado, requisitos duvidosos deveriam ser deixados para uma outra etapa.

Os recursos do projeto deveriam ser focados nos requisitos prioritários e mais importantes. Na prática, a quantidade de recursos disponíveis afeta fortemente a escolha daqueles que entrarão no escopo da solução. A existência de recursos suficientes permitirá que alguns requisitos não essenciais sejam satisfeitos, enquanto que recursos insuficientes forçarão a que requisitos fortemente defendidos sejam excluídos. A questão do tempo é hoje também um fator fundamental na escolha dos requisitos.

Espera-se que, ao final da análise seja obtido um conjunto completo e coerente de requisitos. Neste ponto, as principais áreas de atuação do sistema e as suas

funcionalidades devem ser conhecidas e estarem claras. Subsistemas ou componentes serão definidos para manipular cada uma das áreas principais e funcionalidades. Os requisitos do sistema serão então alocados ou distribuídos aos subsistemas e componentes.

A atividade de particionamento e alocação das funcionalidades do sistema é parte do projeto arquitetônico. O projeto arquitetônico é um trabalho que depende de muitos fatores, tais como, existência de padrões arquitetônicos ou de componentes de prateleira. O estabelecimento da arquitetura do sistema representa um marco no projeto e é crucial para o projeto que se obtenha a arquitetura correta. Em particular, a interação dos componentes do sistema afeta de forma fundamental o modo como o sistema exibirá as propriedades emergentes desejadas. Neste ponto, os requisitos e a arquitetura do sistema são documentados, revistos e considerados como uma "Linha Base" para o desenvolvimento, planejamento e estimativa de custo do projeto.

O refinamento dos requisitos e a arquitetura do sistema é onde o analista de requisitos interage com o projeto do software. Não há nenhuma fronteira clara mas é raro que a análise de requisitos continue abaixo de 2 ou 3 níveis de decomposição arquitetural antes que a responsabilidade seja passada para o time de projeto dos componentes individuais.

Propõe-se, neste trabalho, que a arquitetura do software, no que tange ao particionamento e alocação das funcionalidades, seja diretamente implementada no protótipo.

4.2 Divisão da área de requisitos de software

A divisão da área de conhecimento de engenharia de requisitos adotada neste trabalho é a do SWEBOK (2002), que conforme é citado pelos autores, é compatível com as seções do ISO/IEC 12207-1995[7] que se referem às atividades da engenharia de requisitos. Esta norma vê o processo de software em três níveis diferentes: primário, suporte e organizacional. Para simplificar a divisão, a estrutura foi definida num único ciclo de vida do processo. Não se pode, no entanto, esquecer que o processo é iterativo. Os tópicos identificados incluem as atividades primárias, tais como: levantamento dos requisitos e análise dos requisitos, juntamente com descrições específicas para gerência da engenharia de requisitos e num grau menor, processos organizacionais. Portanto, a validação e a gerência dos requisitos foram identificadas como tópicos separados.

[7] ISO/IEC 12207 - Information Technology - Software Life Cycle Processes

A estruturação adotada para a área foi baseada no processo. Esta estrutura foi escolhida pelo SWEBOK (2002), para refletir o fato de que a engenharia de requisitos precisa ser considerada como um processo para que tenha sucesso, com a complexidade e atividades firmemente acopladas (seqüenciais e concorrentes) inerentes a um processo, ao invés de ser considerada como uma atividade do início do projeto de desenvolvimento de software. Esta divisão compreende 6 (seis) tópicos, como mostrado na Figura 4.

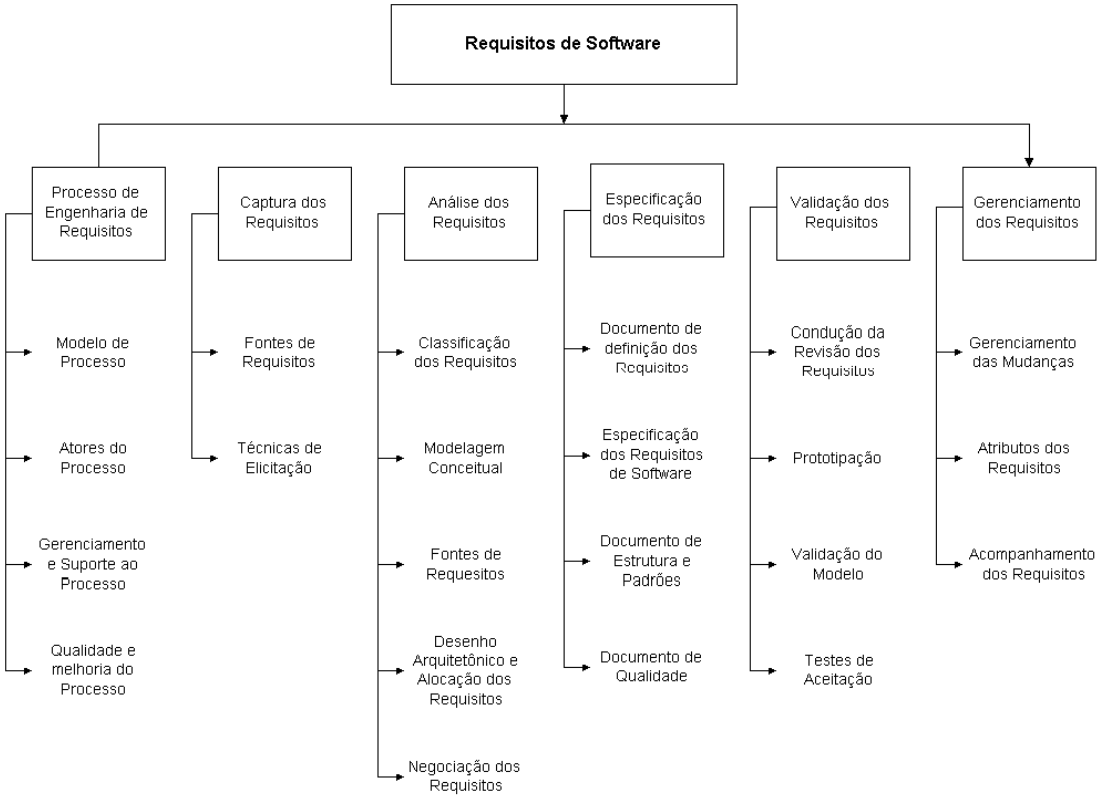

Figura 4. Estruturação da área de engenharia de requisitos, proposta no SWEBOK (2002)

A Figura 5 mostra, conceitualmente, como estas atividades se incluem num processo de engenharia de requisitos iterativo. As diferentes atividades são repetidas até que um documento aceitável de especificação dos requisitos seja produzido ou até que fatores externos, tais como, pressões por prazo ou falta de recursos causem o final do processo. Não se deve esquecer que o término prematuro do processo, normalmente, é prejudicial para a qualidade do projeto. Depois que o documento de requisitos do

sistema for considerado pronto, qualquer mudança fará parte do processo de gerência dos requisitos.

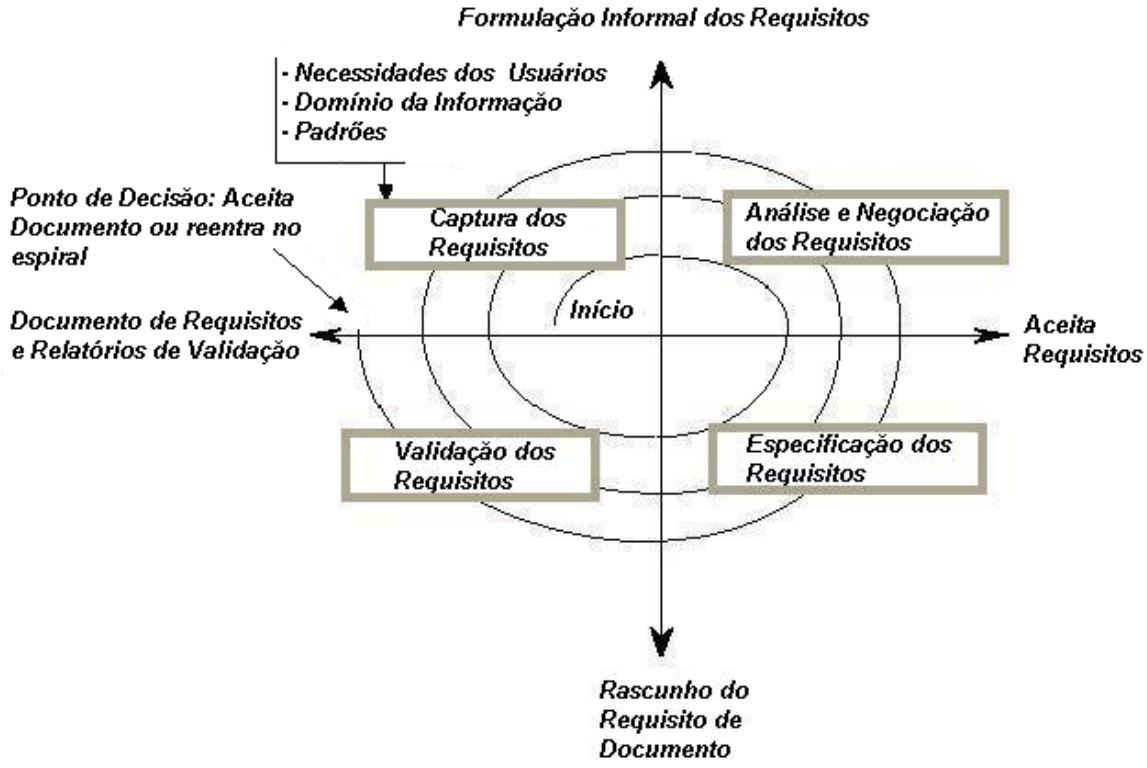

Figura 5. Atividades no processo de engenharia de requisitos, adaptada de SWEBOK (2002)

4.3 A prototipação dentro da área de requisitos de software

Tomando como base a divisão da área de requisitos de software apresentada no SWEBOK (2002) discutiremos apenas aquelas seções onde considera-se que a prototipação desempenha um papel importante.

4.3.1 Processo de Engenharia de Requisitos

A engenharia de requisitos é fundamentalmente interdisciplinar e o analista de requisitos precisa fazer a mediação entre o domínio do usuário e o da engenharia de software. Uma das atividades da engenharia de requisitos é a identificação dos **atores do processo**. Há freqüentemente muitas pessoas envolvidas com o engenheiro de requisitos, cada uma delas com um interesse diferente no sistema. O perfil das pessoas envolvidas poderá variar para diferentes projetos, sendo que usuários/operadores e os clientes (que não precisam ser os mesmos) sempre estarão incluídos. Os principais atores deste processo descritos na literatura são: usuários, gerentes, clientes, auditores, analistas de sistemas, projetistas de sistemas, programadores e pessoal operativo.

Uma condição básica para se ter sucesso no desenvolvimento de software é que todos os atores do processo sejam identificados, a natureza de seus interesses seja analisada e seus requisitos sejam levantados.

À primeira vista, parece que a identificação e conhecimento sobre os atores do processo não está relacionado com a prototipação. No entanto, são os usuários, gerentes, clientes e auditores do sistema que definirão as funcionalidades que o protótipo deve apresentar. O nível de conhecimento em informática dos usuários, sua experiência no uso de software e de protótipos, ditam as regras para o desenvolvimento das interfaces humano-computador. A identificação dos usuários representativos define o grupo de teste e validação do protótipo.

4.3.2 Captura dos requisitos

Trata-se aqui da origem dos requisitos (de onde eles vêm) e como eles podem ser coletados pelo analista de requisitos. A obtenção dos requisitos é o primeiro estágio na construção e entendimento do problema que o software deve resolver. Ela é fundamentalmente uma atividade humana e estabelece o relacionamento entre a equipe de desenvolvimento (usualmente coordenada pelo engenheiro de requisitos), o cliente e usuários.

Num sistema típico, haverá muitas **fontes de requisitos** (humanas e não humanas) e é essencial que se busque identificar todas as fontes potenciais e, além disso, seja feita a avaliação do seu impacto sobre o sistema.

Nem sempre é fácil identificar todas as fontes de requisitos inclusive porque, às vezes, só aparecem depois do software estar em produção. Um exemplo deste problema aconteceu na empresa da autora. Foi desenvolvido um software para controle de entrada e saída de visitantes. Depois que o software entrou em produção, percebeu-se que também poderia ser usado pelo sistema de qualidade da empresa já que os laboratórios que precisavam ter controle das suas visitas. Foi necessário levantar as necessidades desta nova fonte de requisitos e alterar o software de forma a contemplar os novos requisitos.

A captura dos requisitos é uma tarefa muito difícil e o analista de requisitos precisa estar sensibilizado para o fato de que os usuários podem ter dificuldade na descrição de suas tarefas, podem deixar informações importantes não especificadas ou podem ser relutantes ou inábeis para cooperarem. Mesmo se as partes envolvidas forem cooperativas e experientes, o analista de requisitos tem que trabalhar muito para levantar as informações corretas. Existem várias **técnicas de elicitação** para a realização desta atividade, tais como: entrevistas, encontros facilitados, observação, cenários e prototipação. No entanto, não se pode esquecer que os cenários e os protótipos podem ser usados nas entrevistas e nos encontros facilitados como base para as discussões. Na observação, caso já exista um sistema, ele pode ser considerado como um protótipo para o novo sistema.

A prototipação ajuda a descobrir os objetivos de alto nível do sistema, que são a motivação para o desenvolvimento do sistema, mas freqüentemente são vagamente formulados. Nesta fase pode-se usar vários tipos de protótipo para capturar os requisitos. Levantados os requisitos iniciais, protótipo exploratórios, desenhados em papel, podem ser uma fonte de inspiração para que os usuários se lembrem de novos requisitos. Outros softwares existentes e que fazem parte da tarefa do software proposto ou versões anteriores do produto também podem servir de protótipo para a captura dos requisitos.

O protótipo pode servir de linguagem de comunicação entre os desenvolvedores e os usuários.

4.3.3 Análise de requisitos

O processo de análise dos requisitos deve:

- detectar e resolver conflitos entre os requisitos;
- descobrir os limites do sistema e como ele precisa interagir com seu ambiente.

O SWEBOK (2002) inclui, sob o tema de análise, os seguintes assuntos: classificação dos requisitos, modelagem conceitual, fontes de requisitos, desenho arquitetônico e alocação dos requisitos e, ainda, negociação dos requisitos. Trataremos aqui dos assuntos, referentes a análise, onde o protótipo pode colaborar.

4.3.3.1 Modelagem conceitual

O desenvolvimento de modelos do problema é fundamental para a análise de requisitos, e seu propósito é ajudar no entendimento do problema antes de se iniciar o projeto da solução. Os modelos conceituais compreendem modelos das entidades do domínio do problema configurados para refletir suas relações e dependências com o mundo real.

O protótipo pode ser uma forma de modelar o problema, mostrando aos usuários, de forma clara, como o software irá modelar as entidades do mundo real, suas relações e dependências. Além disso, providencia uma forma de comunicação muito mais clara do que os diagramas propostos, por exemplo, na análise essencial, que na maioria das vezes não são entendidos pelos usuários. Um usuário comum entende muito melhor os limites da aplicação vendo um protótipo do que um diagrama de contexto.

Portella (1994, p. 36) em seu trabalho corrobora com estas idéias e afirma que:

> "Em contraste com um analista de informações, o usuário médio não tem experiência, e às vezes até conhecimentos necessários, para lidar com as abstrações que desenvolvedores de sistemas manuseiam. O enfoque tecnicista dessas representações gráficas, as torna de grande auxílio para os desenvolvedores, porém uma barreira quase intransponível para o usuário médio, não afeito a abstrações e à metodologia que embasa a leitura dessas representações.
>
> Além do que, parece-nos claramente impossível comunicar-se claramente o caráter dinâmico de um sistema de informações, por meio de uma descrição estática, qualquer que seja ela. Sistemas interativos pedem uma especificação interativa."

4.3.3.2 Desenho arquitetônico e alocação dos requisitos

O projeto da arquitetura é o ponto no qual a engenharia de requisitos se sobrepõe com o projeto do sistema ou software e mostra como é impossível separar claramente as duas tarefas. Em muitos casos, o engenheiro de requisitos age como o arquiteto do sistema, uma vez que o processo de análise e elaboração dos requisitos requer que os subsistemas e componentes que serão responsáveis pela satisfação dos requisitos sejam identificados e que se faça a alocação dos requisitos (indicação de responsabilidade para a satisfação dos requisitos aos subsistemas).

A alocação é importante para permitir a análise detalhada dos requisitos. Uma vez que um conjunto de requisitos tenha sido alocado a um componente, eles podem ser melhor analisados para se descobrir como o componente precisa interagir com os outros componentes para satisfazer os requisitos alocados. Em grandes projetos, a alocação estimula uma nova rodada de análise para cada subsistema.

4.3.3.3 Negociação de requisitos

A resolução de conflitos está relacionada com a solução de problemas de conflito entre os requisitos, como por exemplo: entre duas partes envolvidas que exigem características mutuamente incompatíveis; entre requisitos e recursos; entre capacidades e restrições. O engenheiro de requisitos não deve tomar uma decisão unilateral, sendo melhor consultar as partes envolvidas para conseguir um consenso sobre uma solução apropriada. É importante que tais decisões possam ser rastreadas pelo cliente. Esta atividade pode ser classificada como um tópico da análise de requisitos, porque problemas podem aparecer como resultado da análise, mas ela também pode ser considerada como parte da validação dos requisitos.

O protótipo ajuda na resolução de conflitos entre requisitos uma vez que sendo mais rápido construir o protótipo do que o software, é possível construir-se mais do que um protótipo mostrando a incompatibilidade de determinados requisitos, fazendo com que as partes envolvidas tenham que escolher entre uma implementação ou outra.

4.3.4 Especificação dos requisitos

O tópico especificação dos requisitos está relacionado com a estrutura, qualidade e veridicidade do documento de requisitos. O SWEBOK (2002) indica a criação de dois documentos ou um único com 2 partes. Cada um deles dirigidos a um tipo de leitor e com propósitos diferentes. São eles: o documento de definição dos requisitos e a especificação dos requisitos de software. Ele ainda considera que a documentação dos requisitos é a condição básica para o sucesso do gerenciamento dos requisitos.

4.3.4.1 Documento de definição dos requisitos do sistema

Este documento (também conhecido como documento dos requisitos dos usuários ou conceito das operações) registra os requisitos do sistema. Ele define os requisitos de alto nível do sistema na perspectiva do domínio. Seus leitores incluem representantes dos usuários e dos clientes do sistema, portanto ele precisa usar termos do domínio. Ele precisa listar os requisitos do sistema juntamente com informações práticas sobre os objetivos gerais do sistema, seus objetivos no ambiente e uma declaração das restrições, suposições e requisitos não funcionais. Ele pode incluir modelos conceituais criados para ilustrar o contexto do sistema, cenários de uso, as principais entidades do domínio, dados, informações e fluxos de trabalho.

4.3.4.2 Especificação dos requisitos de software

Os benefícios da especificação dos requisitos de software incluem:

- estabelecer a base para o entendimento entre os clientes e os contratados ou fornecedores sobre o que o produto de software deve fazer e o que não deve fazer. Para leitores não técnicos ela deve ser acompanhada pelo documento de definição dos requisitos;
- forçar uma rigorosa avaliação dos requisitos antes que o projeto comece, reduzindo assim retrabalho posterior;
- providenciar uma base realista para a estimativa dos custos do produto, riscos e cronograma;
- estabelecer a base para o desenvolvimento do plano de validação e verificação de forma mais produtiva;
- providenciar uma base de informações para a transferência do produto de software para novos usuários e novas máquinas.
- providenciar uma base para futuras extensões do escopo do software.

Precisa-se tomar cuidado na elaboração do documento de requisitos pois por melhor que ele seja é um texto difícil de ser lido e muito mais de ser mantido atualizado. Yourdon (1990, p. 153), citando De Marco (1978), Gane e Sarson (1977) e Weinberg (1978), já dizia que os documentos de requisitos (na época chamados de especificação funcional) padeciam de alguns grandes problemas:

- Documentos monolíticos – dificuldade em se entender uma parte da especificação sem ler o todo;
- Documentos redundantes – contendo a mesma informação repetida várias vezes, dificultando sua manutenção e entendimento;
- Ambigüidade – gerando interpretações diferentes feitas por leitores diferentes;
- Dificuldade de manutenção – dificuldade em se fazer o registro das mudanças.

Hoje já existe toda uma padronização para a especificação dos requisitos mas o que se observa, na prática, é que quando se desenvolve um sistema novo todos os cuidados para a elaboração do documento de especificação dos requisitos são tomados. No entanto, para softwares que já estão em produção há algum tempo e que sofreram manutenção, o documento de requisitos está totalmente desatualizado. Este é o grande problema das equipes que devem dar manutenção nos sistemas legados. Um exemplo

disso pode ser observado na crise ocorrida na questão do "*bug* do milênio" onde muitos sistemas tiveram que ser refeitos por falta de documentação.

Um outro problema é que, geralmente, analistas de sistemas têm dificuldades para escrever, e quando escrevem usam uma linguagem tão técnica que o usuário tem dificuldade para entender. Este foi um dos motivos que levaram à adoção dos diversos tipos de diagrama utilizados na Engenharia de Software.

Sugere-se neste trabalho que o protótipo pode ser usado como a especificação dos requisitos de software, principalmente para projetos de pequeno e médio porte com ciclo acelerado, pois proporciona os benefícios apontados acima. A discussão sobre este assunto é apresentada nas seções 5.4.5.1 e 5.5 deste trabalho.

4.3.5 Validação dos requisitos

É normal que haja formalmente um ou mais pontos no cronograma do processo de engenharia de requisitos onde os requisitos são validados. O objetivo é descobrir quaisquer problemas antes que recursos sejam usados para implementar o requisito. A validação dos requisitos está relacionada com o processo de exame dos requisitos para assegurar que estão definidos corretamente.

A **prototipação** é considerada no SWEBOK (2002) uma das atividades da validação dos requisitos. Ela é empregada, normalmente, para a validação da interpretação feita pelo analista de requisitos sobre os requisitos do sistema, bem como para o levantamento de novos requisitos. Existe um conjunto de técnicas de prototipação e vários pontos no processo onde a validação do protótipo pode ser apropriada. A vantagem dos protótipos é que eles podem tornar mais fácil analisar as suposições do analista de requisitos e dar um retorno muito útil sobre quais estão erradas. Por exemplo, a ação dinâmica de uma interface com o usuário pode ser melhor compreendida através de um protótipo animado do que através de uma descrição textual ou de modelos gráficos. Há também desvantagens que incluem:

- perigo de que a atenção dos usuários seja distraída por características cosméticas, ao invés de se concentrar no entendimento das funcionalidades;
- problemas com a qualidade do protótipo.

Segundo o SWEBOK (2002), por esta razão, muitas pessoas recomendam protótipos que evitam software, tais como simuladores baseados em *flip-chart*. Protótipos podem ser caros e demorados para se desenvolver; entretanto, como eles evitam a perda de recursos causada pela tentativa de satisfazer requisitos errados, seus custos podem ser mais facilmente justificados.

Na prática, no entanto, a elaboração de protótipos não tem se mostrado demorada e cara. Técnicas como criação de *templates* de tela e reuso de código agilizam o processo

e criam consistência no desenvolvimento do protótipo. Não se descarta no entanto o uso de protótipos exploratórios, principalmente na fase de levantamento dos requisitos.

Uma outra atividade da validação dos requisitos são os **testes de aceitação**. Uma propriedade essencial de um requisito do sistema é que deveria ser possível verificar se o produto final irá satisfazer o requisito. Os requisitos que não podem ser validados são realmente apenas “desejos”. Uma tarefa importante é, portanto, o planejamento de como verificar cada requisito. Na maioria dos casos, isto é feito no projeto dos testes de aceitação. A identificação e o projeto dos testes de aceitação pode ser difícil para os requisitos não funcionais. Para serem validados eles precisam primeiro ser analisados para se descobrir como eles podem ser expressos quantitativamente.

4.3.6 Gerenciamento dos requisitos

A gerência dos requisitos é uma atividade que percorre o ciclo de vida do software inteiro. Ela está relacionada fundamentalmente com a gerência das mudanças sobre os requisitos e sua manutenção, de forma a refletir acuradamente o que o software executa e para o que foi criado.

4.3.6.1 Gerência de mudanças

A gerência de mudanças é a parte central da gerência de requisitos. Este sub-tópico descreve as regras da gerência das mudanças, os procedimentos que precisam ser realizados e a análise que deveria ser aplicada para as mudanças propostas. Ela tem ligações fortes com a área de conhecimento de gerência de configuração.

Não se pode esquecer que sendo o protótipo um artefato do processo de desenvolvimento de software, deve estar sujeito ao mesmos procedimentos definidos para os outros artefatos. O protótipo é importante na gerência de mudanças pois documenta os requisitos alocados em cada uma das suas versões, podendo, portanto, ser usado para comparar e verificar as mudanças ocorridas nos requisitos durante o ciclo de vida do software. Pode-se também, através das diversas versões do protótipo analisar a evolução dos requisitos do software.

4.3.6.2 Acompanhamento dos requisitos

O rastreamento dos requisitos está relacionado com a recuperação das fontes dos requisitos e a previsão dos efeitos das mudanças. O rastreamento é fundamental para a realização da análise de impacto quando existe uma mudança do requisito. A partir de um requisito, deveria ser possível saber-se que requisitos derivam dele ou são sua origem e, também, qual a parte envolvida que o motivou e, ainda, quais são os requisitos de software e entidades projetadas para o satisfazer.

Como os requisitos estão implementados no protótipo é necessário que se possa rastreá-los em todas as suas versões, para isso é necessário que se armazene de forma organizada as diversas versões, bem como, os documentos que definem a mudança proposta.

Finalizando, é importante também salientar que o reconhecimento de que, por diversos motivos, os requisitos do software mudam durante todo o ciclo de vida do produto, levou a criação da área de Engenharia de Requisitos. Da forma como aqui foi exposto, parece que a Engenharia de Requisitos é uma seqüência linear de atividades. Deve-se ressaltar que muitas destas atividades ocorrem de forma concomitante e que o processo global é iterativo. Os mesmos conceitos de iteratividade, evolução do produto e de que os requisitos mudam constantemente permeiam o processo de desenvolvimento de software adotado pela DITEL que é descrito no próximo capítulo.

Capítulo 5. A prototipação nos projetos da DITEL

Neste capítulo será feita a apresentação da prototipação no processo de desenvolvimento de software adotado pela DITEL (Divisão de Informática e Telecomunicações) do IPT, mais especificamente no AST (Agrupamento de Software e Telecomunicações), com o objetivo de demonstrar na prática a importância da prototipação e derivar, a partir dela, a principal contribuição do trabalho, que são as recomendações para a prototipação no processo de software, para projetos em ciclo acelerado.

Inicia-se este capítulo apresentando a caracterização dos projetos da área; depois será apresentado o processo de desenvolvimento de software que foi adotado pelo grupo, para a realização de suas atividades. A seguir será detalhado o uso da prototipação dentro do processo e para finalizar faz-se uma discussão sobre a proposta de uso dos protótipos como Especificação dos Requisitos de Software, em condições especiais.

5.1 Caracterização dos projetos da área

Em 1994 iniciou-se um projeto de parceria, entre o IPT e a Itautec, para o desenvolvimento de produtos para informática, tais como: URA (Unidade de Resposta Audível), terminais ATM, impressoras fiscais etc. Com o sucesso desse primeiro projeto, decidiu-se ampliar a parceria criando um grupo de desenvolvimento de software nas áreas de automação comercial, bancária e industrial. Para formar este grupo, foram convidados quatro profissionais de carreira do IPT e foram contratados 30 recém-formados. Integravam também a equipe, diversos técnicos da Itautec, conforme as necessidades dos vários projetos.

Com a ampliação destes projetos e o início de novas parcerias, em 1997 foi criado o Centro de Informática e Telecomunicações – CIT, que passou a divisão em 2001, assumindo o nome de DITEL. Um dos Agrupamentos da DITEL é o AST que tem como objetivo principal o desenvolvimento de software nas mais diversas áreas, e em especial, software voltado para a área de telecomunicações.

Desde a criação do CIT várias outras parcerias e projetos foram sendo assumidos pelo grupo, nas mais diversas áreas de negócio. Entre essas parcerias pode-se citar: Unisys Brasil, Sony, SABESP, Alcatel Telecomunicações, Gytech, Phillips, Sefaz, SCTDET, bem como outras áreas do próprio IPT, que requisitam os serviços de desenvolvimento de software.

Como pode-se ver pela carteira de clientes da DITEL, são muitas as áreas de negócio em que a divisão tem trabalhado.

Com o crescimento do volume de trabalho, percebeu-se a necessidade de que fosse definido um processo de desenvolvimento de software que fosse adotado pelas equipes nos diversos projetos, visando a melhoria da qualidade, refletida na uniformização da forma de trabalho, criação de padrões, e facilidade de treinamento de novos profissionais.

Embora os projetos abranjam várias áreas de negócio, eles, na sua maioria, possuem características comuns:

- São de médio ou pequeno porte;
- São desenvolvidos para ambiente Windows, para rede local ou ambiente WEB com muitas interfaces gráficas;
- Utilizam banco de dados relacional, sendo implementados em Oracle, MS SQL server ou Sybase;
- Possuem muita interação de dados e pouco processamento, ou seja, não necessitam de algoritmos complicados.

Como se pode perceber, os projetos têm características que os tornam ótimos candidatos ao uso da prototipação, já que são:

- fortemente interativos,
- de pequeno ou médio porte,
- possuem muitas interfaces e pouco processamento, fazendo com que o peso esteja na manipulação interativa das informações,
- necessidade de velocidade de desenvolvimento, sendo que se precisa ter pelo menos uma versão inicial num curto espaço de tempo, o que caracteriza o ciclo acelerado.

Foi criado então um primeiro processo que foi evoluindo no decorrer do tempo, conforme o grupo ia ganhando experiência. O processo atualmente adotado é descrito no próximo item.

Uma outra característica importante é a de que os projetos devem ser desenvolvidos numa janela de tempo muito curta. A cada dia mais o mercado exige que o software seja desenvolvido de forma iterativa, com novas versões sendo entregues em prazos cada vez mais curtos. Pode-se considerar que o software nunca está pronto. Eventualmente suspende-se o seu desenvolvimento numa determinada versão.

Muitos são os motivos que levam à necessidade deste tempo tão reduzido. Por exemplo:

- Data de inauguração, este é um caso ocorrido em 2002, onde foi necessário desenvolver o software de controle para o laboratório de controle de riscos de escorregamento da Serra do Mar. Neste caso o prazo estipulado foi de 3 meses pois a data de inauguração já estava agendada. Definiu-se então uma primeira versão da solução que caberia na janela estipulada. Após a inauguração foi definido um novo escopo para a segunda versão, deixando as demais funcionalidades para as próximas;

- Concorrência do mercado, busca-se sempre lançar um produto antes que o concorrente;

- Feiras e eventos, precisa-se de uma versão funcional para a data marcada;

- Exigência do cliente;

- Mudanças tecnológicas. A evolução tecnológica tanto de hardware como de software básico e ferramentas de desenvolvimento são tão rápidas que se houver muita demora no desenvolvimento do produto, ele ficará obsoleto antes de entrar em produção.

Com os problemas de prazo para o desenvolvimento e o desejo de manutenção da qualidade do produto foi necessário definir-se um processo de desenvolvimento de software a ser adotado nos diversos projetos. Embora o processo adotado pela equipe não seja o foco deste trabalho, optou-se por detalhá-lo para mostrar o papel relevante do protótipo neste ciclo de vida. Conforme Thompson e Wishbow (1992) lembram em seu trabalho:

> “Antes de tentar planejar um projeto de prototipação rápida é importante entender o processo de engenharia de software em uso em sua organização. “

O processo global atualmente definido é composto, basicamente de duas etapas mais gerais:

- Negociação da proposta
- Processo de desenvolvimento de software.

5.2 Negociação da Proposta

Existem atualmente duas formas empregadas de negociação de propostas. A mais comum é negociar-se um projeto global com o prazo, tamanho e perfil de equipe estabelecidos e com um escopo amplo e não bem definido. Neste primeiro momento tem-se uma lista de possíveis trabalhos a serem realizados. Depois, durante o decorrer do

projeto, faz-se um levantamento mais detalhado de cada trabalho a ser desenvolvido, definem-se prioridades e prazos e, de acordo com o tamanho da equipe, escolhem-se alguns destes trabalhos para serem realizados.

Uma segunda forma é definir-se o trabalho a ser realizado, e então fazer-se uma proposta técnica e comercial.

Nos dois casos, no entanto, é necessário que se negocie o projeto a ser realizado. Para tal são executadas as seguintes atividades:

- avaliação da complexidade geral do trabalho. Esta avaliação é bastante intuitiva e baseada no conhecimento adquirido no desenvolvimento de projetos anteriores;
- planejamento preliminar das tarefas, estimando o tempo gasto em cada etapa do processo de desenvolvimento do software;
- escolha das ferramentas e linguagens de programação a serem utilizadas;
- escolha do perfil da equipe que realizará o projeto;
- estimativa da quantidade de homens hora que cada membro da equipe gastará;
- elaboração de um macro cronograma;
- levantamento de outros recursos que serão necessários;
- no caso de se estar negociando a proposta individualmente, é necessário também fazer-se a previsão de custos.

Observa-se, deste processo de negociação, que em nenhuma hipótese o cliente contrata um projeto com os requisitos prévia e precisamente definidos, o que demonstra a importância da gerência de requisitos e dentro dela, da prototipação.

Esta etapa é realizada pelo gerente do projeto com a colaboração de alguns membros da equipe.

Cabe salientar que nesta etapa não há um detalhamento exaustivo do trabalho a ser realizado e, portanto, as medidas de complexidade não são feitas aqui.

5.3 Processo de Desenvolvimento de Software

Feita a negociação do projeto, parte-se então para o seu desenvolvimento propriamente dito. Cabe lembrar que o processo de desenvolvimento é evolutivo mas, por simplicidade, serão descritas as etapas de um único ciclo de vida.

O processo atualmente empregado pela equipe para o desenvolvimento de software baseia-se na análise estruturada e contempla as seguintes fases:

- Levantamento de Requisitos
- Análise e modelagem de dados
- Prototipação
- Validação
- Revisão e detalhamento do cronograma de desenvolvimento
- Desenvolvimento
- Testes integrados
- Implantação e testes de homologação
- Treinamento
- Documentação
- Acompanhamento
- Manutenção

Cada uma dessas fases é composta por um conjunto de atividades que serão detalhadas nos subitens seguintes. Deve-se também salientar que algumas destas atividades são executadas em paralelo para aumentar a velocidade de desenvolvimento.

Observe-se que embora a prototipação apareça explicitamente como uma fase, ela permeia o processo todo, como se vê a seguir.

A Figura 6 apresenta graficamente este processo.

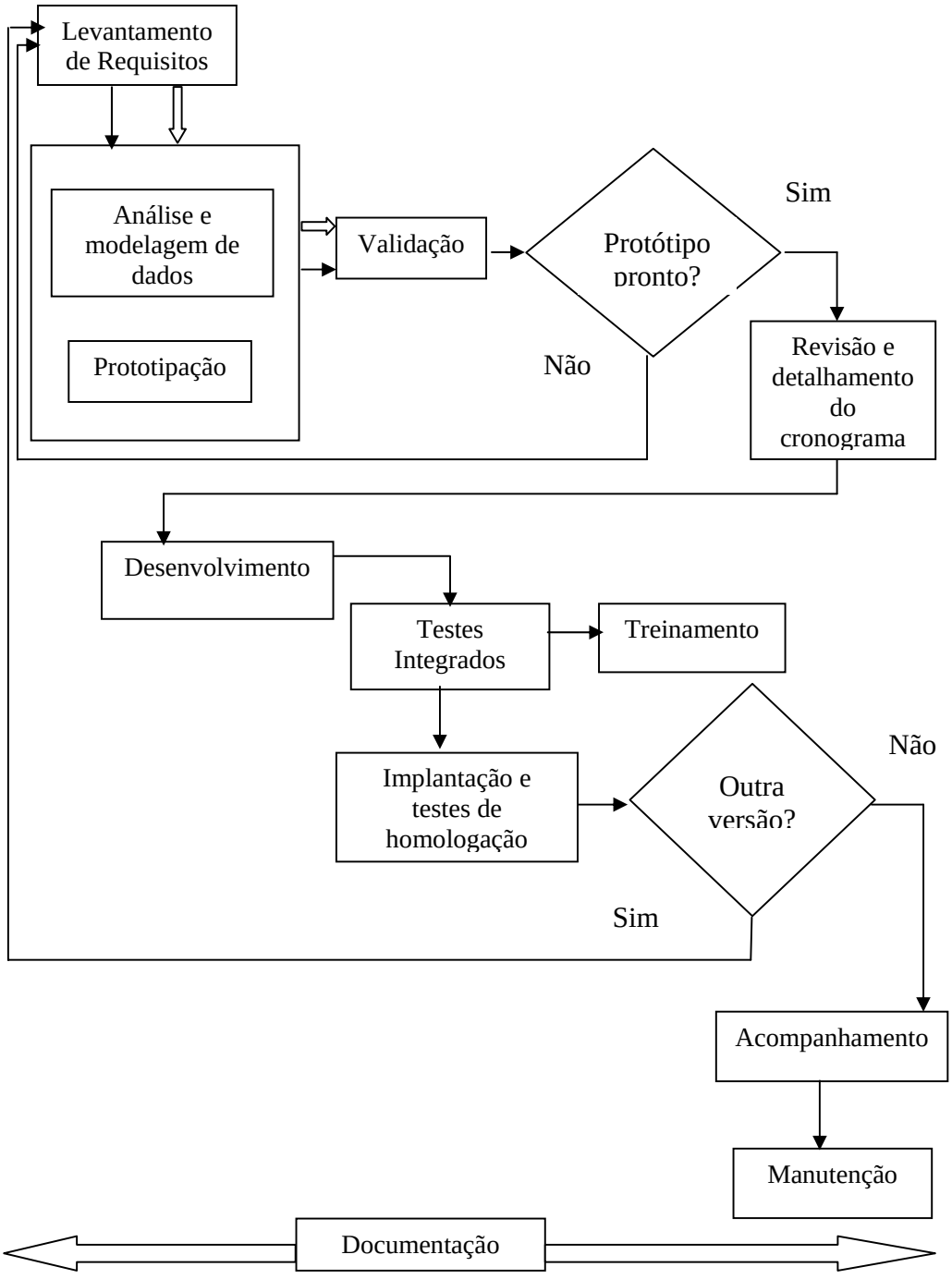

Figura 6. Processo de desenvolvimento de software adotado pela DITEL

5.3.1 Levantamento de Requisitos

Nesta fase é feito o levantamento das necessidades do cliente. Alguns clientes já sabem exatamente como é o processo que desejam automatizar mas, algumas vezes é necessário levantar-se inclusive o próprio processo de trabalho do cliente antes de se propor uma solução sistêmica. É nesta fase que se aprende como o cliente trabalha, as regras do seu negócio, suas dificuldades e os pontos críticos no seu processo. Esta fase pode ser rápida, se o cliente já sabe o que quer ou ser ela mesma é um subprojeto.

Para a realização desta fase é escolhida uma equipe de analistas sênior ou pleno, de acordo com a complexidade e tamanho do problema. Do lado do cliente é escolhida também a equipe que deve participar deste levantamento.

Esta fase é composta das seguintes subatividades: modelagem do processo de negócio e captura de requisitos, descritas a seguir.

Modelagem do Processo de Negócio

Para se conseguir definir bem um sistema, é preciso que se conheça o processo inteiro, mesmo que o sistema proposto automatize apenas uma parte deste processo.

Este levantamento é feito através de reuniões onde o cliente relata, de forma geral, como é realizado seu trabalho, quais são suas atividades e qual parte do processo precisa ser automatizada.

Procura-se também conhecer a empresa, como o sistema irá se inserir dentro da empresa, o relacionamento entre os diversos grupos envolvidos, outros sistemas já existentes na empresa e que provavelmente terão que ser integrados com o que será desenvolvido, se existem padrões corporativos de interface a serem seguidos, qual será o ambiente de operação do sistema, se existem ferramentas de desenvolvimento que deverão ser usadas, tais como: linguagem, banco de dados, gerador de relatórios etc.

Além das reuniões, procura-se visitar a empresa, observar “o clima”, o relacionamento entre os diversos grupos, o ambiente operacional disponível e criar relações amigáveis entre a equipe de desenvolvimento e os futuros usuários. Busca-se também identificar o nível de conhecimento em informática dos futuros usuários do sistema, pois este fator fornece parâmetros para a elaboração do protótipo. Procura-se definir, também, o perfil ou os próprios usuários que irão avaliar o protótipo.

Estas reuniões geram atas e fluxogramas operacionais, se for o caso, que descrevem o processo global. Esta documentação é validada pelo cliente e mantida como uma base para todo o desenvolvimento.

Captura dos requisitos

O entendimento inicial do problema é feito através de reuniões entre o cliente/usuário e parte da equipe de desenvolvimento onde é discutido o processo que deve ser automatizado, quais são suas etapas, como cada etapa funciona e quais são os grupos envolvidos em cada etapa e o que o usuário espera do sistema.

Além das reuniões formais, sempre que possível, procura-se fazer uma análise de tarefas contextual, conforme citado por Mayhew (1999), através da observação do usuários no seu ambiente de trabalho, acompanhando suas atividades para a realização de suas tarefas diárias.

Procura-se então detalhar, para cada etapa do processo a ser automatizado:

- as atividades de cada grupo de usuários;
- entradas, saídas, resultados e documentos a serem gerados;
- algoritmos para a obtenção dos resultados;
- a integração com outros sistemas;
- gargalos existentes;
- necessidade de aprovação.

Como produto das reuniões de levantamento e da observação são elaboradas atas que servem de base para um documento de requisitos que descreve o problema, o processo em linhas gerais e os requisitos da solução desejada. O processo atual é discutido e, eventualmente, alterado.

Nesta etapa, usa-se intensamente a prototipação exploratória e cenários, onde as principais telas são desenhadas no papel, durante a reunião ou antes dela, para servir de base para as discussões. Vendo o esboço das telas os usuários entendem aonde se deseja chegar, opinam sobre as funcionalidades que o sistema irá implementar e lembram de características que haviam esquecido.

O produto final desta fase é então um documento contendo a lista dos requisitos do software a ser desenvolvido e o protótipo exploratório desenvolvido nesta fase. Dependendo da complexidade do processo pode-se ainda usar fluxogramas para elucidar o processo ou outro tipo de desenho que facilite o entendimento entre o cliente e a equipe de desenvolvimento.

Nesta etapa cria-se um novo projeto no *Source Safe*[8] para armazenamento, compartilhamento e controle de versão de todos os artefatos que serão gerados pelo projeto.

O critério de finalização desta etapa é a aprovação do cliente para os documentos acima descritos. Esta aceitação normalmente é dada por escrito ou verbalmente numa reunião ou através de e-mail. Dificilmente tem-se um documento assinado pelas partes.

5.3.2 Análise e modelagem de dados

Na etapa de análise estudam-se os artefatos obtidos na etapa anterior: lista de requisitos, protótipos exploratórios em papel e fluxogramas, quando houver. Faz-se, então, o particionamento das funcionalidades, definem-se os módulos e atribuem-se as funcionalidades aos módulos. Inicia-se em paralelo a modelagem de dados.

É importante ressaltar que como os dados vêm dos objetos conceituais do usuário, e os sistemas desenvolvidos são altamente interativos, a etapa de prototipação descrita no próximo sub-item é realizada em paralelo com esta etapa. Tem-se optado, para a modelagem dos dados, pelo modelo entidade-relacionamento, pois é o modelo conhecido e usado pelos clientes atuais.

É gerado então o documento com o modelo de dados e apresentado ao cliente. Normalmente, usa-se a ferramenta ERwin[9] para desenhar o modelo pois esta ferramenta já gera um dicionário de dados e os *scripts* para a criação do banco de dados e facilita na documentação do modelo.

Na fase de modelagem também é definida a arquitetura do sistema, e o gerenciador de banco de dados que será usado. Esta fase ocorre em paralelo com a fase 3, pois mudanças no protótipo podem gerar mudanças no modelo. A apresentação do protótipo facilita o entendimento dos documentos e dos dados.

A equipe escolhida para esta etapa é normalmente composta por analistas sênior e pleno e, para os projetos típicos da organização, não é muito grande, de um a três analistas, a não ser que o sistema exija uma equipe maior.

---

[8] *Source safe* é uma ferramenta da marca da MicroSoft para armazenamento e controle de versão dos artefatos do projeto.

[9] ERwin – ferramenta da marca da Logic Works para *design* de modelos de dados relacionais e dimensionais.

5.3.3 Prototipação

Nesta fase é alocada uma equipe para a elaboração de um protótipo horizontal, conforme definido na seção 2.4.3, contendo todas as telas que comporão o sistema final. Estas telas apresentam as funcionalidades que o sistema deverá contemplar, como funcionará e como será a navegação entre telas. Como na etapa de levantamento já foram esboçadas em papel as principais interfaces, fica mais fácil esta implementação.

Inicia-se o protótipo pela definição do padrão de telas que será usado. O protótipo é desenvolvido na linguagem que será usada para o desenvolvimento do produto e procura-se utilizar *templates* de telas já definidos e programados. Esse reuso de código aumenta a velocidade do desenvolvimento do protótipo e proporciona consistência para o produto ou família de produtos em desenvolvimento. Quando o cliente possui um padrão de telas próprio da empresa, e que é diferente dos adotados para o grupo, cria-se um novo padrão gerando *templates* próprios para esse cliente, aumentando assim o repositório de *templates* definidos pelo grupo.

A existência de padrões de tela préprogramados, aliada ao uso de protótipos exploratórios em papel, gerados durante o levantamento de requisitos e que fornecem uma idéia bastante razoável de como o sistema deve funcionar, contribui muito para a diminuição do tempo necessário para a realização da atividade de prototipação.

O tempo médio despendido na elaboração de um primeiro protótipo é proporcional ao tamanho total do projeto. Na experiência profissional da DITEL, a prototipação não leva muito tempo. Dependendo do tamanho do projeto, tem-se despendido entre 80 HH a, no máximo, 360 HH para se ter um primeiro protótipo horizontal, já com o modelo de dados definido e a análise feita, ou seja, definidos os principais processos e com a atribuição de suas funcionalidades.

A equipe que participa desta fase é normalmente a mesma que participou das etapas anteriores, além de alguns analistas/programadores mais jovens.

O critério de término da fase é obter-se um protótipo a ser apresentado ao cliente.

5.3.4 Validação

Quando o sistema é de maior porte, procura-se usar cenários ou desenvolver e validar o protótipo por partes para garantir que a solução que está sendo adotada está no caminho certo, evitando assim que se descubra que existem erros graves somente depois dele pronto.

Terminado o protótipo, prepara-se uma apresentação com o protótipo funcionando (somente navegação entre telas) para que o cliente tenha uma idéia do funcionamento do sistema. É gerado, também, um documento impresso com as telas para auxiliar nas

discussões com os usuários. Para complementar o protótipo, se houver alguma funcionalidade que exija um algoritmo mais difícil, este algoritmo, normalmente, é detalhado, mas não é implementado.

O documento com as telas e o protótipo são apresentados ao cliente e entra-se num processo de discussão sobre o sistema. Eventualmente, instala-se o protótipo e estipula-se um prazo para que o cliente o analise. Vendo como o sistema trabalhará ele tem idéias e sugestões que são analisadas e incorporadas ao sistema, ou podem ficar para serem implementadas numa outra versão do produto. Quando as mudanças propostas pelos usuários são grandes e influenciam fortemente a solução proposta, tem-se que rever a análise e o modelo de dados do sistema. Por isso é que as fases 2 e 3 correm em paralelo.

O protótipo ajuda na exploração e consolidação do modelo de dados uma vez que podemos, através dele, verificar os dados, seus atributos e como estão relacionados dentro do sistema. Além disso, pode-se descobrir novos requisitos nesta fase, fazendo com que o documento de requisitos tenha que ser revisto.

Esta fase exige uma série de reuniões e modificações no protótipo e no modelo de dados. Não existe um número fixo de reuniões para se avaliar o protótipo mas, na experiência adquirida pela equipe, não é um processo muito demorado. Este número de ciclos depende:

a) Do ambiente da aplicação;

b) Do número e qualidade do grupo envolvido no processo;

c) Das limitações de prazo e restrições de recursos.

Normalmente, em duas ou três reuniões consegue-se chegar a um protótipo aceito pelo cliente. Segundo Bahn e Nauman (1997), não existe um número fixo de ciclos de crítica e revisão.

O processo de validação e alteração do protótipo, em geral, não leva mais que uma semana e, portanto, num curto espaço de tempo já se tem um protótipo funcionando e validado. Como o protótipo é desenvolvido na linguagem que será usada para o produto, ganha-se tempo na programação.

A equipe que participa desta fase é normalmente a mesma que participou das etapas anteriores, além de alguns usuários representativos.

O critério de término da fase é obter-se um protótipo aceito pelo cliente.

5.3.5 Revisão e detalhamento do cronograma de desenvolvimento

Estando concluída a prototipação, pode-se avaliar mais criteriosamente o trabalho das próximas etapas, já que se tem todas as telas definidas e pode-se avaliar o nível de complexidade de cada uma delas. O nível de complexidade, atualmente, é avaliado tomando-se por base experiências anteriores, dificuldade de implementação das funcionalidades pedidas, exigência de recursos não conhecidos, etc.

Inicia-se, então, o planejamento das próximas etapas do desenvolvimento do sistema. Todos os módulos e telas são listados e atribui-se um prazo previsto para a programação de cada uma delas. Acrescentam-se as tarefas de teste, preparação do treinamento, treinamento dos usuários e homologação, e seus prazos. Na definição dos prazos previstos, usa-se a experiência de desenvolvimento anteriores para fazer a previsão do tempo necessário para a realização das tarefas. Faz-se, então, a alocação da equipe às tarefas, levando em conta o paralelismo de atividades. Se o cronograma obtido está condizente com a previsão inicial não há problemas. Caso contrário, é necessário que se aumente o tamanho da equipe e se renegocie com o cliente, o prazo e os custos do projeto.

Esta não é uma etapa comum dentro do ciclo de vida de projeto definido pela Engenharia de Software; no entanto, é uma prática recomendada na gerência de projetos, evitando que se tenha surpresas numa fase posterior do projeto.

Muitas vezes o cliente deseja uma versão do sistema num prazo menor do que aquele previsto. Define-se, então, um particionamento da solução de forma a ter-se uma versão inicial que conterá as principais funcionalidades exigidas do sistema e deixa-se para uma segunda fase as funcionalidades não tão prementes. Aqui também o protótipo é útil, aumentando a visibilidade da negociação. Quando o sistema é muito grande pode-se dividir o desenvolvimento do sistema em várias etapas, de forma a que sejam desenvolvidas soluções incrementais. Por exemplo, um sistema que deve gerar diversos conjuntos independentes de relatórios, pode ser implementado liberando versões que automatizem um conjunto de relatórios por vez, ou ainda, para um sistema que automatize as atividades de diversos departamentos pode-se desenvolvê-lo por departamento, tendo assim uma solução incremental.

Normalmente, o gerente junto com a equipe que irá desenvolver o sistema, fazem uma lista com todas as tarefas depois discutem o prazo de execução de cada uma delas, chegando a um acordo interno. Esta prática é muito útil, na medida que faz com que a equipe assuma um compromisso com o andamento do projeto.

Este cronograma é enviado ao cliente, discutido e validado. Se o cliente pedir muitas alterações, a equipe revê os prazos e, eventualmente, aumenta-se o tamanho da equipe de forma a cumprir o prazo negociado.

5.3.6 Desenvolvimento

Inicia-se o desenvolvimento, criando todo o ambiente necessário para o desenvolvimento do software. Entre estas atividades pode-se citar: definição de padrões de documentação de código fonte, definição de nomenclatura para varáveis, criação da base de dados, definição de bibliotecas de caráter geral e definição e criação de base de dados para teste.

A seguir, para cada tela ou programa a ser desenvolvido, tem-se que detalhar suas funcionalidades, codificar, realizar os testes básicos e corrigir erros que ocorram nos testes básicos.

5.3.7 Testes integrados

Conforme cada módulo vai ficando pronto inicia-se os testes integrados dentro do módulo e depois, entre módulos. Os testes integrados normalmente são realizados por uma parte da equipe que não fez o desenvolvimento. Os erros encontrados são reportados pela equipe de teste através de um sistema desenvolvido pela própria equipe e que é acessado via Web por todos. Conforme a equipe de desenvolvimento corrige o erro, informa através do sistema que já corrigiu a causa e como se corrige este erro, para que sirva de aprendizado para a própria equipe. Com isto, está-se montando um banco de informações que tanto pode servir de base para não se cometer o mesmo erro novamente, como para avaliação do tempo de manutenção exigido para a correção de erros.

5.3.8 Implantação e testes de homologação

Na etapa de implantação e homologação do software são realizadas as seguintes atividades:

- escolha, por parte do cliente, dos usuários que irão validar o sistema. A escolha destes usuários deverá estar de acordo com o perfil definido na modelagem de negócio;
- definição de um prazo para os testes;
- criação, no cliente, do ambiente de testes e instalação do sistema com acesso para os usuários escolhidos para executar os testes;
- treinamento do grupo de usuários que realizará os testes;
- realização, pelos usuários, dos testes, sendo que os erros são relatados num documento padrão;

- avaliação dos erros encontrados para ver se realmente são erros ou não;
- corrigem-se os erros que comprometem o funcionamento do sistema e libera-se nova versão corrigida. Erros que não comprometam o funcionamento do sistema e sugestões de alteração, entram num cronograma de manutenção, conforme prioridades estabelecidas;
- controle de liberação de novas versões.

Quando o sistema é considerado estável faz-se a instalação em ambiente de produção e libera-se o acesso para a entrada do software em produção.

5.3.9 Treinamento

Antes do sistema entrar em produção é necessário que seja feito o treinamento dos usuários. O treinamento pode ser feito de várias formas, dependendo da quantidade de usuários a serem treinados, da qualificação dos usuários, do tempo disponível para o treinamento.

Normalmente, inicia-se o treinamento por uma apresentação geral do sistema para os usuários e gerentes envolvidos, para que todos possam ter uma visão global do sistema. Em seguida é feito o treinamento de cada grupo na realização das tarefas que lhes compete. Este treinamento pode consistir na administração de seminários ou cursos para ensinar o usuário a usar a nova ferramenta de trabalho ou, se o grupo for pequeno, pode-se treiná-los *on the job*.

Após o treinamento, o cliente normalmente quer que a equipe do projeto permaneça na empresa nos primeiros dias para acompanhar e ensinar os usuários.

5.3.10 Documentação

A etapa de documentação ocorre durante todo o ciclo de vida do projeto. Vários tipos de artefatos podem ser gerados. Pode-se citar:

- Atas de reunião incluindo fluxogramas de processo, protótipos exploratórios, outros desenhos e documentos obtidos no levantamento de requisitos;
- O documento de requisitos, contendo: lista dos requisitos, modelo de dados, dicionário de dados e DFDs e outros diagramas de definição das funções do sistema. No caso da DITEL os DFDs não são gerados;
- Descrição de algoritmos;

- Documentação do código fonte;
- Material de treinamento;
- Manual de operação;
- *Help online*.

A obtenção dos três últimos artefatos citados pode ser acelerada pelo uso do protótipo, uma vez que pode-se iniciar seu desenvolvimento usando como base o protótipo, em vez de ter que esperar o sistema estar pronto.

5.3.11 Acompanhamento

Durante a homologação, e possivelmente nos primeiros tempos após a entrada em produção do sistema, é realizado o acompanhamento. Esta etapa consiste em se manter uma parte da equipe no cliente para acompanhamento diário do uso do sistema, detecção de possíveis falhas, resolução das mesmas no local ou repasse dos problemas, para a equipe de desenvolvimento, para que sejam solucionadas.

Nesta etapa também é feita a avaliação do produto e são levantadas sugestões de melhoria que devem ser implementadas imediatamente ou na próxima versão do sistema.

São levantados: erros de execução, questões de desempenho não detectadas na fase de homologação, dificuldades no uso do sistema, melhorias e novas funcionalidades pedidas pelos usuários.

Paralelamente a esta fase, normalmente, é iniciado o desenvolvimento da próxima versão, de forma que esta etapa ou termina porque o sistema está estável e o cliente já se sente tranqüilo para usá-lo sem necessitar da presença da equipe ou volta-se a fase de levantamento de requisitos da próxima versão do sistema.

5.3.12 Manutenção

É combinado entre as partes um prazo para que o cliente efetue seus testes e relate erros dos sistema e melhorias desejadas. Dentro da atividade de manutenção, geralmente, são montadas três equipes: uma responsável pelo suporte, uma responsável pela implementação da manutenção e uma para efetuar os testes depois das correções. Além disso, uma pessoa é designada para controlar a liberação de novas versões do sistema, depois de efetuadas as correções.

5.4 Análise da prototipação no processo de desenvolvimento de software da DITEL

Esta seção é uma contribuição deste trabalho, na medida em que se faz uma análise da importância da prototipação no ciclo de vida adotado pela DITEL. Na apresentação do processo de desenvolvimento de software a prototipação aparece com uma etapa do processo. Na realidade, o protótipo tem um papel muito mais abrangente neste processo. O protótipo ajuda no levantamento dos requisitos, na análise, é usado na validação dos requisitos, ajuda no processo de definição do particionamento da solução para implementação iterativa. Como a forma de desenvolvimento é iterativa, seja pela implementação dos módulos em várias etapas, seja pela implementação de novas versões, no final do desenvolvimento, o sistema, ou a parte dele que foi desenvolvida, passa a ser um protótipo para a próxima versão.

Além disso, o protótipo é usado como o ponto de partida para a programação e permite que se inicie a documentação do usuário antes mesmo dele estar pronto. O protótipo ainda é usada como a especificação dos requisitos de software, nas condições especiais que se está tratando.

5.4.1 A prototipação no levantamento de requisitos

Como já foi dito, na descrição do processo, começa-se, como é natural, pelo levantamento de requisitos. Certamente, na primeira vez, não se consegue levantar exatamente como o sistema deverá funcionar. Vários fatores impedem que o levantamento seja exaustivo:

- Quase sempre, o próprio usuário não sabe exatamente o que quer;
- O usuário não tem muito tempo para se dedicar a esta tarefa;
- O usuário não sabe descrever com precisão suas tarefas, esquece detalhes, não lembra das exceções que podem ocorrer, e assim por diante;
- Como normalmente são sistemas novos, os usuários não tem noção de como ele poderá auxiliar no seu trabalho;
- Em alguns casos, os usuários não têm experiência em trabalhar de forma automatizada e aí tem-se mais dificuldades ainda, pois o usuário não consegue ajudar muito e se prende à forma como sempre tem trabalhado;
- Alguns usuários ficam amedrontados com a possibilidade de perder o emprego com a implantação do novo sistema e decidem colaborar o mínimo possível;
- A gerência sempre acha que se está levando muito tempo nesta atividade e fica pressionando por resultados.

Frente a esses problemas, optou-se por fazer o melhor levantamento inicial possível e então partir para as próximas etapas. Protótipos exploratórios são largamente utilizados nesta etapa para facilitar o entendimento entre as partes. Quando se consegue concluir um protótipo e ele é apresentado aos usuários, eles começam a perceber como o sistema funcionará, o que está certo e o que está errado, as coisas que esqueceram de contar e, como pede-se sua colaboração, normalmente eles se sentem como parte do processo e resolvem colaborar. Portanto o protótipo pode ser considerado também uma ferramenta de levantamento de requisitos.

Além disso, requisitos não funcionais considerados superficiais, do tipo cor de tela, posição e tamanho de botões e menus, posição dos campos, invariavelmente não são levantados inicialmente, mesmo porque é diferente se falar sobre uma cor do que ver seu efeito na tela de um sistema. Quando o protótipo é apresentado, o usuário é capaz de criticar as escolhas feitas pela equipe. Isto é muito importante pois deve-se lembrar que o usuário poderá ter que trabalhar várias horas por dia utilizando o sistema e se ele não for agradável o usuário poderá ter impacto em seu desempenho.

A análise, o projeto de arquitetura e a modelagem de dados são realizadas ao mesmo tempo em paralelo com a prototipação. Nesta etapa busca-se analisar os requisitos do sistema procurando:

- Identificar os agentes externos que irão interagir com o sistema, sejam pessoas, outros sistemas ou bancos de dados (diagrama de contexto);
- Identificar os principais processos do sistema, a interação entre eles e o fluxo de informações entre os processos (DFD nível 0);
- Elaborar a modelagem de dados;
- Elaborar a especificação dos algoritmos para os quais isto for necessário;
- Definir os módulos do sistema e associar os requisitos que comporão cada módulo.

Como os sistemas são relativamente pequenos ou podem ser divididos em sub-sistemas menores, não é necessário nenhum detalhamento de nível inferior. O protótipo é elaborado paralelamente, dispensando um detalhamento maior dos processos. Não é feito um documento com a especificação dos requisitos pois o protótipo é a própria especificação. Os requisitos são alocados diretamente no protótipo não sendo necessário que se faça um documento especificando a arquitetura do sistema.

Os diagramas de navegação, que representam a parte dinâmica do sistema, não precisam ser desenvolvidos, já que o protótipo, em si, mostra toda a navegação do sistema. Se o cliente o exigir, os DFDs detalhados e os diagramas de navegação podem ser desenvolvidos *a posteriori*. No entanto, isto até agora nunca foi necessário.

O protótipo também ajuda na modelagem dos dados já que na verificação do protótipo também são detectados problemas no modelo, falta de atributos e erros nas características dos atributos das entidades. No entanto, o protótipo não elimina a necessidade da documentação do modelo de dados.

5.4.2 A prototipação na validação dos requisitos

O protótipo é uma ferramenta muito importante na validação dos requisitos. Esta sempre foi considerada como sua principal função dentro do processo de desenvolvimento de software. Como já foi caracterizado anteriormente, o tipo de software desenvolvido pela equipe favorece o uso do protótipo para a validação dos requisitos. O ciclo de validação dos requisitos é realizado da seguinte forma:

1. Apresenta-se o protótipo numa reunião geral para a qual são convidados gerentes e usuários, cujo objetivo é a apresentação dos aspectos globais da aplicação. Para que as pessoas possam acompanhar é distribuído um documento com a cópia das telas que serão discutidas.
2. Após, ou durante, esta apresentação os gerentes e usuários apresentam suas críticas. Elas são debatidas, pois muitas vezes aparecem sugestões conflitantes. É feita então uma relação das mudanças sugeridas. Para cada uma dessas sugestões é decidido se a mudança deve ser implementada imediatamente ou se será feita numa nova versão.
3. Se as alterações são muito grandes e alteram de forma importante a solução proposta, o protótipo é refeito e volta-se a etapa 1 (um) deste ciclo.
4. Se as alterações são de pequena monta, são realizadas e passa-se à etapa de validação detalhada com os usuários.
5. Nesta etapa valida-se, módulo por módulo, diretamente com o grupo de usuários que usarão o sistema. Se o grupo for muito grande, faz-se primeiramente uma apresentação com o maior número de usuários possível, ou com aqueles mais representativos. Nesta apresentação procura-se detalhar e discutir todas as características que serão implementadas, os resultados de cada característica e documentos que serão gerados. Novamente escreve-se um documento com todas as sugestões de alteração, os conflitos são resolvidos e são atribuídas prioridades a cada sugestão, dentro de uma certa classificação, como por exemplo: fundamentais, importantes e desejáveis.
6. Se as alterações são muito grandes e alteram de forma importante a solução proposta, o protótipo é refeito e volta-se a etapa 5 (cinco) deste ciclo.
7. Se as alterações são de pequena monta, são realizadas, considera-se o protótipo como homologado, salva-se esta versão, na ferramenta de controle de versão para

que não se percam as decisões tomadas e passa-se à próxima etapa do processo de desenvolvimento do software.

5.4.3 A prototipação no cronograma de desenvolvimento

A prototipação é de fundamental importância no planejamento e definição do cronograma de desenvolvimento do produto. A partir do protótipo pode-se obter a lista de todas as telas que serão desenvolvidas, avaliar a complexidade de cada uma delas, estabelecer os prazos previstos como necessários para a programação de cada uma delas, a possibilidade de paralelismo e a obrigatoriedade de sequenciamento entre tarefas. A partir da complexidade do protótipo obtido pode-se também prever o prazo e a equipe necessária para os testes do produto e o tempo necessário para o treinamento dos usuários e a homologação do produto.

A partir destas informações pode-se definir a equipe necessária e estabelecer um cronograma de forma a cumprir os prazos combinados dentro dos recursos previstos. Se for impossível cumprir o prazo combinado inicialmente com o cliente dentro dos recursos previstos, deve-se renegociar com o cliente o prazo e os recursos ou a definição de uma solução com escopo menor, de forma a se ter uma versão no prazo previsto, deixando para uma segunda versão as funcionalidades menos prementes.

5.4.4 A prototipação na programação

Com o advento dos ambientes visuais de desenvolvimento, os protótipos já são desenvolvidos na linguagem final. Além disso, como é usada a prototipação horizontal, todas as características do software já estão criadas de forma superficial, portanto, o aproveitamento do protótipo é quase total. Com isso, alia-se a consistência gerada pelo protótipo com as vantagens de se poder aproveitar o protótipo. O que se precisa fazer na etapa de programação é aprofundar as características do software. Como as interfaces e a navegação já estão programadas tem-se então que programar, basicamente, os mecanismos de segurança, os algoritmos, acesso a banco de dados, relatórios e consultas. Com isso tem-se economia no tempo de programação, pois parte do trabalho foi feito na etapa anterior.

5.4.5 A prototipação na documentação dos requisitos

A prototipação na documentação de requisitos é importante porque há uma expansão do uso tradicional do protótipo. Segundo Rudd e Isensee (1994, p. 37):

> “Quando o protótipo representa acuradamente o que os usuários estão olhando, ele providencia uma especificação para os desenvolvedores

> codificarem. O estilo antigo de especificação funcional tipicamente contém centenas de páginas de detalhes técnicos obtusos. Um protótipo providencia um "especificação viva" que o revisor pode ver, tocar e brincar. A prototipação pode aumentar grandemente a eficiência do processo de desenvolvimento pela eliminação de revisões de especificações obtusas que somente uma pequena porcentagem dos revisores lêem e entendem. Se você não está apto a começar a se livrar das especificações escritas, a um mínimo, o protótipo suplementa a palavra escrita."

A necessidade da documentação de requisitos pode ser vista por dois ângulos: o ângulo do usuário e o do desenvolvedor. O usuário precisa da documentação dos requisitos para ter certeza de que suas necessidades foram entendidas e atendidas. Por outro lado o desenvolvedor precisa da documentação dos requisitos para:

- ter certeza que atenderá os requisitos (o que e como deve ser feito);
- ter certeza de que o software estará de acordo com as necessidades do usuário, não só as objetivas (requisitos funcionais e não funcionais) como as subjetivas (visual das interfaces, padrões, satisfação dos usuários...);
- Quando for necessário dar manutenção, saber o que foi definido e como mudar e quais os impactos da mudança.

O conjunto ideal de documentos é aquele que já foi apresentado em 5.3.10. No entanto, no desenvolvimento de software com ciclo acelerado pode-se substituir, ou dispensar temporariamente, alguns desses documentos. No caso de sistemas descartáveis pode-se inclusive dispensar "*ad eternum*" alguns desses documentos uma vez que o sistema será jogado fora depois de um tempo e não há necessidade de preocupação com a manutenção do mesmo. As atividades de documentação e manutenção da documentação atualizada têm forte impacto no ciclo acelerado, podendo torná-lo impraticável.

No caso de ciclo acelerado a interface/ protótipo substitui alguns dos documentos exigidos.

A modelagem dinâmica do sistema pode ser substituída pelo protótipo uma vez que o protótipo representa a própria dinâmica do sistema ou seja, representa o sistema operando seus casos de uso.

O protótipo também pode substituir a definição detalhada de processos padronizados agrupados pela regra de negócio. Operações de inclusão, alteração e exclusão não precisam ser detalhadas uma vez que se tem o modelo de dados e que a interface apresenta todos os dados necessários para a operação. Através do uso de *combos*, para campos que têm um conjunto pré-definido de valores, a interface pode apresentar apenas os valores possíveis para serem escolhidos, dispensando assim a documentação do domínio destes campos. No caso de alteração, pode-se proteger os campos que não poderão ser alterados.

No entanto, o protótipo não substitui o modelo de dados nem a especificação de algoritmos mais sofisticados.

Sugere-se aqui, também, que o protótipo pode ser usado como a especificação dos requisitos de software, principalmente para projetos de pequeno e médio porte com ciclo acelerado, pois proporciona todos os benefícios apontados acima.

O protótipo, desde que usado o protótipo horizontal, retrata tudo o que o produto de software fará. Com isso ele é capaz de corresponder a todos os benefícios que uma especificação de requisitos de software deve ter, conforme já visto em 4.3.4, ou seja, estabelece uma regra de entendimento entre contratados e contratantes, permite a avaliação rigorosa antes do início do desenvolvimento, estabelece uma base para o plano de validação e testes, permite estimativas de prazos e custos mais realistas e serve como base para futuros desenvolvimentos

5.4.5.1 O uso do protótipo para especificação de requisitos na DITEL

Para se entender como a prototipação permite a especificação e a gerência dos requisitos de software, é necessário entender claramente como e onde são gerados os diversos tipos de protótipo dentro do ciclo de vida do software e como eles são gerenciados.

Para facilitar o entendimento, apresentamos na Figura 7, novamente, o ciclo de vida adotado pela DITEL, acrescido do fluxo de criação dos protótipos através do processo.

Neste diagrama as setas largas indicam o sentido do fluxo dos diversos tipos de protótipo e a legenda abaixo apresenta o significado dos símbolos empregados.

Sigla	**Significado**
Temp	*Template*
TAtu	*Template* atualizado, retorna ao repositório
PExp	Protótipo exploratório
Prot	Protótipo
PAlt	Protótipo com sugestão de alteração
SW	Software pronto

Para facilitar o entendimento vamos exemplificar através do acompanhamento deste fluxo para um projeto de um novo software.

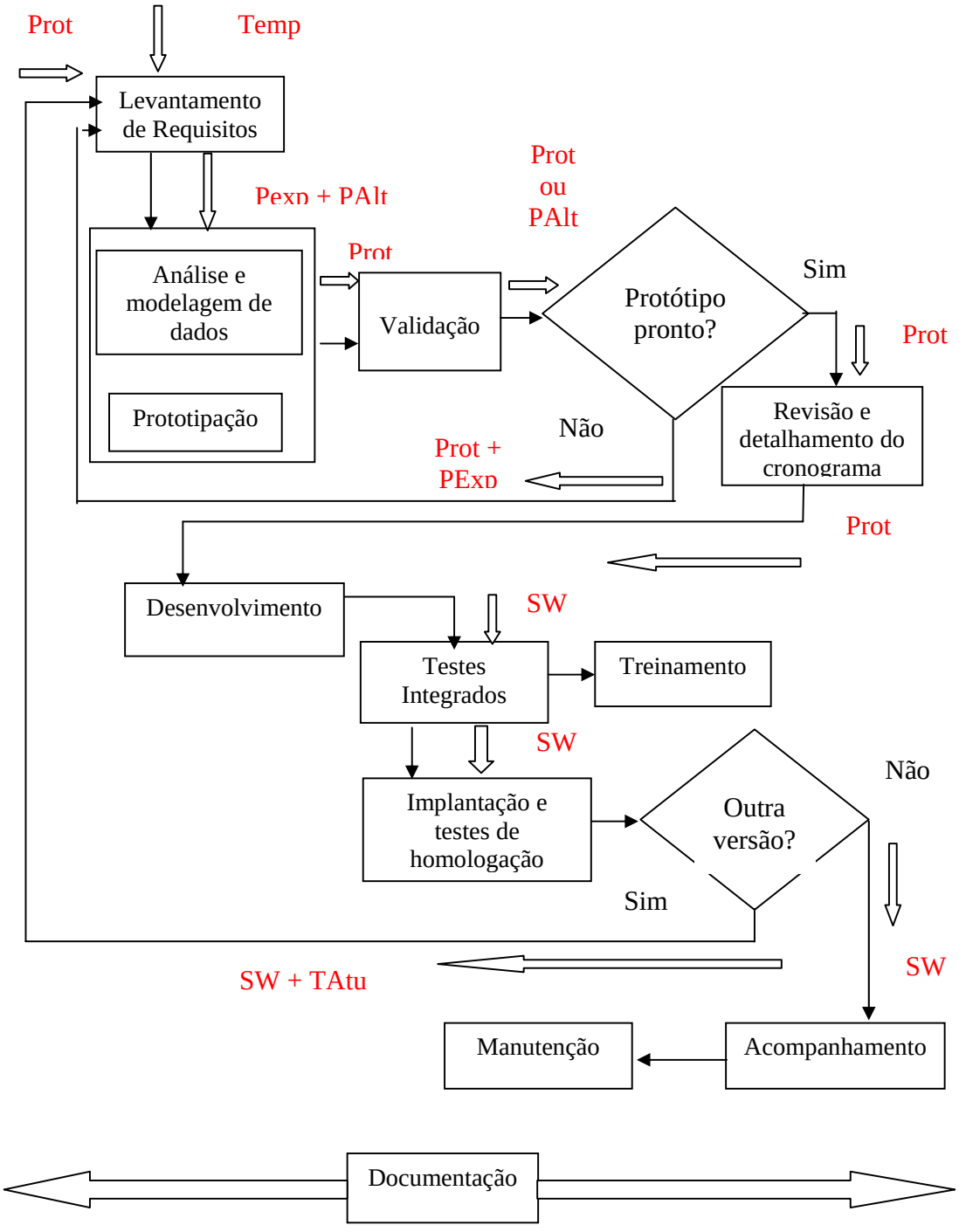

Figura 7. Processo de desenvolvimento de software da DITEL, com o fluxo dos protótipos

Imagine que se está iniciando um novo projeto de software e que toda a infra-estrutura necessária já foi criada pela empresa, já que ela está preparada para o desenvolvimento em ciclo acelerado.

Inicia-se então o levantamento preliminar. Neste momento não se tem nenhum protótipo, apenas *templates* de telas que podem ser escolhidos pelo cliente. Durante esta fase do levantamento são escolhidos os ***templates* de tela que serão adotados** e são **gerados protótipos exploratório**s. Os documentos adicionais gerados aqui são as **atas de reunião** e, se necessário, um **fluxograma das principais etapas do processo**.

Passa-se então a etapa de geração do protótipo. Nessa etapa ocorrem em paralelo a análise dos requisitos, projeto de arquitetura, modelagem de dados e programação do protótipo. Ao final desta etapa temos um **protótipo** horizontal que reflete os requisitos especificados. Como documentos adicionais são geradas as **especificações de algoritmos** mais complicados, se houver, e **atas de reuniões**, se ocorrerem, **o modelo de dados** e um **dicionário de dados** gerado pela ferramenta de modelagem de dados.

Inicia-se então a validação dos requisitos. O protótipo é apresentado aos usuários para experimentação. Na verdade, a sessão de validação é também uma sessão de levantamento de requisitos pois a medida que o usuário experimenta o protótipo ele lembra de novos requisitos ou altera requisitos já definidos. Para documentar as alterações pedidas no protótipo ou se anota ou desenha nas cópias das telas. Os documentos gerados nesta etapa são os **protótipos exploratórios** (cópia das telas do protótipo com as anotações sobre as mudanças a serem feitas) e **atas de reunião**. Entra-se então num processo iterativo de validação e geração do protótipo até que todos os requisitos tenham sido levantados ou se decida que quaisquer outros requisitos ficarão para uma próxima versão. Neste momento tem-se um protótipo validado e pronto para ser implementado.

O protótipo serve então de base para um estabelecimento do cronograma de desenvolvimento e revisão do escopo caso não seja possível cumprir a janela de validade. No caso de alteração do escopo deve-se registar em ata e rever no protótipo eliminando funcionalidades ou simplesmente bloqueando-as até uma próxima versão.

Passa-se então a etapa de desenvolvimento. Os programadores recebem o protótipo e têm acesso a todos os documentos gerados nas etapas anteriores. Cabe lembrar que para a equipe de programação um protótipo é uma especificação muito mais clara do que um modelo oriundo da análise. Com o protótipo em mãos eles sabem exatamente o que devem fazer e não perdem tempo tentando saber o que o analista deseja. Este é um fator de aceleração do processo. Como resultado desta etapa tem-se o **software pronto**. Precisa-se acostumar a equipe a **documentar o código fonte**.

A próxima etapa são os testes que são compostos pelos testes integrados, implantação no cliente e homologação. Nesta etapa é importante que os testes sejam exaustivos mas devem se concentrar nas funcionalidades implementadas. Como o

protótipo foi validado pelo cliente, é muito difícil que se tenha esquecido qualquer funcionalidade que deveria fazer parte desta solução. No caso do usuário desejar mais alguma funcionalidade ela deve ser anotada para a próxima versão do produto, ou seja, não há mudança de escopo nesta fase. Isto não causa problemas em relação ao cliente pois o processo é evolutivo e com ciclo acelerado, o que significa que uma nova versão estará se iniciando em breve. Erros encontrados devem ser registrados, bem como sua solução. Ao final desta etapa tem-se o **software pronto** e um **registro das imperfeições ocorridas**. Esse registro serve de aprendizado para a equipe de desenvolvimento. Paralelamente a etapa de desenvolvimento e a de testes é gerado o **manual do usuário** ou ***help online*** tendo como base o protótipo que foi homologado.

Como o processo é evolutivo, volta-se a etapa de levantamento para uma nova versão. Neste caso existem duas possibilidades:

1. A evolução vai ocorrer pela implementação de novas funcionalidades no software atual. Neste caso, o produto desenvolvido passa a ser o protótipo para a nova versão;

2. A evolução vai ocorrer pela implementação de novos subsistemas. Neste caso, os *templates* escolhidos para a versão anterior são adotados, para manter a consistência do produto.

Como pode ser visto pela descrição acima, o protótipo sempre representa a especificação dos requisitos do software que está sendo desenvolvido, no momento atual, eliminando assim o antigo problema das especificações que ficam desatualizadas.

Num certo momento, o cliente decide que o produto atende todas as suas necessidades e portanto está pronto O ciclo é então interrompido. Se depois ele pedir uma manutenção que não seja a correção de erros, a manutenção pedida faz com que o software volte ao início do processo evolutivo, tendo como protótipo inicial a mesma escolha descrita acima para uma nova versão do software.

Para garantir o gerenciamento e a rastreabilidade dos requisitos usa-se o *Source Safe* como descrito nas recomendações, no Capítulo 6 deste trabalho. O único problema na utilização desta ferramenta é que depende-se do fator humano para registrar que alteração foi feita e porque, por isso, é necessário que a equipe seja treinada e se comprometa com a qualidade do projeto.

Como vimos, no processo da DITEL, o protótipo é capaz de trazer todos os benefícios da ERS.

Como estamos trabalhando em ciclo acelerado, os requisitos são alterados ao longo de todo o ciclo de vida do produto, conforme o proposto na Engenharia de Requisitos, mas depois de validado o protótipo só é alterado na próxima versão.

A manutenção do software em produção ocorre para resolução de imperfeições, nunca para inclusão de novas funcionalidades. Este caso é tratado como um novo desenvolvimento, diminuindo assim os riscos de se perder o controle e do protótipo ficar desatualizado.

Não se está afirmando neste trabalho que a melhor solução para o desenvolvimento de software em ciclo acelerado, seja a adotada pela DITEL mas apenas que ela tem apresentado resultados satisfatórios. O último desafio da equipe foi desenvolver uma pequena aplicação contendo: 1 programa que executa uma aplicação externa e importa os dados resultantes da execução da aplicação, 3 telas, 2 relatórios e 2 tipos de gráfico, tendo um algoritmo razoavelmente complicado para cálculo de previsão de vazão em tubulações a partir de dados de campo (dados importados e armazenados num banco de dados), com o prazo de 1 semana de prazo para estar instalado e funcionando no cliente.

5.5 O uso dos protótipos para a documentação dos requisitos de software

A importância da prototipação no desenvolvimento de software em ciclo acelerado praticado na DITEL leva a uma discussão importante sobre a viabilidade de se substituir a especificação de requisitos de software pelo protótipo.

A razão desta discussão está no fato de que, na prática da DITEL ela efetivamente ocorre, com sucesso.

Observe-se que a prototipação de que fala este trabalho é a da IHC.

Deseja-se portanto, identificar nesta discussão os seguintes aspectos:

1) **Qual a vantagem para a aceleração do ciclo se a especificação de requisitos deixar de ser feita?**

   Para se trabalhar com ciclo acelerado é necessário simplificar ao máximo as atividades a serem realizadas, sem comprometer o projeto. A elaboração do documento de especificação dos requisitos é uma tarefa árdua e que demanda tempo. Como cita Correia (2003, p. 36) “Deve-se estudar a utilização de ferramentas que agilizam, não apenas na atividade de programação, mas também, nos testes dos programas, na documentação, no controle de versões, na captura de requisitos e na diagramação de modelos conceituais e físicos (classes, banco de dados).”

   Como o protótipo já está sendo desenvolvido para a validação dos requisitos seu aproveitamento para especificação dos requisitos acelera o desenvolvimento.

2) **Em que tipos de sistema e em que condições de projeto o protótipo pode substituir com vantagens a ERS?**

   Esta proposta de uso ampliado está restrita ao desenvolvimento de sistemas de informação (muita interface humano-computador), para sistemas de pequeno e médio porte com ciclo de vida acelerado, onde o tempo é fator primordial e a documentação deve ficar restrita ao mínimo possível, sem perda de qualidade. Para projetos de grande porte, se puderem ser reduzidos a sub-sistemas que atendam às condições acima, pode-se usar o mesmo raciocínio para cada sub-sistema.

3) **Qual o conteúdo da ERS que pode ficar explícito no protótipo? Todo? Parte?**

   No protótipo ficam explícitos os requisitos funcionais e os de usabilidade. Não estarão contemplados os requisitos de desempenho, de comunicação e outros que só poderão ser avaliados depois do sistema pronto.

4) **Esta substituição é definitiva ou temporária?**

   Para software "descartável" a substituição é definitiva, já que ele não estará sujeito a manutenção.

   Para software não descartável e de médio porte, sugere-se que quando o projeto for "paralisado" faça-se o documento de especificação de requisitos.

   Para software não descartável e de pequeno porte, a substituição pode ser definitiva, já que, para projetos deste porte, mesmo que o projeto termine e depois seja necessária uma manutenção é muito fácil resgatar os requisitos a partir do próprio protótipo.

5) **Em que condições a substituição <u>não</u> deve ser feita?**

   Não se aconselha a substituição no caso de sistemas grandes, com muitos algoritmos e pouca interface ou sistemas críticos, sem restrições de prazo, e equipes grandes. Nestes casos, aconselha-se a usar uma abordagem tradicional.

Para que o protótipo possa ser usado como a especificação do software ele precisa atender a dois quesitos básicos: especificar os requisitos que o software deve implementar e permitir a gerência dos requisitos. Tratemos então do primeiro quesito.

No quesito "especificar os requisitos que o software deve implementar", resgatando as discussões de capítulos anteriores, uma especificação dos requisitos de software deve:

1) **forçar uma rigorosa avaliação dos requisitos antes que o projeto comece**

   A validação dos requisitos é a principal função do protótipo. Para sistemas de informação, ele tem se mostrado uma ferramenta essencial na validação dos requisitos, inclusive defendida como tal em toda a literatura de Engenharia de Usabilidade e Engenharia de Requisitos. No protótipo pode-se validar inclusive os requisitos de usabilidade, que não poderiam ser validados nesta etapa se fossem usados documentos textuais. O protótipo permite, então a validação dos requisitos funcionais e de usabilidade. Outros requisitos que não possam ser validados no protótipo e o modelo de dados devem ser documentados e validados em separado.

2) **estabelecer a base para o entendimento entre os clientes e os contratados sobre o que o software deve fazer**

   O protótipo atende a esta necessidade pois é uma linguagem clara e fácil de ser entendida por clientes e desenvolvedores. Tendo-se um protótipo validado pelo cliente fica muito mais fácil garantir que não haverá problemas futuros do que se usarmos um documento textual que é sujeito a interpretações.

3) **providenciar uma base realista para a estimativa do custo do produto, riscos e cronograma**

   A partir do protótipo têm-se todas as funcionalidades que serão implementadas, e portando, é possível se fazer previsões bastante realistas sobre os prazos, custos e riscos, tendo como métrica o número de interfaces e a complexidade de cada uma delas.

4) **estabelecer a base para o desenvolvimento do plano de validação e verificação**

   Os cenários implementados no protótipo formam um conjunto de casos de uso para a validação do sistema. Verificação não é possível. O plano de validação pode ser desenvolvido em paralelo, tomando-se como base o protótipo validado. No entanto, a empresa deve garantir que não ocorrerão mudanças no produto final que não estejam refletidas no protótipo.

5) **providenciar uma base de informações para transferência do software para novos usuários e novas máquinas**

   A partir do momento que se tenha um protótipo validado, já é possível iniciar o desenvolvimento do material de treinamento e manual do usuário, mesmo antes do produto estar pronto.

6) **providenciar base para mudanças de escopo.**

   Uma vez que o protótipo é reutilizável, pode se considerar que ele cresce em ambas as dimensões (horizontal e vertical) à medida que avança o desenvolvimento. O

protótipo serve de base para a incorporação de mudanças que se fizerem necessárias. A Engenharia de Usabilidade sugere que, para a ampliação do software, o produto pronto e avaliado em campo possa ser o protótipo para a próxima versão do produto.

Desta análise verificasse que o protótipo cumpre todos as condições necessárias para ser uma Especificação de Requisitos de Software, para requisitos funcionais e de usabilidade.

Quanto à questão da “gerência dos requisitos”, precisa-se garantir que o protótipo sempre reflita os requisitos atuais, que se possa avaliar o impacto que uma mudança de requisito ocasionará no sistema e sempre saber o que foi alterado e porque.

Certamente, o protótipo não é capaz de garantir sozinho o gerenciamento dos requisitos. A empresa precisa estar preparada para gerenciar os requisitos de seus softwares e a equipe precisa ser motivada a faze-lo. O protótipo é uma das ferramentas para se conseguir a gestão mas são necessários, também, procedimentos e ferramentas e principalmente decisão política. Analisa-se a seguir como o protótipo pode garantir a gerência dos requisitos:

1) refletir sempre os requisitos atuais

A visão de prototipação adotada neste trabalho é um enfoque evolucionário de desenvolvimento de software, com reutilização do protótipo em cada ciclo de vida, portanto o protótipo reflete os requisitos atuais.

2) Permitir a avaliação do impacto de mudanças nos requisitos

O protótipo é uma ferramenta adequada para esta avaliação. É fácil verificar num protótipo quais alterações terão que ser feitas no produto para garantir a mudança de um requisito e qual o impacto disso. Num documento textual a análise dos reflexos da mudança é muito mais difícil.

3) Gerenciar as alterações

Para que se consiga gerenciar as alterações, a empresa deve dispor de ferramentas ou procedimentos que garantam o gerenciamento, seja dos requisitos ou de outros artefatos do projeto. Pode-se usar as diversas versões do protótipo para controlar as mudanças ocorridas ao longo do tempo. Uma comparação entre uma versão e a subsequente mostra o que foi alterado. No entanto, somente guardar estas versões não é o suficiente. Precisa-se registrar também qual o requisito que deu originou a mudança (lembrar que a alteração de um requisito pode implicar na alteração de vários outros). É desejável também que se registre quem pediu a mudança e o motivo. Sugere-se, também, a adoção de ferramentas que auxiliem neste tarefa, para que haja um efetivo controle das versões do software.

Assim, nestas condições especiais, o protótipo pode ser usado como a Especificação dos Requisitos de Software pois atende a todas as necessidades de uma ERS e também às do gerenciamento e rastreabilidade dos requisitos.

No Capítulo 6 apresenta-se um roteiro de boas práticas para o desenvolvimento de projetos em ciclo acelerado. Nestas situações especiais o conjunto de recomendações é útil para acelerar o ciclo.

Capítulo 6. Roteiro para o desenvolvimento de software em ciclo acelerado utilizando protótipos

Neste capítulo apresenta-se um roteiro de boas práticas para a elaboração de protótipos. Este guia, núcleo da contribuição deste trabalho, foi baseado no estudo realizado sobre a literatura de Engenharia de Requisitos e de Engenharia de Usabilidade, submetido ao crivo da experiência profissional adquirida ao longo dos anos no desenvolvimento de projetos, obtendo práticas que têm se mostrado úteis e que produzem resultados satisfatórios. Este roteiro consolida estas práticas. Sugere-se sua aplicação nos projetos de desenvolvimento de software que compartilhem as características dos projetos da DITEL, ou seja:

- Sistemas fortemente interativos,
- Sistemas de pequeno ou médio porte,
- Sistemas com muitas interfaces e pouco processamento, com manipulação interativa das informações,
- Desenvolvimento em ciclo de vida acelerado, com pouco tempo para a liberação de uma versão inicial e indicando o uso de processo evolutivo de desenvolvimento.

Este trabalho é particularmente voltado para o desenvolvimento de software em ciclo acelerado. Como advoga Correia (2003) em seu trabalho, é necessário que a empresa desenvolvedora esteja preparada para este tipo de desenvolvimento, ou seja, além das atividades para o desenvolvimento do produto, a empresa deve ter realizado atividades antes do desenvolvimento propriamente dito e realizar atividades após a liberação do produto, de forma a que se produza o software dentro de sua janela de validade e se garanta sua qualidade.

Mantendo-se consistência com o trabalho citado, procurou-se dividir o roteiro de boas práticas em recomendações para as atividades anteriores ao desenvolvimento, durante o desenvolvimento e pós entrega do produto.

De forma geral, alguns fatores são fundamentais para a garantia do sucesso do desenvolvimento de software em ciclo acelerado:

- Planejamento e controle das atividades – em projetos de ciclo acelerado, precisa-se acompanhar muito fortemente a execução das atividades, de forma que se houver um desvio em relação aos prazos previstos, possam ser tomadas atitudes imediatas para corrigir o desvio. Por isso a gestão do projeto é muito importante;

- Conhecimento na área de negócio – o ciclo acelerado depende fortemente do conhecimento do negócio. Se não for possível este tipo de especialização deve-se poder adquirir este conhecimento, contando-se com especialistas da área, para auxiliar a equipe de desenvolvimento;

- Equipe treinada e qualificada – é condição básica, para que se consiga acelerar o processo, que a equipe conheça muito bem as ferramentas que serão usadas, o repositório de *templates* e o de partes disponíveis, tenha facilidade em aprender e se adaptar a novas tecnologias. A equipe também deve ser treinada e convencida a utilizar as ferramentas de gerenciamento de projeto que serão adotadas;

- Equipe experiente em levantamento de processos e de requisitos – entender o que o usuário quer e necessita, a partir de uma explicação nem sempre objetiva, requer experiência. Além disso, precisa-se ter o profissional com o perfil adequado. Nem sempre é o melhor desenvolvedor que é capaz de entender o negócio do cliente. O profissional também deve ter facilidade de se fazer entender, utilizando um vocabulário pertencente ao universo do cliente;

- Reuso de software – o reaproveitamento de partes de software é fator fundamental na economia de tempo de desenvolvimento.

Visando o uso da prototipação para acelerar o ciclo de desenvolvimento, apresenta-se um conjunto de boas práticas. Este conjunto amplia os fatores de sucesso anteriormente apresentados, para os casos de software em que o aspecto interativo é importante

6.1 Boas práticas anteriores ao início do processo

Para que se possa desenvolver em ciclo acelerado, a empresa deve estar preparada para a tarefa. Apresenta-se aqui algumas recomendações que são de caráter geral e que a empresa deveria adotar independentemente de qualquer projeto, outras devem ser realizadas antes do início de um projeto específico.

Recomendação 1. Defina um processo para o desenvolvimento de software

Defina um processo de desenvolvimento de software para a empresa. Mesmo que não seja o melhor processo é melhor ter um do que nenhum. Esta é uma recomendação defendida por Correia (2003), onde ele também apresenta algumas alternativas de processos, de acordo com as características do projeto em questão.

Escolha um processo que leve em conta as questões de usabilidade, pois, nos dias atuais, este é um fator fundamental para o sucesso de qualquer projeto de software. No Capítulo 3 deste trabalho foram apresentados três ciclos de vida que poderiam ser adotados: Engenharia de Usabilidade de Nielsen (1993), Mayhew (1999) e Ciclo de vida em estrela, mas é importante lembrar que para que se tenha aceleração no processo é necessário que ele seja bem conhecido pela equipe e já ter sido praticado. Somente o conhecimento e a experiência proporcionam a aceleração.

Conforme sugere Correia (2003, p. 33), deve-se simplificar ou excluir as atividades e artefatos intermediários que não sejam necessários. A geração de produtos que não agregam valor pode comprometer o prazo.

Recomendação 2. Divulgue o processo escolhido

Procure fazer com que toda a equipe de desenvolvimento conheça o processo escolhido. As atividades que devem ser executadas e os artefatos que devem ser produzidos devem ser bem conhecidos pela equipe. Quando a equipe sabe antecipadamente o que e como deve ser feito, o resultado é muito mais rápido. Esta é uma das ações propostas em Correia (2003, p. 33).

Modelos de documentos, tais como: modelo de ata de reunião, modelo de especificação de algoritmos, padrões para a documentação de código fonte, etc, devem ser criados e divulgados.

Recomendação 3. Treine a equipe nas principais ferramentas

A equipe deve ser treinada, antecipadamente, nas principais ferramentas de desenvolvimento. O investimento neste treinamento trará retorno na execução dos projetos que serão realizados, devido ao aumento de produtividade e qualidade do produto. Esta é outra ação proposta por Correia (2003, p. 37).

Aproveite os intervalos entre projetos e incentive a equipe a se preparar para novas tecnologias e avanços da Engenharia de Software, efeitos de interface, novos paradigmas e mesmo “modismos” que trazem o efeito de modernidade.

Recomendação 4. Crie ou treine a gerência em ferramentas de planejamento e controle de projetos

Uma gerência que tem conhecimento em ferramentas de planejamento e controle de projetos, certamente, terá mais facilidade para gerenciar a equipe. Produtos para

elaboração de cronogramas, como por exemplo o *MsProject* da marca da Microsoft, devem ser conhecidos tanto pela gerência como pela equipe.

Incentive também o treinamento em técnicas de gerenciamento, principalmente nas questões relativas ao relacionamento pessoal, seja dentro da equipe, seja com o cliente. Um gerente que tem dificuldade com o relacionamento interpessoal terá maior dificuldade no seu trabalho. Esta é uma recomendação baseada em idéias propostas no PMBOK (2000).

Recomendação 5. Adote uma estrutura para armazenamento dos artefatos do projeto

Defina uma estrutura de diretórios ou use uma ferramenta para armazenamento de todos os artefatos do projeto e suas versões. No caso da DITEL é utilizado *o Source Safe* que é uma ferramenta capaz de armazenar e controlar versões, permitindo o compartilhamento de todos os artefatos, inclusive controlando quem está atualizando o artefato no momento, e não permitindo que outra pessoa possa alterá-lo simultaneamente. Se não houver ferramenta disponível, criar pelo menos uma estrutura específica de diretórios num servidor e criar procedimentos operacionais para o controle de versões e compartilhamento dos artefatos.

O uso de ferramentas de controle certamente é muito mais seguro. No início, quando ainda não se utilizava o *Source Safe*, houveram diversos problemas de analistas inadvertidamente salvarem versões mais antigas sobre novas, destruindo o trabalho realizado pelo colega de equipe.

Antes de se iniciar o desenvolvimento de um novo projeto, deve-se criar toda a estrutura necessária para o controle de configuração dos artefatos, separando documentos, protótipos e programas. Não se pode esquecer a importância da gerência de configuração de todos os artefatos gerados, conforme recomendado por modelos de qualidade como o CMM, visto em Paulk (1993a, 1993b) e as normas técnicas de desenvolvimento de software.

A ferramenta sugerida permite que se crie um projeto e defina a equipe que pode ter acesso aos artefatos armazenados. Além disso, pode-se criar uma estrutura de armazenamento dos artefatos, separando-os por categoria. Quando um membro da equipe acessa um artefato para alteração, ele fica alocado para a pessoa e disponível somente para leitura para o resto da equipe. Depois de efetuada a alteração o artefato deve ser devolvido com o registro do motivo da alteração, data em que ocorreu e quem efetuou a alteração. Com isso é possível rastrear todas as mudanças ocorridas nos artefatos do projeto.

Recomendação 6. Prepare o ambiente de desenvolvimento

Antes de iniciar o projeto prepare o ambiente de desenvolvimento, instalando as ferramentas necessárias e disponibilizando toda a documentação para a equipe que irá trabalhar no projeto. Certifique-se de que toda a equipe está com o mesmo ambiente operacional e que estão usando a mesma versão das ferramentas. É necessário que haja uma cultura homogênea.

Ferramentas de caráter geral, tais como, gerenciadores de banco de dados, softwares para controle de versão dos artefatos, geradores de relatório, cronogramas, etc, devem ser instalados num servidor para acelerar o processo e garantir a integridade do ambiente.

Recomendação 7. Crie um ambiente para a realização dos testes similar ao dos usuários

Crie um ambiente de testes, tanto para o protótipo, como para o produto final, similar ao ambiente dos usuários. Essa é uma regra muito importante para que a equipe tenha uma avaliação de como o protótipo e o sistema final irão funcionar no cliente. Esta diretriz parece óbvia mas num projeto realizado pela equipe, tanto o protótipo como o software final foram sempre apresentados e validados num computador pertencente à DITEL. Quando se levou o software para homologação na empresa do cliente, descobriu-se que os equipamentos dos usuários eram muito inferiores aos usados pela equipe. Como conseqüência, o sistema, no cliente, apresentou um desempenho totalmente insatisfatório. Foi necessário fazer-se muitas modificações no sistema, incluindo nesta fase uma análise de performance num equipamento similar ao do cliente.

Recomendação 8. Escolha a equipe de projeto criteriosamente

Escolha a equipe responsável pelo projeto. Ela deve ser composta por membros sênior da equipe de desenvolvimento e por representantes do cliente ou dos usuários.

Embora seja recomendação geral, a equipe de analistas deve conhecer o repositório de *templates*, ter experiência e sensibilidade para identificar os principais processos e conhecer a seqüência para prototipação. Procure fazer com que todos se comprometam com o projeto.

Na prática, não é possível se trabalhar apenas com analistas sênior, até por questões de custo. Sugere-se a formação de equipes compostas por analistas de vários níveis, inclusive para que os mais novos aprendam com os mais experientes.

Procure formar pessoas que tenham facilidade para conhecer áreas de negócio diferentes e entender o que o cliente quer e precisa.

Recomendação 9. Crie uma identidade para seus produtos

Procure criar uma identidade para os projetos desenvolvidos pela equipe, ou seja, defina um padrão geral que será replicado para novos projetos. Mayhew (1999) sugere que se crie padrões documentados de objetos de tela e que a partir destes padrões sejam criados e documentados os padrões de projeto de tela.

Na DITEL optou-se por criar *templates* de projetos de tela, proporcionando o reaproveitamento de código fonte, projeto e documentação. Na Figura 8 e na Figura 9, onde são apresentadas respectivamente uma tela do Sistema Integrado de Bacias Hidrográficas (SIBH) e uma do Sistema Integrado de Hidrovias (SINHV), dois produtos de uma mesma família, pode-se ver o uso do efeito de identidade.

Existem outros *templates* que foram adotados em outros sistemas mas que também não fogem muito do padrão acima mostrado. Na Figura 10 e na Figura 11 são mostradas duas telas dos sistemas SDAS (*Sales Deviation Analysis System*) e SIAP (Sistema Integrado de Acompanhamento de Projetos) que são dois produtos de uma outra família. Pode-se ver que os dois sistemas apresentam a mesma padronização que, embora não seja igual à das figuras anteriores, mantêm ainda assim uma certa semelhança, tais como: cores, existência e localização de barra de menus, botão de retorno, etc.

O conjunto de *templates* para as telas hoje adotado pela DITEL é resultado de projetos desenvolvidos nos últimos 6 anos.

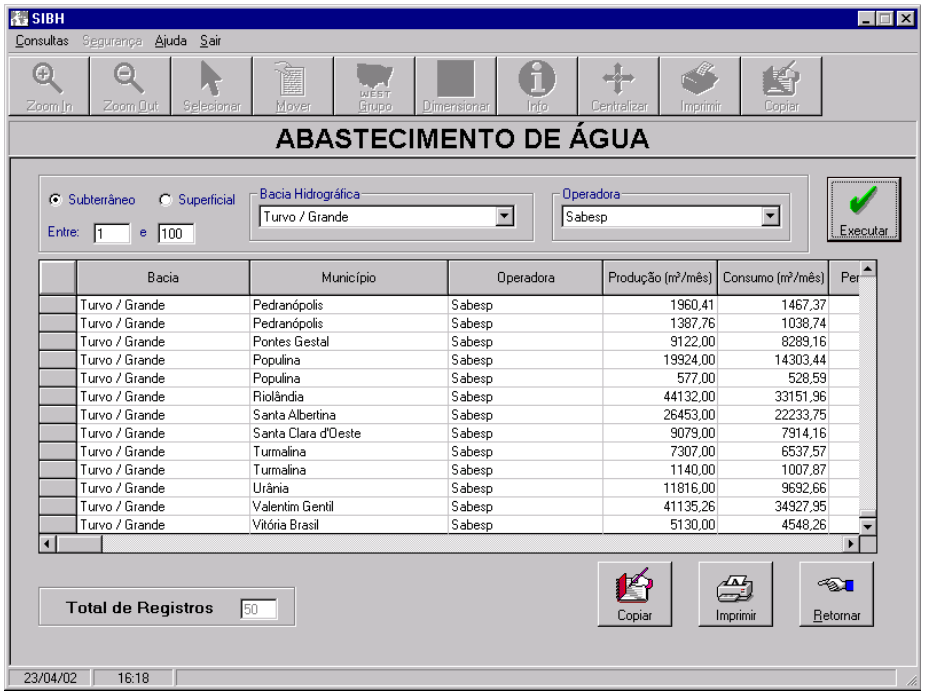

Figura 8. Tela de pesquisa de abastecimento de água do sistema SIBH

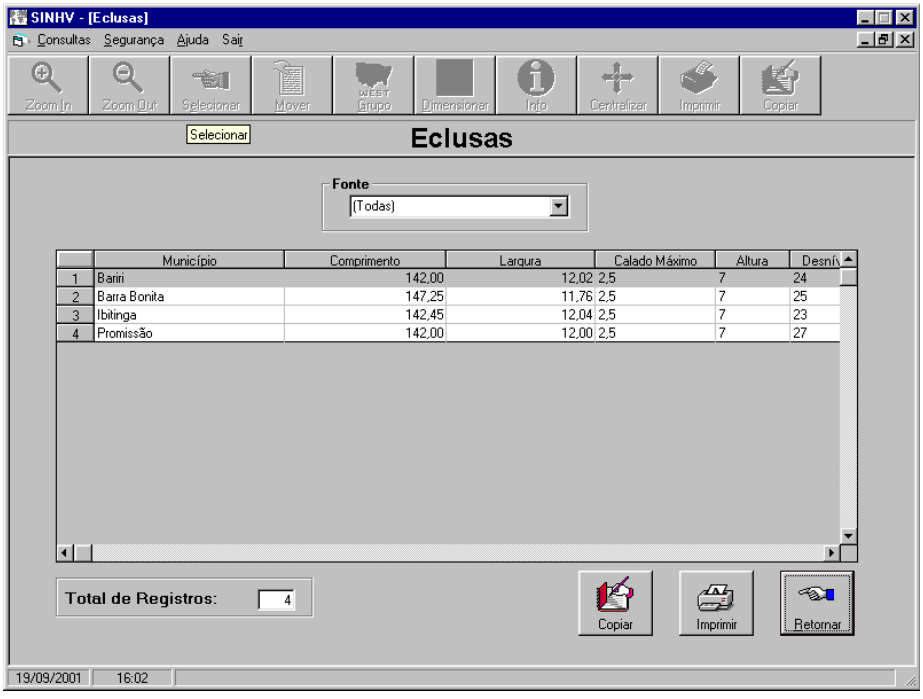

Figura 9. Tela de consulta de Eclusas da hidrovia do sistema SINHV

Figura 10. Tela de consulta de contratos do sistema SDAS

Figura 11. Tela de especificação dos itens a serem integrados na Fábrica, sistema SIAP

Recomendação 10. Crie um repositório de *templates* de telas

Os *templates* desenvolvidos para as diferentes famílias de produtos devem ser armazenados num repositório. O reuso pode acontecer não só entre projetos de uma mesma família, mas também sempre que se partilhar o estilo da interface. O uso destes *templates* tem facilitado: o desenvolvimento de protótipos diminuindo o prazo uma vez que há reaproveitamento de código e do projeto da interface, o treinamento da equipe de desenvolvimento e o treinamento dos usuários, pois as interfaces seguem o mesmo padrão. Além disso, Nielsen (1993) advoga que o reaproveitamento de código para o desenvolvimento das interfaces ajuda na consistência do software.

Se a equipe trabalhar com diversas plataformas de desenvolvimento, por exemplo: Visual Basic para software em rede e ASP para aplicações Web, crie repositórios diferentes com os padrões para cada uma das plataformas.

Recomendação 11. Crie um repositório de módulos ou partes de software prontos

Além dos *templates* deve-se criar um repositório de módulos e partes de software de caráter geral. Por exemplo: módulo de administração de usuários, a maioria dos aplicativos exigem algum tipo de controle sobre os usuários, portanto pode-se usar um módulo que execute esta tarefa em diversos sistemas.

Recomendação 12. Evolua seus *templates* de telas

O padrão utilizado para os *templates* deve evoluir ao longo do tempo. Procure mantê-lo atualizado conforme o grupo ganha experiência com novos desenvolvimentos e novos usuários, incorporando novas idéias e novas tecnologias e verificando a melhor forma de apresentação. Garanta o efeito de “modernidade”.

Um destes *templates* é adotado na prototipação sempre que não existe um padrão já definido pelo cliente. Optou-se pela padronização através de código pela facilidade de apresentação para novos membros da equipe e pela economia de tempo de desenvolvimento.

Recomendação 13. Faça um macro cronograma das etapas do projeto

Antes de iniciar o processo de desenvolvimento, faça um macro cronograma contemplando uma avaliação do tempo para cada etapa do projeto de forma a garantir que seja cumprida a janela de validade do projeto. Se nesta avaliação preliminar, for

verificado que será impossível cumprir o prazo estipulado, negocie com o cliente que será desenvolvida uma primeira versão do software, passível de ser feita na janela possível. Esta é uma recomendação baseada em idéias propostas no PMBOK (2000).

6.2 Boas práticas no desenvolvimento do projeto

As recomendações sugeridas para a etapa de desenvolvimento do software, serão divididas em quatro sub-ítens para melhor compreensão:

6.2.1 Boas práticas no levantamento de requisitos

Embora o levantamento de requisitos seja independente do uso da prototipação, optou-se por indicar algumas boas práticas desta atividade na medida em que esta etapa é de fundamental importância para o sucesso do projeto e qualidade do produto. Sugere-se, a seguir, algumas práticas que têm se mostrado úteis e que produzem resultados satisfatórios.

Recomendação 14. Identifique o contexto do trabalho dos usuários

Marcar as reuniões de levantamento de requisitos na empresa do cliente. Isto pode trazer problemas de interrupção dos trabalhos mas, em compensação, ajuda a entender como a empresa funciona. Se for necessário, é possível verificar na hora como alguma coisa funciona, o relacionamento entre as pessoas no seu local de trabalho, a organização da empresa, a interação entre grupos, o “ambiente”, etc. Esta é uma prática sugerida na Engenharia de Usabilidade. Esta é uma recomendação baseada em idéias propostas por Nielsen (1993).

Recomendação 15. Identifique outros sistemas que o usuário manipula

Procure conhecer outros sistemas que os usuários manipulam, padrões de interface existentes, outros sistemas que rodam na empresa, preferências dos usuários em relação aos sistemas existentes, etc. Mesmo que os sistemas existentes não pertençam ao domínio da aplicação a ser desenvolvida, os comentários dos usuários sobre cada um deles pode fornecer uma orientação sobre as preferências dos usuários.

Como sugerido por Nielsen (1993), se existir um sistema que execute as funcionalidades, ou parte delas, que o novo software deverá implementar, verifique como este sistema funciona, suas qualidades e defeitos. Esse sistema poderá ser usado como um protótipo inicial para o levantamento dos requisitos e fornecerá muitas idéias.

Recomendação 16. Conheça o melhor possível os futuros usuários

Conheça o melhor possível os futuros usuários, seu nível de conhecimento em informática, se estão abertos a novas idéias, se vão aceitar o sistema com facilidade, se já usam outros sistemas existentes, se já participaram no desenvolvimento de outros projetos, se a experiência foi boa etc. Estas informações vão fornecer idéias sobre os problemas que vão aparecer e sobre as metas de usabilidade a perseguir, como sugerido por Nielsen (1993). Por exemplo: se os usuários não conhecem informática e nunca usaram sistemas, devemos projetar o protótipo tendo como meta prioritária a facilidade de aprendizagem e memorização; por outro lado se os usuários forem experientes, a eficiência deve ser uma meta prioritária.

Recomendação 17. Procure conhecer os tipos de usuários que estarão envolvidos

Procure descobrir todos os tipos de usuários que estarão envolvidos e para cada tipo, defina quem são as pessoas a serem entrevistadas. Escolha, com o aval e compromisso da gerência, pelo menos um de cada tipo para ser entrevistado no levantamento de requisitos. Procure certificar-se de que não tenha sido esquecido nenhum tipo de usuário, pois se isto ocorrer, certamente os requisitos estarão incompletos.

Recomendação 18. Faça o levantamento iniciando pela gerência

Comece o levantamento, de preferência, com a gerência da empresa. Isto fará com que se tenha uma idéia mais global do sistema e onde se deseja chegar, sem os detalhes operacionais. Somente depois de saber qual a intenção da gerência em relação ao software, inicie o detalhamento dos requisitos do software.

Recomendação 19. Faça o levantamento com os usuários operacionais

Tendo a idéia global sobre o software, inicie então o levantamento junto aos usuários operacionais. Faça reuniões de levantamento, documente-as através de atas e protótipos exploratórios.

Recomendação 20. Envie a documentação antes das reuniões

Quando for a uma reunião de levantamento ou validação com os usuários, envie antecipadamente os documentos que serão analisados na reunião (atas ou outros documentos desenvolvidos pela equipe como conseqüência das reuniões anteriores) para que eles tenham tempo de os avaliar. Isso torna a reunião mais produtiva. Durante a reunião discuta o documento enviado, ponto por ponto, continuando com este processo junto ao usuário, até que seja considerado encerrado.

Recomendação 21. Use e abuse de desenhos, fluxogramas e protótipos

Utilize desenhos, fluxogramas e protótipos exploratórios para mostrar ao usuário o que está entendendo. Por exemplo, quando estiver levantando um processo faça um fluxograma das atividades do processo. Isto facilita o entendimento entre as duas partes. Rudd e Isensee (1994) afirmam que na empresa em que trabalham (IBM) é sempre aplicado o lema “Um desenho pode valer mais que mil palavras, mas um protótipo vale mais que mil desenhos” .

Recomendação 22. Observe os usuários realizando suas tarefas

Além das reuniões, procure observar os usuários realizando suas tarefas normais relativas ao processo que está sendo automatizado. Sente ao lado deles e apenas observe. Procure verificar quais as tarefas que dão mais trabalho, as mais difíceis, as mais demoradas e como o usuário resolve os problemas. Verifique gargalos e descontentamentos. Esta é uma das etapas do ciclo de vida da Engenharia de Usabilidade, citada por Nielsen (1993), mais especificamente faz parte da Análise Contextual de Tarefas, Mayhew (1999).

Recomendação 23. Documente todas as reuniões

Após todas as reuniões, sejam as de levantamento de requisitos, sejam as de validação do protótipo, faça a ata com todos os pontos discutidos, as decisões tomadas, as pendências e os responsáveis pela solução. A ata deve ser validada pelos participantes e nela devem constar desenhos, fluxogramas e os protótipos exploratórios.

Recomendação 24. Procure identificar padrões para as telas

Se houver algum padrão de tela na empresa do cliente, procure utilizá-lo, mesmo que seja parcialmente pois, normalmente, as pessoas preferem trabalhar com algo já conhecido do que com algo totalmente novo. Além disso, facilitará o treinamento dos usuários.

Se a empresa não tiver seus padrões de tela, escolha no repositório de *templates* de tela aquele que melhor se adapta a aplicação que será desenvolvida. Comunique a toda a equipe de desenvolvimento a decisão tomada.

6.2.2 Boas práticas na elaboração de protótipos

Apresenta-se nesta seção as boas práticas empregadas na elaboração do protótipo.

Recomendação 25. Identifique o contexto onde o sistema estará inserido

Antes de iniciar o desenvolvimento do protótipo identifique claramente os limites da aplicação, os atores e sistemas que irão interagir com o seu aplicativo, recomendado em toda a literatura de Engenharia de Software.

Recomendação 26. Identifique as principais funções, casos de uso

Identifique as principais funções ou casos de uso do sistema. Isto facilitará o entendimento global do software. Cada função ou caso de uso importante tende a ser um sub-sistema ou pelo menos uma tela principal a partir da qual serão acessadas as demais funcionalidades.

Recomendação 27. Defina os sub-sistemas que comporão o software

Se o produto automatizar um processo ou fluxo de trabalho, provavelmente cada departamento ou grupo de trabalho, que exerça atividades diferenciadas, deverá ter um sub-sistema diferente para a execução de suas tarefas. No exemplo da Figura 12, apresenta-se um processo extremamente simplificado de controle de reclamações para o departamento de Ouvidoria da empresa, apenas para ilustrar idéias. Neste exemplo são incluídos três departamentos: Protocolo, Ouvidoria e Departamento Técnico. Cada um desses departamento realiza tarefas diferentes e portanto deseja ver informações diferentes. Neste caso dever-se-ia optar por criar três sub-sistemas, um para cada departamento. Identifique, portanto, os papéis.

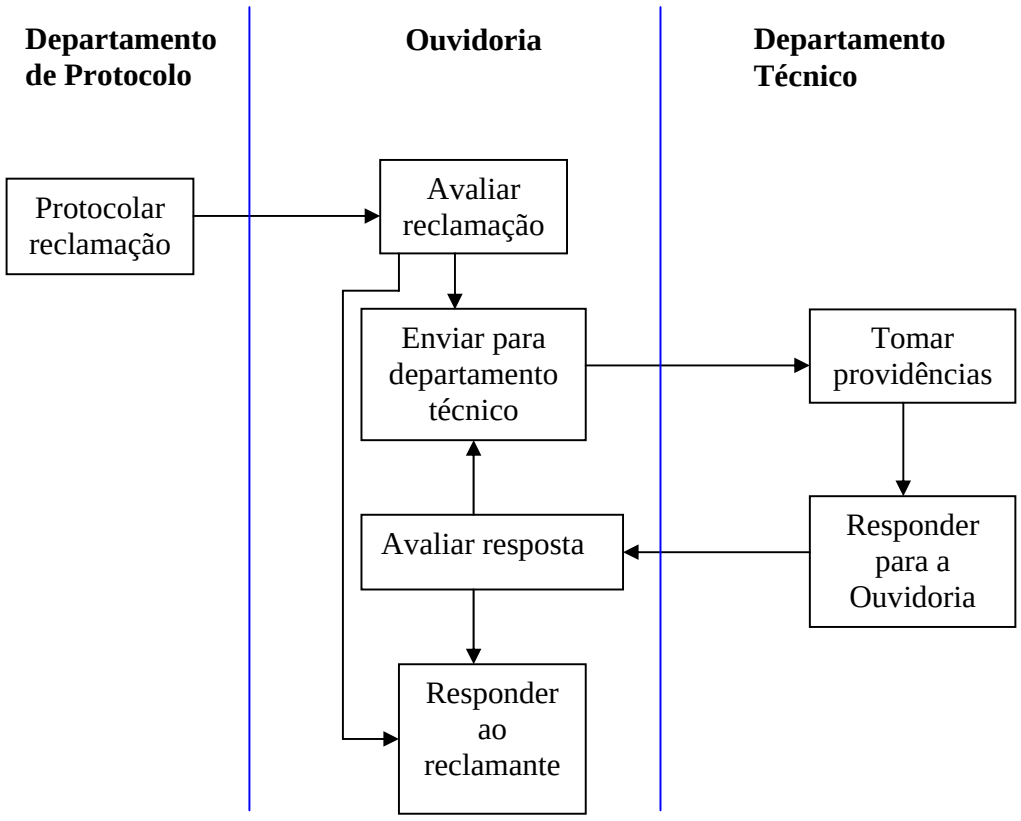

Figura 12. Exemplo ilustrativo de fluxo de processo

Recomendação 28. Esboce um rascunho do protótipo

Se foi desenhado um protótipo exploratório, durante o levantamento de requisitos, use-o como base para o desenvolvimento do protótipo. Se não foi criado este protótipo, exercite primeiro suas idéias no papel desenhando, as principais interfaces dentro do padrão escolhido, teste suas idéias discutindo com a equipe que participou do levantamento de requisitos antes de passar para a equipe de desenvolvimento.

Recomendação 29. Identifique por onde começar o protótipo

Para cada sub-sistema definido, inicie o desenho do protótipo pela interface que mais será usada. Coloque-se no lugar do usuário e pense qual será a tarefa prioritária ao longo do dia ou o que ele precisa ver e controlar. Inicie o protótipo por esta função. Exemplo: para um sistema de Call Center, a interface principal, provavelmente, será a de registro de atendimento; num sistema de controle do andamento de projetos, a interface principal, provavelmente, será uma lista com todos os projetos em andamento e um resumo das informações sobre sua situação atual (data de início, data prevista para término, atividade atual, situação, etc). A escolha correta da interface principal é fundamental pois influencia o desempenho do usuário na execução de suas tarefas.

Recomendação 30. Identifique as demais funcionalidades

A partir da interface principal, identifique as demais funções que o usuário deverá realizar. Crie, na interface principal, uma forma de acesso a estas funções (botões ou menu, conforme padrão estabelecido) e desenhe as interfaces identificadas. Repita o processo para cada uma delas até que todos os requisitos tenham sido atendidos. Por exemplo: Para o sistema de controle de projetos, poderíamos ter uma função para ver informações mais detalhadas de um projeto selecionado na tela principal, que por sua vez daria acesso a uma interface com a relação de compras efetuadas dentro do projeto.

Recomendação 31. Faça a programação do protótipo

Passe para a equipe de desenvolvimento todas as informações necessárias (padrão definido para as interfaces, atas de reunião, protótipo em papel, desenhos e fluxogramas) para que seja programado o protótipo.

Aconselha-se que pelo menos um membro da equipe de levantamento faça parte da equipe de programação do protótipo, pois facilitará na resolução de dúvidas.

Depois que o protótipo estiver funcionando, analise-o contra toda a documentação para verificar se não foi esquecido algum requisito e se está de acordo com os requisitos levantados. Observe as metas de usabilidade, exceto aquelas que envolvam desempenho.

6.2.3 Boas práticas na validação de protótipos

A validação do protótipo é uma fase muito importante no desenvolvimento do software.

Recomendação 32. Ressalte que protótipo não é sistema pronto

Quando os usuários vêem o protótipo têm sempre a impressão de que o sistema está pronto ou, pelo menos, praticamente pronto. Antes de apresentar o protótipo aos usuários é necessário esclarecer que o que está sendo apresentado é apenas um protótipo e quão longe se está de uma solução final. Se os usuários já participaram de um processo de desenvolvimento de software que utilizou prototipação será mais fácil, mas se for a primeira vez que estarão em contato com esta técnica haverá necessidade de esclarecer muito bem os usuários sobre os limites do protótipo para que eles não tenham expectativas que serão frustradas.

Procure esclarecer muito bem os usuários, principalmente o nível gerencial que é sempre quem está mais apressado para ter a solução funcionando.

Recomendação 33. Ressalte para o seu departamento comercial que protótipo não é sistema pronto

Certifique-se de que o departamento comercial de sua empresa entende que o protótipo não é o produto, e portanto, não pode ser vendido. Na primeira experiência que a equipe teve com prototipação, por mais que fosse explicado que era apenas um protótipo para ajudar na validação dos requisitos do sistema e apresentar como funcionaria a solução, o departamento comercial do cliente continuou vendendo a imagem de que já existia um sistema pronto. Isto causou muitos problemas, pois o sistema era muito grande e demorou muito tempo para ser concluído. Com isso, a equipe ficou numa situação complicadíssima, chegando a ser considerada até ineficiente, e quase o projeto teve de ser abortado.

Recomendação 34. Valide o protótipo primeiro com a gerência

Assim como o levantamento dos requisitos deve ser iniciado pela gerência, a validação do protótipo também deve ser feita inicialmente com eles. A gerência será mais capaz de dizer se o software corresponde, num nível mais alto, às necessidades da empresa. Nesta validação não é necessário chegar-se aos detalhes operacionais. Se o software corresponder às expectativas da empresa, então continue a validação. Se não corresponder aos anseios deles é melhor rever os requisitos antes de continuar.

Recomendação 35. Valide o protótipo com os usuários operacionais

Apresente o protótipo para os usuários operacionais. Se existirem usuários que utilizarão partes diferentes do sistema, como por exemplo, departamentos diferentes, escolha usuários que consigam validar todas funcionalidades do sistema. Verifique se a estrutura dos dados está correta.

Observe se eles estão satisfeitos com as interfaces, se elas correspondem ao que eles esperam. É muito mais fácil mudar no protótipo do que no software pronto.

Recomendação 36. Valide até encontrar erros

Faça os testes com o protótipo. Assim que encontrar os primeiros erros, pare os testes, corrija o protótipo e reinicie a sessão de testes. Relembrando a recomendação de Nielsen (1993):

> "Mudanças feitas para resolver problemas de usabilidade podem introduzir novos problemas de usabilidade. Deve-se lembrar que novos problemas de usabilidade aparecem nos testes depois que os mais óbvios problemas foram corrigidos, portanto, não há necessidade de testar exaustivamente o projeto inicial, uma vez que ele vai mudar de qualquer modo. A interface deve ser alterada e re-testada tão logo um problema de usabilidade tenha sido detectado e entendido, assim aqueles problemas que tinham sido mascarados pelo problema inicial passam a ser encontrados."

Recomendação 37. Valide algoritmos mais complicados

Se existirem algoritmos mais complicados valide-os, seja através de sua implementação no protótipo, seja através de um documento descrevendo o algoritmo, neste caso, tome cuidado para que a linguagem de apresentação seja simples e possa ser entendida pelo usuário.

Recomendação 38. Apresente o protótipo para o maior número possível de pessoas

Quanto mais o protótipo for avaliado por pessoas diferentes, maior garantia se terá de que o sistema atenderá às expectativas. Aproveite todas as oportunidades para mostrá-lo. Esta é uma recomendação sugerida por Rudd e Isensee (1994) da IBM.

Recomendação 39. Preocupe-se com o controle de versão

Utilize a estrutura de armazenamento e compartilhamento dos artefatos criada antes do início do projeto. Lembre-se que a prototipação é um processo iterativo e portanto haverá várias versões do protótipo. Deve-se manter cada versão do protótipo, inclusive o exploratório e os documentos de avaliação, que indicam as mudanças a serem feitas e o motivo das mesmas.

Recomendação 40. Preocupe-se com o prazo previsto para esta etapa

A fase de validação exige uma série de reuniões e modificações no protótipo e no modelo de dados. Não existe um número fixo de reuniões para se avaliar o protótipo. Cuide para que esta etapa não se prolongue demais. Fique atento para que o usuário não se perca em características menos importantes atrapalhando a validação como um todo.

Não se deve esquecer que o prazo para a realização do projeto está pré estipulado. Existe uma tendência em se procurar levantar e validar todos os requisitos nesta etapa, mesmo os que não são prioritários. Se houver tempo disponível esta prática é correta mas fique atento ao tempo que foi destinado a esta tarefa. Se ele for ultrapassado, vai comprometer as próximas etapas e a obtenção do produto dentro de sua janela de validade.

6.2.4 Boas práticas na finalização do produto

Seguem as recomendações para as etapas de desenvolvimento, homologação e entrega do produto.

Recomendação 41. Reveja o cronograma e negocie o escopo possível

Com o protótipo validado, pode-se então entrar na etapa de programação, mas antes de iniciar deve-se rever o cronograma. Tendo em mãos o protótipo, é possível avaliar-se a complexidade de cada tela (número de objetos na tela, complexidade de implementação dos objetos, algoritmos complicados...) e o tempo necessário para finalizar sua programação e os testes. Para realizar esta avaliação, baseie-se no tempo gasto em projetos anteriores.

Faça um cronograma detalhado contendo cada uma das atividades (telas a serem programadas, testes, importação de bases com dados existentes...). Atribua a cada membro da equipe as tarefas que ficarão sob sua responsabilidade. Procure usar o paralelismo de tarefas, conforme recomendado por Correia (2003).

Se o resultado desta programação de tarefas resultar num prazo dentro da janela de validade, ótimo. Se não estiver veja se é possível alocar mais profissionais de forma a ser possível cumprir o prazo ou negocie a retirada de algumas funcionalidades que não sejam essenciais, nesta versão do produto.

Recomendação 42. Acompanhe o cronograma de programação

Convença a equipe de que devem atualizar o cronograma toda vez que iniciam ou terminam uma tarefa. Parece que esta é uma tarefa fácil mas a equipe que não está acostumada sempre “esquece” de fazê-lo.

Defina uma periodicidade de avaliação do cronograma. Esta atividade é muito importante pois se ocorrer um desvio nos prazos ele é detectado imediatamente, os riscos são avaliados e atitudes para a correção de rumos podem ser tomadas em tempo. Conforme Correia (2003):

> “Sabe-se que o planejamento e controle das atividades de um projeto são extremamente importantes e garante sua correta finalização. Mas, nos projetos com ciclo acelerado, o planejamento e controle devem ter mais ênfase, já que não há tempo para ser desperdiçado.”

Recomendação 43. Escolha uma equipe de testes independente da que desenvolveu

Embora esta seja uma recomendação feita em toda literatura de Engenharia de Software, o que se observa na prática, em equipes pequenas e premidas pelo prazo, é que

os próprios desenvolvedores testam o software. Esta é uma prática muito perigosa. O desenvolvedor, quando testa, sabe o que o software deve fazer e acaba, mesmo sem querer, tentando provar que ele está correto.

A existência de uma equipe de testes ajuda a encontrar erros que de outra forma não seriam descobertos.

Não se pode esquecer que é muito melhor encontrar os erros antes da entrega do produto do que deixar que o cliente encontre-o. Um erro simples e fácil de ser corrigido pode causar um impacto muito negativo no usuário.

Uma outra atitude importante é registrar os erros encontrados, sua causa e, nos casos mais complexos, como a correção foi feita. Estas informações ajudarão a equipe a não cometer os mesmos erros e servem como uma fonte de aprendizado.

Recomendação 44. Estabeleça um prazo para a homologação do produto

Após a entrega do produto, estabeleça um prazo, junto com o cliente, para que ele teste e homologue o produto. Quando não há um tempo combinado entre as partes, ou o cliente não encontra tempo para realizar os testes ou fica pedindo alterações que não estavam no escopo desta versão. Combine com ele que na etapa de homologação só serão corrigidos erros de implementação. Mudanças de escopo, tais como: novas consultas, relatórios e funcionalidades, serão implementados numa nova versão do produto. Como estamos trabalhando com ciclo acelerado, o tempo entre uma versão e a próxima é pequeno e, portanto, o cliente aceita esperar pela nova versão.

6.3 Boas práticas no aproveitamento do protótipo após a entrega

Após a entrega do produto, ou de uma versão dele, algumas atividades devem ser realizadas para garantir a qualidade do produto e dar subsídio para a evolução da equipe e do processo.

Recomendação 45. Faça apenas a documentação necessária

Se o software for descartável, os documentos gerados durante o desenvolvimento são suficientes pois o produto não sofrerá manutenção.

Para software não descartável, além dos documentos já gerados durante o processo de desenvolvimento, recomenda-se a elaboração do material de treinamento e um *help online* ou o manual do usuário, conforme o cliente preferir. Como o software é

desenvolvido de forma evolutiva, a cada nova versão basta a atualização destes documentos.

Recomendação 46. Melhore seu repositório de *templates*

Se no processo de desenvolvimento deste produto foi criado um novo padrão de telas adicione-o ao seu repositório de *templates*.

Muitas vezes usa-se um *template* para iniciar um projeto e durante o desenvolvimento faz-se uma melhoria seja na codificação, seja nas características de usabilidade. Neste caso, deve-se ao final do projeto substituir o padrão antigo pelo novo.

Recomendação 47. Melhore seu repositório de módulos ou partes do software de caráter geral

Ao terminar um software, verifique se existem partes dele que têm um caráter geral e que poderão ser aproveitadas em novos produtos. Por exemplo: uma tela onde o usuário pode navegar entre os diretórios e escolher um determinado tipo de arquivo, uma tela para digitação de login e senha do usuário, etc. Guarde então estas partes selecionadas num repositório de partes e módulos já programados. A utilização destas partes em novos projetos acelerará ainda mais o tempo de desenvolvimento, conforme corrobora Correia (2003) em seu trabalho:

> "A reutilização é um bom caminho para proporcionar a aceleração do desenvolvimento. É importante garantir que os artefatos já estejam prontos para uso, desde o início do projeto, permitindo que o esforço gasto em um projeto é reutilizado (reciclado), posteriormente, em outros."

Se foi usada uma parte de software e esta sofreu melhorias, substitua a versão antiga pela nova no repositório. Com esta prática seu repositório estará sempre atualizado.

Recomendação 48. Guarde o cronograma do projeto

Como é recomendado na literatura de Engenharia de Software e também na de Qualidade de Software, deve-se arquivar, de maneira adequada, o cronograma de realização do projeto para que sirva de base histórica para a previsão de prazos nos próximos projetos.

A falta de um histórico faz com que as previsões sejam sempre baseadas na experiência e sensibilidade da equipe e não em valores reais obtidos ao longo do tempo.

Armazene, também, referências sobre a complexidade do projeto, como por exemplo, número de telas do protótipo inicial (exploratório) e final (sistema), para dar noção da possibilidade de crescimento do sistema.

Recomendação 49. Retorne ao campo para avaliar o produto

O principal objetivo do trabalho de usabilidade, depois da liberação de um produto, é obter dados de usabilidade para a próxima versão e para futuros produtos, pois um produto recém liberado pode ser visto como um protótipo de futuros produtos. Deve-se avaliar como os usuários reais usam a interface para a execução das suas tarefas no seu ambiente de trabalho.

O retorno de campo pode ser obtido através da análise das reclamações dos usuários, pedidos de modificação dos requisitos e chamadas ao suporte ou pelos mesmos métodos de estudos de campo e análise de tarefas. Esta é uma recomendação encontrada na literatura de Engenharia de Usabilidade.

Capítulo 7. Conclusões

Neste capítulo são apresentadas as conclusões finais deste trabalho, as principais contribuições e sugestões para futuros trabalhos que poderiam complementar este estudo.

7.1 Conclusão

Devido às mudanças de mercado, atualmente, o grande desafio das empresas de desenvolvimento de software é conseguir desenvolver seus produtos rapidamente, com custo baixo e com qualidade. Os produtos, também, devem corresponder a todas as expectativas dos usuários, sejam elas de caráter prático ou subjetivas.

Neste trabalho buscou-se estudar em que medida a prototipação pode ser uma ferramenta aceleradora do processo de desenvolvimento de software e mais, em que caso ela pode ser usada como documentação dos requisitos do software.

Relembrando, este trabalho trata especificamente de sistemas de informação, de pequeno ou médio porte, ou ainda descartáveis, altamente interativos e que não exigem muitos algoritmos. No caso de um sistema grande porte, se ele puder ser subdividido em módulos de pequeno ou médio porte, relativamente independentes, as mesmas considerações podem ser aplicadas a cada um dos módulos, como se fossem softwares separados, contanto que não se esqueça de tratar as conexões entre eles.

Apresenta-se, a seguir, um resumo das considerações que levam à conclusão.

- **Prototipação como ferramenta de levantamento de requisitos**

  Levantar requisitos não é uma tarefa fácil. O domínio de conhecimento dos desenvolvedores e dos usuários, normalmente, é muito diferente. Entender o que o usuário realmente deseja pode ser complicado, já que muitas vezes nem ele sabe direito o que deseja. Esta tarefa pode ser facilitada considerando-se o uso dos diversos tipos de protótipo (em papel, versões anteriores do produto, outros produtos de mercado, etc), É criada uma linguagem comum, que as duas partes entendem. É muito mais fácil mostrar uma interface e perguntar ao usuário: Ao iniciar a execução do software, que informações você gostaria de ver primeiro? Que informação você estará controlando prioritariamente? Que tamanho tem este campo? Que valores esta variável pode assumir? A partir desta atividade, qual a próxima a ser executada? Este relatório corresponde ao que você precisa? Você acha que a cor desta tela será agradável, uma vez que será usada a maior parte do dia? Esta interface está suficientemente simples de ser usada? Você prefere usar mouse ou tecla de atalho?

Além disso, o protótipo ajuda no levantamento de todos os requisitos, que de outra forma poderiam ser esquecidos.

A prototipação, também, permite que requisitos funcionais e não funcionais, principalmente relativos a questões de usabilidade, sejam obtidos mais facilmente.

- **Prototipação como ferramenta de análise e projeto do software**

O protótipo facilita a tarefa do analista para descobrir quais são os principais processos, a atribuir as funcionalidades aos processos, a definir a arquitetura da aplicação, a definir o domínio da aplicação, já que fica bastante claro o escopo do software, a definir as interfaces com terminadores externos. Ele também auxilia na modelagem de dados, uma vez que as informações, suas características e relacionamentos são claramente levantados e validados pelos usuários.

- **Prototipação como ferramenta de apresentação e validação dos requisitos**

O uso de protótipos para a validação dos requisitos é uma função reconhecida em toda a literatura, desde o início da Engenharia de Software até os dias atuais. O protótipo garante que o cliente saiba exatamente o que vai receber logo nas primeiras fases do projeto. Incorreções podem ser verificadas e corrigidas no protótipo, evitando, assim, retrabalho, frustração do cliente, desperdício de tempo e diminuição dos custos do projeto.

A falta de funcionalidades que não foram levantadas, ou porque o cliente esqueceu ou porque o desenvolvedor não percebeu, pode ser detectada e corrigida neste momento.

Questões de usabilidade, tais como: cores empregadas, ícones, localização dos objetos, facilidade de uso, facilidade de entendimento e de treinamento, etc são encontradas e corrigidas. Não há nada pior do que um cliente descontente com a interface que foi projetada.

- **Prototipação como acelerador da programação**

Quando entregamos aos programadores um protótipo, eles estão recebendo uma especificação muito clara do que o software deve realizar, numa linguagem de comunicação que eles compreendem facilmente. Isto reduz muito o erro de comunicação entre analistas e programadores, aumentando a eficiência e garantindo os resultados.

Além disso, como o protótipo é desenvolvido na linguagem de implementação, uma vez que ele esteja validado, parte da programação já está feita, o que, obviamente, reduz muito o tempo necessário para o desenvolvimento do produto.

- **Prototipação na documentação dos requisitos**

A documentação de requisitos é necessária, basicamente, para: garantir que todos os requisitos foram levantados e serão corretamente implementados; possibilitar a manutenção do sistema; garantir o gerenciamento dos requisitos.

Nas situações especiais tratadas neste trabalho, fica claro de que o protótipo é uma das melhores ferramentas para garantir a primeira necessidade.

Em relação a segunda necessidade, manutenção do software, o protótipo já é a especificação do que foi o sistema deve fazer. Definir e avaliar mudanças e seu impacto sobre o produto utilizando-se o protótipo é fácil e quando uma manutenção é necessária, o requisito que precisa ser alterado é rapidamente identificado e o impacto da mudança pode ser avaliado.

Na questão de gerenciamento dos requisitos, é importante que se tenha uma estrutura de controle de versões para garantir o gerenciamento dos requisitos usando-se os protótipos.

Além disso, o uso do protótipo como uma “especificação viva”, faz com que se economize tempo de documentação e revisão do documento de especificação, além de garantir que os requisitos do usuário não ficarão apenas documentados mas serão realmente implementados no software.

Precisa-se deixar claro que, embora se utilizando o protótipo para documentar os requisitos, o modelo de dados é um documento que não pode ser abandonado.

Concluindo, cada um destes pontos *de per si* , seria suficiente para justificar o uso da prototipação. O conjunto de todas estas vantagens torna a prototipação ainda mais interessante como fator de aceleração no desenvolvimento de software em ciclo acelerado.

Ainda mais, a criação de repositórios de templates de telas e de partes de software, propicia uma efetiva aceleração do processo de desenvolvimento, além de garantir a consistência entre produtos de uma mesma família.

O reuso propiciado por estes repositórios, garante economia de tempo nas diversas etapas de desenvolvimento, além de que, se forem usadas as recomendações 46 e 47, a cada novo projeto, as telas e partes de programas serão revistos e melhorados, seja pela otimização da programação, melhoria na documentação, melhores características de usabilidade ou ainda atualização tecnológica, isto propiciará a melhoria dos repositórios refletindo na qualidade dos próximos projetos.

Não se pode deixar de lembrar que para se conseguir trabalhar em ciclo acelerado, a gestão do projeto é fundamental. Como o prazo do projeto é muito curto, a elaboração de cronogramas realistas, seu acompanhamento rigoroso, a tomada de decisões tão logo problemas sejam detectados e principalmente o comprometimento da equipe com o projeto são fatores fundamentais para o sucesso do projeto.

7.2 Limitações do estudo

A avaliação prática deste estudo, teve como universo de análise apenas os projetos desenvolvidos na empresa da autora, mais especificamente no AST da DITEL. Embora o número de projetos já desenvolvidos, utilizando a prototipação como ferramenta básica, seja relativamente grande (mais de vinte softwares e dois sistemas que podem ser considerados de grande porte, mas que foram subdivididos em módulos relativamente independentes) e a equipe esteja utilizando e aprimorando a técnica há mais de oito anos, ainda assim seria desejável que se aumentasse o universo de estudo, avaliando-se esta ferramenta em outras empresas.

Uma outra limitação do estudo é que o grupo não tem trabalhado com o paradigma de orientação a objetos, embora estejamos atualmente trabalhando com aplicações em ambiente Web. Parece-nos que a prototipação continuará a ser importante, independentemente de se trabalhar com análise estruturada ou com orientação a objetos, uma vez que de qualquer forma as interfaces são fundamentais, já que são o que o cliente percebe do sistema. De qualquer modo, valeria a pena fazer um estudo mais apurado para este caso.

Acredita-se que o roteiro apresentado também pode ser útil para “usuários-programadores”, no entanto, não foi feito um estudo de aplicação dele por este tipo de desenvolvedor. Com certeza eles não usarão todas as recomendações e seria interessante avaliar quais eles consideram mais importantes e como o roteiro pode colaborar com eles.

7.3 Principais contribuições e futuros desenvolvimentos

A primeira contribuição deste trabalho foi o resgate da importância da prototipação, através da releitura da Engenharia de Requisitos e Engenharia de

Usabilidade. A bibliografia sobre o assunto específico é antiga, o que mostra que durante os últimos anos, o assunto não mereceu destaque. Novos paradigmas da engenharia de software têm se destacado na literatura e a prototipação aparece apenas como uma das atividades dos novos ciclos de vida propostos. Como conseqüência, não se evoluiu muito na análise de novos usos para a prototipação e no seu aproveitamento como fator de aceleração de ciclos de desenvolvimento. Com as empresas, principalmente as de pequeno porte, tendo que desenvolver software em ciclos cada vez mais acelerados, o resgate e ampliação de uso desta ferramenta se faz necessário.

Como contribuição acadêmica deste trabalho, foi desenvolvido um roteiro de boas práticas para auxiliar desenvolvedores com o benefício da prototipação, distinguindo-se critérios que auxiliam o bom uso da técnica. Espera-se que este roteiro possa colaborar com o trabalho de equipes de desenvolvimento de software com características similares à da DITEL, bem como, fornecer um caminho para aqueles "usuários-programadores" que desejem desenvolver pequenos softwares para seu grupo de trabalho.

A discussão do uso ampliado da prototipação em ciclo de vida acelerado, a partir da visão crítica da prática profissional visando substituir com vantagens a especificação de requisitos de software é uma outra contribuição deste estudo.

Do ponto de vista profissional, a principal contribuição deste trabalho é a revisão e documentação das práticas atualmente empregadas na empresa, bem como a análise crítica da prática *versus* a teoria, e a melhoria do processo profissional, especialmente pela incorporação do processo de Engenharia de Usabilidade. Embora estas práticas fossem conhecidas pela equipe nas suas atividades diárias, sua documentação facilitará a uniformização dos conhecimentos e o treinamento de novos membros. A revisão efetuada no processo usado pela DITEL, resultou em sua melhoria. A elaboração do roteiro permitiu que se trabalhasse idéias e procedimentos que eram executados de forma automática, muitas vezes intuitiva, revisando-os sob a luz dos conceitos de usabilidade. Além disso, permitiu que se incorporasse recomendações encontradas na literatura.

Algumas sugestões para futuros desenvolvimentos para dar continuidade a este trabalho e que contribuiriam com o trabalho profissional são:

1. Formalizar o repositório de *templates,* documentando-o adequadamente;

2. Rever e atualizar o repositório de templates existentes, através de estudos de usabilidade das telas e da documentação de padrões para os elementos de tela, como sugerido por Mayhew (1999);

3. Revisão do processo de desenvolvimento de software da DITEL, buscando-se atualizá-lo para novos paradigmas de desenvolvimento;

4. Embora se tenha guardado muitos cronogramas com os prazos previstos e os realizados, ainda não foi criada uma estrutura para armazenamento e

recuperação destas informações, portanto usa-se a experiência de desenvolvimento anteriores para fazer a previsão de prazos para a realização das tarefas. Pretende-se criar uma estrutura de armazenamento destes históricos, para que se tenha uma melhoria na previsão de prazos para as atividades. Além disso, poderia ser criada uma forma de classificação da complexidade de cada tipo de tela e manter um histórico dos tempos reais do desenvolvimento;

5. As recomendações aqui apresentadas são fruto da experiência da equipe da DITEL; como continuidade, estas recomendações precisariam ser aplicadas em outras empresas para análise de sua efetividade em outros ambientes de desenvolvimento e melhoria do roteiro;

6. Sugere-se também a análise deste roteiro em empresas que trabalhem com o paradigma de orientação a objetos verificando-se quais artefatos podem ser substituídos pelos protótipos e qual o impacto gerado em termos de aceleração do ciclo de desenvolvimento.

REFERÊNCIAS BIBLIOGRÁFICAS

BAHN, D. L; NAUMANN, J. D. Evaluation versus construction: distinguishing user review of software prototypes from conventional review of software specifications. In: CONFERENCE ON COMPUTER PERSONNEL RESEARCH, 1997, San Francisco, CA. Proceedings...New York: ACM Press, 1997. p.240-145.

BOAR, B. H. Application prototyping: a requirements definition strategy for the 80's. New York: John Wiley, 1984. 210p.

CORREIA, M. G. Proposta para a gestão do processo de desenvolvimento de projetos de software em um ciclo de vida acelerado. 2002. .Dissertação (Mestrado Profissional) Centro de Aperfeiçoamento Tecnológico, Instituto de Pesquisas Tecnológicas, São Paulo.

DEMARCO, T. Structured Analysis and System Specification. New York: Youdon Press, 1978.

GANE, C; SARSON, T. Structured Systems Analysis and Design**.** New York: Improved Systems Technologies, Inc., 1977.

GOMAA, H.; SCOTT, D. B. H. Prototyping as a tool in the specification of user requirements. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 5., 1981, San Diego, CA. Proceedings....[S.l.]: IEEE, 1981. p.333-342.

GREINER, M. R. LETAC: técnica para análise de tarefas e especificação de fluxos de trabalho cooperativo. 2000. 187p. Dissertação (Mestrado) – Escola Politécnica; Universidade de São Paulo, São Paulo.

HAKIN, J.; SPITZER, T. Effective prototyping for usability. In: IEEE PROFESSIONAL COMMUNICATION SOCIETY INTERNATIONAL PROFESSIONAL COMMUNICATION CONFERENCE AND ANNUAL ACM INTERNATIONAL CONFERENCE ON COMPUTER DOCUMENTATION: TECHNOLOGY & TEAMWORK, 18., 2000, Cambridge, Mass. Proceedings...Piscataway, NJ: IEEE Educational Activities Department, 2000. p.47-54.

HARTSON, H. R.; HIX, D. Human-computer interface development: concepts and systems for its management. ACM Computing Surveys, New York, v.21, n.1, p.5-92, Mar. 1989.

LICHTER, H.; SCHNEIDER-HUFSCHMIDT, M.; ZULLIGHOVEN, H. Prototyping in industrial software projects: Bridging the gap between theory and practice. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 15., 1993, Baltimore. Proceedings...Los Alamitos, CA: IEEE Computer Society Press, 1997. p.221-229.

MARTINEZ, M. L. Um método de *webdesign* baseado em usabilidade. 2002. 310p. Tese (Doutorado) – Escola Politécnica, Universidade de São Paulo, São Paulo.

MAYHEW, D. J. The usability engineering lifecycle: a practitioner´s handbook for user interface design. São Francisco: Morgan Kaufmann, 1999. 542p. (The Morgan Kaufmann series in interactive technologies)

NATIONAL CANCER INSTITUTE. Usability Basics: Provides a basic overview of and general information about usability. Disponível em: <http://usability.gov/basics/index.html#definition> Acesso em 24 fev. 2003.

NIELSEN, J. How to conduct a heuristic evaluation. 1995a. Disponível em: <http://www.useit.com/papers/heuristic/heuristic_evaluation.html> Acesso em:17 fev. 2003.

_____. Ten usability heuristics. 1995b. Disponível em: <http://www.useit.com/papers/heuristic/heuristic_list.html> Acesso em:17 fev. 2003.

_____. Usability engineering. Boston : AP Professional, 1993. 361p.

PAULK, M. et al. Capability maturity model for software (version 1.1): technical report. 1993a. Disponível em: <http://www.sei.cmu.edu/publications/documents/93.reports/93.tr.024.html> Acesso em:17 fev. 2003.

_____. Practices of the capability maturity model version 1.1: technical report. 1993b. Disponível em: <http://www.sei.cmu.edu/publications/documents/93.reports/93.tr.025.html> Acesso em:17 fev. 2003.

PMBOK - Project management body of knowledge. Belo Horizonte: Project Management Institute, 2000. 133p. (tradução livre realizado pelo PMI de Minas Gerais)

PORTELLA, C. R. R. Técnicas de prototipação na especificação de requisitos e sua influência na qualidade do software**.** 1994. Dissertação (Mestrado em Informática) - Instituto de Informática da Pontifícia, Universidade Católica de Campinas, Campinas.

PRESSMAN, R. S. Engenharia de software. 5.ed. Rio de Janeiro: McGraw-Hill, 2002. 843p.

_____. Engenharia de software. São Paulo: Makron Books, 1995. 1056p.

RIBEIRO, R.A.; BUNKER, R.E. Prototyping analysis, structured analysis, prolog and prototypies. In: ACM SIGCPR CONFERENCE ON MANAGEMENT OF INFORMATION SYSTEMS PERSONNEL, 1988, Maryland. Proceedings...New York: ACM Press, 1988. p 109-118.

RIZZO, A.; MARCHIGIANI, E.; ANDREADIS, A. The AVANTI project: Prototyping and evaluation with a cognite walkthrough based on the Norman's model of action. In: CONFERENCE ON DESIGNING INTERACTIVE SYSTEMS : PROCESSES, PRACTICES, METHODS, AND TECHNIQUES: PROCESSES, PRACTICES, METHODS, AND TECHNIQUES, 1997 , Amsterdam, NDL. Proceedings...New York: ACM Press , 1997. p. 305-309.

RUDD, J.; ISENSEE, S. Twenty-two tips for a happier, healthier prototype. Interactions, New York, v.1, n.2, p. 35-40, Apr. 1994.

SWEBOK: Guide to the software engineering body of knowledge - trial version. 2002. Disponível em: <http://www.architecture.myweb.nl/images/documents/swebok2002.pdf> Acesso em: 15 set. 2002

THOMPSON, M.; WISHBOW, N. Prototyping: tool and techniques: improving software and documentation quality through rapid prototyping. In: ANNUAL INTERNATIONAL CONFERENCE ON SYSTEMS DOCUMENTATION, 10., 1992**,** Ottawa. Proceedings**...** New York: ACM Press, 1992. p.191-199.

WEINBERG, V. Structured Analysis. New York: Yourdon Press, 1978.

YOURDON, E. Análise estruturada moderna. Rio de Janeiro: Campus, 1990. 836p.

BIBLIOGRAFIA RECOMENDADA

ALAVI, M. An assessment of the prototyping approach to information systems development. Communications of the ACM, New York, v.27, n.6, p.556-563, June 1984.

BÄUMER, D. et al. User interface prototyping: concepts, tools and experience. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 18., 1996, Berlin. Proceedings...Los Alamitos, CA: IEEE Computer Society Press, 1996. p.532-541.

BERGHEL, H.. New Wave prototyping: the use and abuse of vacuous prototypes. Interactions, New York, v.1, n.2, p.49-54, Apr.1994. Disponível em: <http://www.acm.org/~hlb/publications/new-wave/new-wave.html> Acesso em: 17 fev. 2003.

BOEHM, B. W.; GRAY, T. E.; SEEWALDT, T. Prototyping vs. Specifying: a multi-project experiment. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 7., 1984, Orlando, FL. Proceedings... New York: ACM Press, 1984. p.473-484.

CARNEIRO, W. L. Engenharia e pratica de requisitos de informações. 2000. 51p. (Mestrado em Informática Qualidade de Software) – Universidade Católica de Brasília, Brasília.

CATANI, M.B.; BIERS, D.W. Usability evaluation and prototype fidelity: users and usability professionals. In: ANNUAL MEETING OF THE HUMAN FACTORS AND ERGONOMICS SOCIETY, 42**.,** 1998. Proceedings: Disponível em: <http://www.humanfactors.com/download/dec98.asp> Acesso: 15 set. 2002.

CORNACHIONE JR, E. B. Prototipação em sistemas de informações gerenciais: uma abordagem de gestão econômica de empresas. São Paulo: FEA, s.d. 16p. Disponível em: <http://www.eac.fea.usp.br/eac/arquivos/artigos/edgard/cbc_edg.pdf> Acesso em: 24 fev. 2003.

GANEMAN-RUSSELL. When Requirements Interfere With Usability Goals. Disponível em: <http://www.ganemanrussell.com/newsletter/07012001.html> Acesso em: 22 fev. 2003.

HURST, A. One-Day Usability Testing. 10 may 2000. Disponível em: <http://www.clickz.com/ebiz/prod_manage/article.php/830171> Acesso em: 22 fev. 2003.

INSTONE, K. Usability Engineering for the Web. Disponível em: <http://www.w3j.com/5/s3.instone.html> Acesso em: 23 fev. 2003.

KIRAKOWSKI, J. (Cop.) Questionnaires in Usability Engineering A List of Frequently Asked Questions**:** Human Factors Research Group, Cork, Ireland. Disponível em: <http://www.ucc.ie/hfrg/resources/qfaq1.html> Acesso em: 24 fev. 2003.

MANTEI, M. M.; TEOREY, T. J. Cost/benefit analysis for incorporating human factors in the software lifecycle. Communications of the ACM, New York, v.31, n.4, p.428-439, Apr. 1988.

MAYHEW, D. J. The usability engineering lifecycle: formerly managing the design of the user interface. In: CONFERENCE ON CHI98 SUMMARY: HUMAN FACTORS IN COMPUTING SYSTEMS, 1998, Los Angeles, CA. Proceedings...New York: ACM Press, 1998. p.127-128.

NATIONAL CANCER INSTITUTE. Usability: methods for designing usable web sites developing prototypes. Disponível em: <http://usability.gov/methods/prototype_development.html> Acesso em: 22 fev. 2003.

NIELSEN, J. Guerrilla HCI: Using Discount Usability Engineering to Penetrate the Intimidation Barrier. 1994. Disponível em: <http://www.useit.com/papers/guerrilla_hci.html> Acesso em 25 fev. 2003.

_____**.** Return on Investment for Usability.. Alertbox, January 7, 2003. Disponível em: <http://www.useit.com/alertbox/20030107.html> Acesso em: 22 fev. 2003

_____. Finding usability problems trough heuristc evaluation. In: SIGCHI CONFERENCE ON HUMAN FACTORS IN COMPUTER SYSTEMS, 1992, Monterey, CA. Proceedings... New York: ACM Press, 1992. p.373-380.

OSSWALD, M**.** This Issue's Topic: User Testing: Does Your Site Need It? Hanson News, v.1, n.7, July 2001. Disponível em: <http://www.hansoninc.com/newsletter/news-i7.asp> Acesso em: 21 fev. 2003.

PANOIU, A. Moving from direct manipulation to indirect management - impact on UI design methodologies. In: WORKSHOP ENSURING USABLE INTELLIGENT USER INTERFACES, 1999. Eletronics Proceedings...Disponível em: <http://www.dfki.de/imedia/workshops/i3-spring99/w2/index.html> Acesso em: 23 fev. 2003.

RATIONAL & CONTEXT**.** Building web solutions with rational unified process: unifying de creative design process and the software engineering process: A rational software & context integration white paper. Disponível em: <http://www.rational.com/media/whitepapers/76.pdf> Acesso em: 23 fev. 2003.

RESEARCH-BASED. Web Guidelines: Design Process. Disponível em: <http://usability.gov/guidelines/designprocess.html#four> Acesso em: 23 fev. 2003.

RETTIG, M. Prototyping for tiny fingers. Communications of the ACM, New York, v.37, n.4, p.21-27, Apr. 1994.

SHNEIDER, K. Prototypes as assets, not toys: why and how to extract knowledge from prototypes. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 18., 1996, Berlin. Proceedings....Los Alamitos, CA: IEEE Computer Society Press, 1996. p.522-531.

STEIGERWALD, R.; HUGHES, G.; BERZINS, V. CAPS as a requirements engineering tool. In: CONFERENCE ON TRI-ADA '91: TODAY'S ACCOMPLISHMENTS; TOMORROW'S EXPECTATIONS, 1991, San Jose CA. Proceedings...New York: ACM Press, 1991. p.75-83.

VERNER, J. M.; CERPA, N. The effect of department size on developer atitudes to prototyping. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 1997, Boston. Proceedings...New York: ACM Press , 1997. p.445-455.

VIRZI, R. A.; SOKOLOV, J. L.; KARIS, D. Usability problem identification using both low – and high – fidelity prototypes. In: CHI96, 1996, Vancouver. Eletronics Proceedings**...** New York: ACM Press, 1996. Disponível em: <http://www.acm.org/sigchi/chi96/proceedings/papers/Virzi/RAVtext.htm> Acesso: 15 set. 2002.

WILEDEN, J. C.; CLARKE, L.; WOLF, A. L. A comparative evaluation of object definition technique for large prototype systems. ACM Transactions and Programming Languages and Systems, New York, v.12, n.4, p.670-699, Oct. 1990.