UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM
CIÊNCIA DA COMPUTAÇÃO

Wilton Cardoso de Souza

Uma Abordagem de Sistema

para

Auxílio a Orientadores

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Mestre em Ciência da Computação.

Prof. Antonio Cezar Bornia, Dr.

Florianópolis, novembro de 2003.

Uma Abordagem de Sistema para Auxílio a Orientadores

Wilton Cardoso de Souza

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação, Área de Concentração Sistemas de Computação e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

_____

Prof. Raul Sidnei Waslawick, Dr.
Coordenador do Curso

Banca Examinadora:

_____

Prof. Antonio Cezar Bornia, Dr.
Presidente da Banca (Orientador)

_____

Prof. Mario Antonio Ribeiro Dantas, PhD.

_____

Prof. Vitório Bruno Mazzola, Dr.

Thomas Edison fez duas mil experiências para conseguir inventar a lâmpada. Um jovem repórter perguntou o que ele achava de tantos fracassos. Edison respondeu: "Não fracassei nenhuma vez. Inventei a lâmpada. Acontece que foi um processo de 2000 passos."

Por isso não devemos achar nunca que NOSSO TEMPO acabou. Enquanto estivermos aqui, há algo para aprendermos e, muito possivelmente, alguém para aprender conosco também.

AGRADECIMENTOS

A Deus, por sua presença constante e protetora, a fé que possuo nas coisas que acredito, e creio que estará sempre presente na minha vida.

Aos meus pais, que são a razão de minha existência, sem eles acredito que não estaria hoje onde estou, a minha querida e amada companheira que sempre soube entender mesmo nos momentos de falta e distância.

Aos amigos que fiz e/ou que se fizeram presentes durante o curso e que, de alguma forma, tornaram este período da minha vida bem menos conturbado através de sua presença alegre, e momentos memoráveis.

Aos professores que tive durante o curso e que de alguma forma me incentivaram, em especial, aos participantes da banca e ao professor, orientador e amigo Antonio Cezar Bornia pelas ajudas prestadas e apoio em momentos difíceis e pelo incentivo para a realização deste trabalho.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE QUADROS

RESUMO

Souza, Wilton Cardoso. **Uma Abordagem de Sistema para Auxílio a Orientadores,** 2003. Dissertação (Mestrado em Ciência da Computação) – UFSC, Florianópolis, SC.

O presente trabalho faz uso das técnicas de engenharia de software e mapeamento de processos para projetar e implementar uma ferramenta computacional de auxílio a orientadores. Tendo como objetivo, além de desenvolver o protótipo da ferramenta, integrar a parte de Gerência de Processos, mais precisamente mapeamento de processos as técnicas de desenvolvimento de softwares, bem como um estudo no processo de orientação. Com este interesse, são realizados estudos nas principais técnicas de desenvolvimento de software e implementação de banco de dados voltadas ao desenvolvimento de um protótipo que auxilie os orientadores em suas funções e lhes permita, prestar e ter uma melhora contínua em suas atividades. Finalmente, é desenvolvido o protótipo, no qual são aplicados os conceitos estudados, tanto em nível de gerenciamento de informação, quanto dos conceitos computacionais abordados. E por fim faz-se uma especificação de um modelo para acesso a base de dados via *Web*, permitindo a troca de mensagens e arquivos entre orientador e orientando sem o uso do correio eletrônico convencional e sim, através de um meio privativo, que poderá ser acessado de qualquer lugar que possua uma conexão de internet.

Palavras-chave: Desenvolvimento de Software, Gerência de Projeto e Mapeamento de Processos.

ABSTRACT

Souza, Wilton Cardoso. **An Approach for a Supervising Assisting Tool,** 2003. Dissertation (Masters in Computing Science) – UFSC, Florianópolis, SC.

The present study employs software engineering and process mapping techniques to design and implement a computing tool to assist supervisors. Apart from developing the prototype for this tool, the study aimed at integrating the Process Management component, and more specifically, process mapping in software development techniques, as well as a study on the supervising process. In order to do so, studies are undergone on the most important software development and database implementation techniques designed for the development of a prototype that can be used as a tool to help supervisors perform their roles, enabling them to improve their performances continuously. After that, the prototype is developed and the concepts studied are applied to it, at an information management level and using the computing concepts approached. Finally a model for accessing database via Web is specified, allowing message and file exchanges between supervisor and student, using not a common electronic mail, but a private media that can be accessed at any location where Internet connection is available.

Key words: Software Development. Project Management and Process Mapping.

1. INTRODUÇÃO

1.1 CONTEXTUALIZAÇÃO

Nas mais diversas áreas de atuação do homem na sociedade atual visualiza-se uma tendência muito forte: o uso do computador como ferramenta de apoio ao desenvolvimento de atividades cotidianas. Esta tendência estende-se por todos os campos do conhecimento humano, devido principalmente, aos avanços em termos de tecnologias de informação, no que tange aos dispositivos físicos e lógicos.

Desta forma este trabalho vem de encontro à afirmativa a cima, pois hoje em dia, os sistemas de *software* estão em todos os lugares. E as atividades de professor e orientação estão diretamente ligadas ao uso de dispositivos de controle eletrônico e não mais a anotações em papéis ou em aplicativos que não possibilitem um acesso rápido, centralizado e eficiente a tais informações. Apesar de muitos ainda não usarem os recursos que esta tecnologia pode proporcionar, seja por não se adaptarem a tais, ou por não possuírem ferramentas apropriadas para desempenhar tal função.

Para o desenvolvimento de sistemas que venham proporcionar algum tipo de controle eletrônico, faz-se uso de técnicas de desenvolvimento, projeto e gerência, técnicas estas que são abordadas dentro da engenharia de *software*. Segundo Pressman (1995) a engenharia de *software,* abrange um conjunto de três elementos fundamentais que possibilitam o controle do processo de desenvolvimento de *software* por parte do gerente e oferece uma base para a construção de *softwares* de alta qualidade. Estes elementos são:

- *Métodos,* que proporcionam os detalhes de como construir *software*. Os métodos envolvem um amplo conjunto de tarefas que incluem: planejamento e estimativa de projeto, análise de requisitos e de sistemas, projeto da estrutura dos dados, arquiteturas de programas, codificação e testes.
- *Ferramentas*, que proporcionam apoio automatizado aos métodos. Atualmente existem ferramentas para sustentar cada um dos métodos anotados anteriormente. Um exemplo é as ferramentas *CASE* (*Computer-Aided Software Engineering*).
- *Procedimentos,* servem de elo para manter juntos os métodos e as ferramentas, e proporcionar o desenvolvimento de *softwares*. Dentre estes procedimentos encontra-se os marcos de referências, planejamento, dentre outros.

De acordo com Sommerville (2003), a engenharia de *software* foi desenvolvida em resposta a construção de sistemas de *software*, pois mesmo os sistemas mais simples de

software têm uma alta complexidade inerente e, conseqüentemente, os princípios da engenharia devem ser empregados em seu desenvolvimento.

1.2 APRESENTAÇÃO DO PROBLEMA

Apesar de quase todas as pessoas e organizações hoje em dia possuírem computadores, a maioria não se utiliza dos benefícios da informatização, devido à inadequação dos recursos de *software*, seja por não terem sido projetados para a devida função ou por não executarem ao que se propõem de forma clara e objetiva. Dentro das universidades o mesmo problema também se mostra visível, pois segundo as pessoas que foram entrevistadas todas demonstraram esta carência.

Da mesma forma que há esta carência em aplicativos para o controle do processo de orientação, há também um mercado de trabalho cada vez mais exigente, que requer a cada dia, mais profissionais com formação em nível de mestrado, doutorado e especialização. Assim como existe também uma discussão acerca de limitar o número de alunos que um orientador deva orientar em um curso de Pós-Graduação. Com base nestas informações e com a possibilidade de se usar o computador como uma ferramenta de apoio à orientação e desta forma prover ao professor uma facilidade maior no controle deste processo, dando-lhe assim, mais tempo para exercer suas atividades e até quem sabe, com o tempo, proporcionar um aumento no número de seus orientados e assim, possibilitar que um número maior de pessoas tenha acesso a uma qualificação. Seja em nível de especialização, mestrado ou doutorado, suprindo desta forma as exigências que o mercado de trabalho impõe hoje em dia. E ainda, possibilitar a formação e o incentivo de novos pesquisadores.

Desta forma, a proposta/problema deste trabalho é disponibilizar o protótipo de uma ferramenta voltado às necessidades deste grupo, que venha auxiliá-los no gerenciamento destas informações e contribuir para uma melhor qualidade de suas funções.

1.3 OBJETIVOS

1.3.1 OBJETIVO GERAL

O objetivo geral deste trabalho é o desenvolvimento de um protótipo genérico, que venha auxiliar os orientadores, no controle do processo de orientação de seus orientados e respectivamente especificar um ambiente *Web* que suporte a troca de informações entre os mesmos, sem o uso do correio eletrônico convencional.

1.3.2 OBJETIVOS ESPECÍFICOS

Dentre os objetivos específicos encontram-se:

- Definir um paradigma de desenvolvimento de *software* para viabilizar a implementação do projeto;
- Integrar mapeamento de processos as técnicas de desenvolvimento de *software*;
- Implementar o protótipo do aplicativo proposto;
- Planejar um modelo para um ambiente *Web*.

1.4 JUSTIFICATIVA

Apesar de toda a evolução na área tecnológica e no processo e desenvolvimento de *softwares*, os desenvolvedores ainda entregam projetos fora do prazo e que não refletem a real necessidade do usuário em termos de função, qualidade e custos. Com base nestas necessidades é que se propõe o desenvolvimento de um protótipo, mesmo que genérico, mas que venha a satisfazer estas exigências da melhor forma possível. Além disso, outro fator foi o de não ter sido encontrado nas pesquisas realizadas até o momento, nenhum tipo de aplicativo com as referidas funções e propósito, os trabalhos correlatos que foram encontrados tratavam apenas de orientação educacional na escola e sem um aplicativo para tal. O trabalho que mais aproximou-se do que é proposto aqui, propunha uma ferramenta para assessoramento a estudantes na análise de currículos em cursos de graduação, cujo título é *The Advisor's Assistant*, ressalta-se ainda, que as buscas efetuadas foram na procura por pesquisas e aplicativos que tivessem o mesmo propósito do que é proposto neste trabalho. As pesquisas se deram nos seguintes locais *Softwaretechnic* – Universidade de Berlim; *ACM* – Grupo de Engenharia de *Software*; *UNICAMP*; *USP*; *UFSC*; *UFRGS*; *UFSM*; *PUC* – Rio; IBICT – Biblioteca Digital de Teses e Dissertações; *RICTEC – CAPES*; *ISI – Institute for Scientific Information*. Os endereços das procuras são citados na seção 1.5.2. Desta forma, justifica-se este trabalho pela carência de uma ferramenta computacional que venha dar suporte aos orientadores em suas atividades diárias, de forma otimizada e direcionada a este grupo.

Sendo assim, as pesquisas feitas para o desenvolvimento deste trabalho visam, além de propor a implementação de um protótipo, fazer um estudo bibliográfico na parte de engenharia de *software* e mapeamento de processos, para conseguir-se desenvolver o referido projeto com a melhor qualidade possível. É dada uma ênfase maior à parte de processo de

desenvolvimento e gerenciamento de *software*, por serem consideradas as partes mais importantes para a concepção de um produto de *software* de qualidade, pois o sucesso do desenvolvimento de *softwares* está diretamente relacionado ao gerenciamento geral do projeto.

Outra questão abordada é a integração de métrica de *software* ao planejamento/gerenciamento de projetos, como forma de viabilizar informações consistentes para a tomada de decisão pertinente ao gerenciamento de projeto.

Um benefício evidente desta integração é citado por Pressman (1995), o qual afirma que, sem informações quantitativas a respeito de todo o processo de desenvolvimento de *softwares* por parte de uma empresa ou equipe de desenvolvedores de produto é impossível tirar qualquer conclusão sobre de que forma está evoluindo a produtividade. Assim, através da obtenção de medidas relativas à produtividade e à qualidade, é possível que metas de melhorias no processo de desenvolvimento sejam estabelecidas. Pois, a avaliação quantitativa do desenvolvimento viabiliza promover pequenos ajustes no processo de desenvolvimento, como forma de eliminar ou reduzir as causas de problemas que afetam de forma significativa o projeto de um aplicativo de *software*.

1.5 METODOLOGIA

1.5.1 CARACTERIZAÇÃO DA PESQUISA

A organização de um processo de *software* é uma atividade crítica que engloba o gerenciamento de diversos aspectos, tais como: tarefas, recursos e tempo, bem como o gerenciamento de todos os produtos produzidos durante o ciclo de vida do sistema. A organização do processo também envolve a definição de métodos e ferramentas dentro de metodologias a serem aplicadas no desenvolvimento e gerenciamento de projetos de *software*.

De acordo com Carvalho e Chiossi (2001), os métodos são linhas gerais que governam a execução de alguma atividade e as ferramentas dão suporte à aplicação de métodos e metodologias. Os autores ainda citam que uma metodologia de desenvolvimento detalha as atividades do ciclo de vida, especificando um conjunto único e coerente de princípios, métodos, linguagem de representação, normas, procedimentos e documentação, que permitem ao desenvolvedor de *software* implementar sem ambigüidade as especificações advindas das fases do ciclo de vida do *software*.

Assim, esta seção apresenta uma metodologia utilizando e/ou adaptando métodos já existentes na literatura para o desenvolvimento de projetos de *software*, caracterizando esta

parte como sendo uma pesquisa de ordem bibliográfica e contemplando também com a especificação de um aplicativo de apoio a orientadores. No capítulo 4 é apresentado o protótipo do aplicativo que suporta a metodologia e os métodos usados.

Para o desenvolvimento deste trabalho, foi realizada também uma pesquisa aplicada, objetivando-se gerar um conhecimento para aplicação prática da solução do problema levantado e da mesma forma ocorreu em paralelo uma pesquisa de levantamento, que envolveu a interrogação direta das pessoas envolvidas no processo (SILVA e MENEZES, 2001).

Sendo assim, esta seção definiu a caracterização da pesquisa do trabalho que está sendo proposto. A seguir são abordados os procedimentos que foram elaborados para o desenvolvimento da pesquisa.

1.5.2 PROCEDIMENTOS

Um dos procedimentos utilizado neste trabalho foi a técnica de *Brainstorming*. Primeiramente houve um contato com o orientador e através de conversas e reuniões, elaborou-se uma Lista de Verificação, a qual serviu como base para se iniciar as entrevistas com outros orientadores, para as quais se utilizou a técnica citada. Tomou-se o cuidado de selecionar para as entrevistas pessoas de diferentes áreas, pois só desta maneira seria possível sentir a real necessidade expressa por todos e assim mapear e elaborar um aplicativo genérico, que venha servir e facilitar o uso de um número maior de orientadores, da melhor forma possível.

Técnica Brainstorming

É uma técnica utilizada por grupos para geração de um grande número de idéias. Tenta-se evitar julgamentos, avaliações ou críticas dessas idéias no momento em que estão sendo geradas. Seu princípio básico é o de que o grande número de idéias produzidas aumente a probabilidade de se encontrar uma solução melhor para o problema. Seu procedimento básico é o seguinte:

- O problema é apresentado/descrito geralmente como uma questão aos participantes do grupo;
- É providenciada a privacidade dos participantes (separação física ou de outra forma);
- Os participantes elaboram e escrevem as suas idéias (em aproximadamente dez minutos);

- Apresentação das idéias ao grupo;
- Avaliação das alternativas, com o uso de uma escala de zero (pouca contribuição) a quatro (máxima contribuição para a solução do problema).

Etapas Seguidas Antes do Levantamento dos Dados

Tendo como objetivo um melhor resultado da pesquisa, planejou-se algumas tarefas, sendo que as mais importantes são citadas a seguir:

- *Busca por trabalhos correlatos*: esta fase se deu desde o início da pesquisa, como é relatado no item 1.4. Até o momento não encontrou-se nenhum trabalho que tenha desenvolvido uma aplicação como a que está sendo proposta neste trabalho. Nesta fase houve uma pesquisa sobre trabalhos correlatos que tenham usado gerência de processos ou mapeamento de processos junto com técnicas de análise de dados para desenvolver aplicativos de *software*. Nesta fase de procura, foram encontrados muitos trabalhos que usavam gerência de processos, mas somente três tinham relevância para o que é proposto neste trabalho. Os trabalhos são: Um Modelo de Gestão da Informação Digital em Bibliotecas Acadêmicas na Educação à Distância; Uma Ferramenta Computacional para Gestão por Processos e Modelagem de Sistemas de Informações para o Gerenciamento Integrado de Cadeias Logísticas: Uma Demonstração das Possibilidades de Aplicação na Indústria de Petróleo.
- *Elaboração da Lista de Verificação*: teve-se a idéia de criar a lista de verificação, para dar-se inicio as entrevistas a partir de uma pré-definição de proposta. Pois desta forma ficou mais fácil para os orientadores entenderem a proposta do trabalho e assim, responder a lista e demonstrar o que necessitavam de funções na ferramenta. A partir da lista os orientadores foram colocando suas necessidades e as mesmas foram discutidas e planejadas. Sendo visto a melhor forma de colocá-las na ferramenta.
- *Mapeamento das entrevistas*: resolveu-se escolher um grupo de pessoas de várias áreas dentro da UFSC, para desta forma verificar o que cada área necessitava mais e qual a forma que usam para desenvolver o processo de orientação. Primeiro foi explanado aos orientadores a proposta do trabalho, a seguir usou-se a lista de verificação para começar a entrevista e coletar as sugestões dos entrevistados.
- *Definição das necessidades do sistema*: esta fase só foi concluída após duas reuniões com cada um dos orientadores entrevistados, mesmo assim, houve várias alterações após a definição, sendo que teve-se que refazer algumas das entrevistas.

- *Aplicação piloto*: após as entrevistas fez-se o mapeamento do processo, a definição e modelagem dos campos essenciais que iriam compor o banco de dados do sistema. Então, elaborou-se a primeira versão do protótipo contendo os campos essenciais para o referido aplicativo.
- *Locais das Pesquisas*: http://isiknowledge.com/; http://www.bu.ufsc.br; http://www.eps.ufsc.br; http://rictec.capes.gov.br/login.asp; http://www.teses.usp.br/; http://swt.cs.tu-berlin.de/swt_e.html; http://www.acm.org/sigsoft/publications.htm; http://www.rau-tu.unicamp.br/nou-rau/sbu; http://www.ibict.br/bdb/portal/bdb_main_consorcio.php; http://www.unicamp.br/anuario/2002/IC/IC-dissertacoesmestrado.html; http://www.biblioteca.ufrgs.br/bibliotecadigital/; http://www.maxwell.lambda.ele.puc-rio.br/cgi-bin/db2www/PRG_0452.D2W/INPUT;

Este item abordou algumas das tarefas que envolveram a definição da metodologia a ser usada para a coleta dos dados e implementar o referido protótipo, na seção seguinte será explanado sobre o levantamento dos dados com as pessoas entrevistadas.

1.5.3 LEVANTAMENTO DE DADOS

Segundo Rezende (1999), o levantamento de dados é uma tarefa que está presente em praticamente todos projetos de *SW*. O sucesso de um projeto depende fundamentalmente, do levantamento de dados, pois é através dele que o pesquisador poderá compreender os requisitos do *software*. Porém conforme o autor, esta atividade tão importante é elaborada de forma intuitiva e sem metodologia.

Dentre as técnicas sugeridas por Rezende (1999) para a coleta dos dados deste projeto, adotou-se a técnica de questionário e entrevistas simultaneamente. Pois como citado anteriormente, elaborou-se um questionário para definir algumas das necessidades básicas dos entrevistados e juntamente a este questionário deu-se início à entrevista para saber a real necessidade do grupo.

1.5.3.1 ENTREVISTAS COM ORIENTADORES

Este procedimento teve como objetivo principal à obtenção de informações sobre o processo de orientação como um todo, em especial as informações relacionadas com os processos que serão necessários para a implementação do protótipo.

As entrevistas foram realizadas com profissionais ligados ao meio acadêmico, pois estes têm condições de relatar o funcionamento da orientação com precisão e riqueza de

detalhes, para a compreensão preliminar do processo estudado. Estas pessoas são chamadas de facilitadoras da pesquisa, pois podem relatar todas as etapas com o intuito de facilitar a realização do trabalho.

Antes de iniciar a entrevista, fez-se uma breve apresentação sobre o trabalho a desenvolver e ressaltou-se a importância das informações prestadas em todas as etapas da pesquisa, evidenciado-se a relevância da participação do entrevistado. Esta atitude resultou em retorno direto para se atingir os objetivos do trabalho, pois, entende-se que uma vez que o entrevistado esteja ciente de sua importância, sentir-se-á motivado a participar da pesquisa e prestar as informações necessárias.

Foram contatadas pessoas de cursos variados dentro da UFSC – Universidade Federal de Santa Catarina, tais como: EPS – Engenharia da Produção e Sistemas, CPGA – Curso de Pós Graduação em Administração, CSE – Centro Sócio Econômico, PGCC – Pós Graduação em Ciência da Computação. Os orientadores entrevistados foram: Leonardo Ensslin, Ilse Maria Beuren, Antônio Diomário de Queiroz, Holf Hermann Erdmann, Mário Dantas e Vitório Bruno Mazzola.

Desta forma, após ter estudado, analisado e mapeado o processo de orientação, assunto que é abordado no capítulo 3, definido o ponto crítico, que é a inexistência de um aplicativo que concentre o controle das atividades do aluno e que possa ao mesmo tempo proporcionar ao orientador, acompanhar e consultar esta base quando desejar em um único aplicativo, isto é, sem ter que usar outros aplicativos que não são próprios para este fim. Como o editor de textos *Word*, planilhas, dentre outros, é que se deu início a implementação do protótipo que supra estas necessidades e venha a contribuir e facilitar o controle do processo de orientação. E, futuramente, eliminar a troca de mensagens e arquivos através do correio eletrônico convencional, e sim, fazendo uso de um correio privado que poderá ser acessado de qualquer lugar que possua uma conexão com a *Web*.

1.6 LIMITAÇÕES DA PESQUISA

Destaca-se como fator de limitação desta pesquisa o fato da mesma não contemplar a implementação da parte de *Web*, restringindo-se apenas a especificação da mesma, em função das restrições existentes de prazo e complexidade da mesma. Outro ponto a ser citado é o fato de se efetuar a implementação de um protótipo e não de um aplicativo em definitivo.

O estudo feito em cima do mapeamento do processo de orientação foi exclusivamente para se entender o mesmo e para poder-se definir os módulos da ferramenta, sendo assim, não abordou-se neste trabalho um detalhamento do mesmo.

As técnicas de busca implementadas na ferramenta suportam apenas buscas no banco de dados em que o usuário estiver logado, não sendo implementado buscas em bancos separados.

Uma outra questão que não foi abordada e nem projetada para o modelo *Web* foi o uso de algum mecanismo para a tradução do material científico que poderá ser disponibilizado pelo mesmo, já que se terá uma base bibliográfica com material em vários idiomas.

1.7 ESTRUTURA DO TRABALHO

Este trabalho está estruturado da seguinte forma: o Capítulo 1 é um capítulo introdutório, onde são apresentados a contextualização, o problema, os objetivos do trabalho, metodologia, sua justificativa e limitações.

O Capítulo 2 versa sobre a fundamentação teórica, abordando os assuntos sobre processo, mapeamento de processos e engenharia de *software*, com ênfase nos assuntos relativos a desenvolvimento e gerenciamento de *software*. É também abordado neste capítulo, assuntos relativos a interface homem-computador, qualidade de *software*, teste de *software* e banco de dados.

O Capítulo 3 trata da descrição do planejamento do projeto, da coleta dos dados, da descrição do mapeamento do processo de orientação. Apresenta ainda as ferramentas usadas para desenvolver o sistema e faz-se uma especificação do sistema.

No Capítulo 4 é apresentado o projeto lógico do sistema proposto, bem como suas especificações e algumas telas do sistema implementado. As conclusões e trabalhos futuros são apresentados no Capítulo 5.

2. FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta a revisão da literatura, sendo abordado no mesmo os conceitos de processos, desenvolvimento e gerência de *software*. Questões de fundamental importância para o desenvolvimento de projetos. Considerando os objetivos do projeto proposto para este trabalho, são também citados neste capítulo, questões referentes à *interface* homem-computador, qualidade de *software*, testes de *software*, manutenção de *software* e banco de dados.

2.1 PROCESSOS

2.1.1 DEFINIÇÃO

Objetivando alinhar as conceituações de processos, algumas definições serão recorridas com este fim.

Para Davenport (1994), um processo é uma ordenação específica das atividades de trabalho no tempo e no espaço, com começo, fim, *inputs* e *outputs* claramente identificados. Enfim é uma estrutura para ação.

Johansson et al. (1995) definem processo como um conjunto de atividades ligadas, que tomam um insumo *(input*) e o transformam para criar um resultado *(output*). Teoricamente, a transformação que nele ocorre deve adicionar valor e criar um resultado que seja mais útil e eficaz ao recebedor acima ou abaixo da cadeia produtiva.

Rummler e Brache (1994) afirmam ser uma série de etapas criadas para produzir um bem ou serviço, incluindo várias funções e abrangendo o "*espaço em branco*" entre os quadros do organograma, sendo visto como uma "*cadeia de agregação de valores*".

Assim, um processo dispõe de *input*s, *output*s, tempo, espaço, ordenação, objetivos e valores que, interligados logicamente, irão resultar em uma estrutura para fornecer bens ou serviços ao cliente. Sua compreensão é importante, pois é a chave para o sucesso em qualquer negócio. Afinal, uma organização é tão efetiva quanto os seus processos, pois eles são responsáveis pelo que será ofertado (JOHANSSON et al., 1995; RUMMLER e BRACHE, 1994).

Segundo Harrington (1993 *apud* Souza, 2002) processo é definido como qualquer atividade que recebe uma entrada (*input*), agrega-lhe valor e gera uma saída (*output*) para um cliente interno ou externo. Complementando, o autor registra que recursos de toda ordem são alocados na organização através dos processos, objetivando gerar resultados concretos.

Desta forma, pode-se concluir que o valor agregado é a diferença do valor da saída com o da entrada. A figura 2 ilustra esta definição, identificando os seus elementos principais.

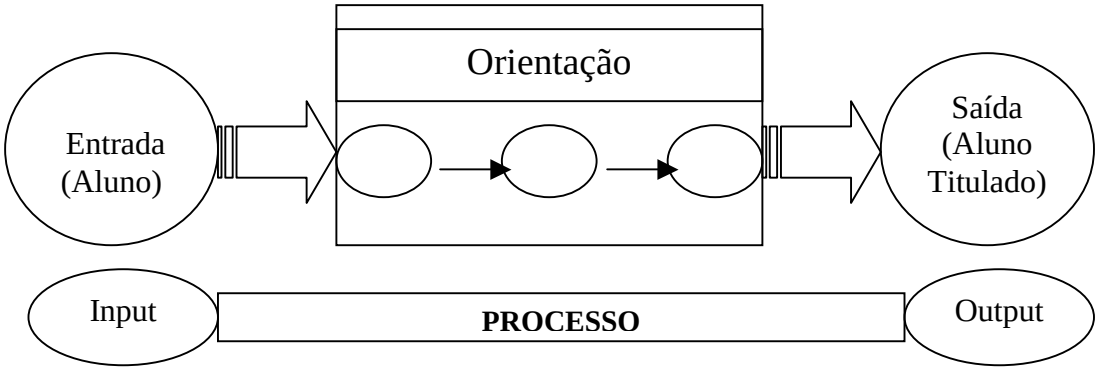


Figura 1 [Adaptado de Harrington (1993)]: Elementos Genéricos de um Processo

Segundo Zahran (1998), processo é uma seqüência de passos executados para uma dada proposta, o autor cita ainda como exemplo, o processo de desenvolvimento de *software*. Pois é definido também como um conjunto de atividades e recursos inter-relacionados que transformam entradas em saídas.

Do ponto de vista macro, Harrington (1993) comenta que os processos são necessários para administrar e/ou operar uma organização. Desta forma, a identificação dos principais processos de uma organização deve responder às perguntas: "*O que se faz como organização*?" e "*Como se faz isto*?"

2.1.2 HIERARQUIA DOS PROCESSOS

Antes de identificar as características comuns dos processos, é necessário saber que eles compõem a estrutura organizacional através de uma hierarquia, onde é representado o nível de detalhamento com que o trabalho está sendo abordado.

Esta hierarquia é assim apresentada: (HARRINGTON, 1993; DAVIS e WECKLER, 1997).

- *Macroprocesso*: é um processo que geralmente envolve mais que uma função na estrutura organizacional, e a sua operação tem um impacto significativo no modo como a organização funciona;
- *Processo*: é um conjunto de atividades seqüenciais (conectadas), relacionadas e lógicas que tomam um *input* com um fornecedor, acrescentam valor a este e produzem um *output* para um consumidor;

- *Subprocesso*: é a parte que, inter-relacionada de forma lógica com outro subprocesso, realiza um objetivo específico em apoio ao macroprocesso e contribui para a missão deste;
- *Atividades*: são coisas que ocorrem dentro do processo ou subprocesso. São geralmente desempenhadas por uma unidade (pessoa ou departamento) para produzir um resultado particular. Elas constituem a maior parte dos fluxogramas;
- *Tarefa*: é uma parte específica do trabalho, ou melhor, o menor microenfoque do processo, podendo ser um único elemento e/ou um subconjunto de uma atividade. Geralmente, está relacionada a como um item desempenha uma ação específica.

2.1.3 MAPEAMENTO DE PROCESSOS

Para comparar a situação atual e a desejada, torna-se necessário mapear o processo como ele é, identificando qual é o problema do processo para modelar como ele deverá ser, para apresentar um mapa de "*Com*o" o problema será resolvido ou da implantação do novo processo (HUNT, 1996).

Desta maneira, a realização de uma mudança organizacional significativa necessita de um profundo conhecimento das atividades que constituem os processos essenciais da orientação e os subprocessos que os apóiam, em termos de sua finalidade, pontos de início, entradas, saídas e influências limitadoras. Este entendimento pode ser melhor alcançado pelo "mapeamento", "modelagem" e medida dos processos (JOHANSSON et al., 1995).

O mapeamento de processos é uma ferramenta gerencial analítica e de comunicação, que tem a intenção de ajudar a melhorar os processos existentes ou de implantar uma nova estrutura voltada para processos. A sua análise estruturada permite ainda, a redução de custos no desenvolvimento de bens e serviços, a redução nas falhas de integração entre sistemas e melhora de desempenho, além de ser uma excelente ferramenta para possibilitar o melhor entendimento dos processos atuais e eliminar ou simplificar aqueles que necessitam de mudanças (HUNT, 1996).

Segundo Hunt (1996), este mapeamento foi desenvolvido e implementado pela General Eletric como parte integrante das estratégias de melhoria significativa do desempenho, onde era utilizado para descrever, em fluxogramas e textos de apoio, cada passo vital dos seus processos. Porém, o mapeamento de processo teve suas origens em uma variedade de áreas, sendo que, a origem da maioria das técnicas como o diagrama de fluxo, o diagrama de cadeia, o diagrama de movimento, os registros fotográficos, os gráficos de

atividades múltiplas e os gráficos de processo podem ser atribuídos a Taylor e a seus estudos de melhores métodos de se realizar tarefas e organização racional do trabalho na *Midvale Steel Works* (JOHANSSON et al, 1995).

Para Johansson et al. (1995), o mapeamento de processo pode ser suplementado por uma técnica chamada modelagem de dados, a qual evoluiu do reconhecimento crescente da necessidade crítica de administrar dados complexos e muito distribuídos como um ativo na criação de processos de negócios radicalmente novos. Assim, são localizadas as eficiências na obtenção, domínio e disseminação dos dados, para que se evite duplicação e sobreposição desnecessárias e se mantenha o valor do dado como um ativo.

Segundo Pidd (1998), faz sentido modelar o processo para descobrir os componentes essenciais e sensíveis em que as melhorias farão diferença, já que as mudanças tecnológicas permitem que o processo seja mudado no espaço ou no tempo, capacitando a organização a operar mudanças rapidamente auxiliadas por modelos simulados em computador e pela engenharia dos processos.

Desta forma, as duas técnicas não devem ser confundidas, a modelagem de dados não é um substituto para o mapeamento de processo. Na modelagem de dados, a meta é entender as relações entre os dados elementares e as ligações entre os conjuntos de dados onde aqueles podem estar presentes, enquanto que o mapeamento de processos busca entender os processos existentes e futuros, para criar melhor satisfação do cliente e melhor desempenho.

Uma grande quantidade de aprendizado e melhoria nos processos pode resultar da documentação e exame dos relacionamentos *input-output* representados em um mapa de processos. Afinal, a realização deste mapa possibilita a identificação das *interfaces* críticas, a definição de oportunidades para simulações de processos, a implantação de métodos de contabilidade baseados em atividades e a identificação de pontos desconexos ou ilógicos nos processos. Desta forma, o mapeamento desempenha o papel essencial de desafiar os processos existentes, ajudando a formular uma variedade de perguntas críticas, como por exemplo: *Esta complexidade é necessária? São possíveis simplificações? Existe excesso de transferências interdepartamentais? As pessoas estão preparadas para as suas funções? O processo é eficaz? O trabalho é eficiente? Os custos são adequados?* (HUNT, 1996; JOHANSSON et al., 1995)

Em um mapa de processos, consideram-se atividades, informações e restrições de *interface* de forma simultânea. A sua representação inicia-se do sistema inteiro de processos como uma única unidade modular, que será expandida em diversas outras unidades mais detalhadas, que, conectadas por setas e linhas, serão decompostas em maiores detalhes de

forma sucessiva. Esta decomposição é que garantirá a validade dos mapas finais. Assim sendo, o mapa de processos deve ser apresentado em forma de uma linguagem gráfica que permita (HUNT, 1996):

- Expor os detalhes do processo de modo gradual e controlado;
- Encorajar concisão e precisão na descrição do processo;
- Focar a atenção nas *interfaces* do mapa do processo;
- Fornecer uma análise de processos poderosa e consistente com o vocabulário do *design*.

Esta linguagem gráfica necessária ao mapeamento de processos encontra-se em uma variedade de ferramentas de análise disponíveis para auxiliar o analista de processo.

Estas ferramentas foram desenvolvidas durante um longo tempo, ocorrendo a adequação entre grupos de ferramentas e metodologias de mudança e reestruturação de processos, sendo que houve um desenvolvimento paralelo e mais rápido das ferramentas baseadas em computador (JOHANSSON et al., 1995).

2.1.4 MAPEAMENTO DE PROCESSOS COMPUTADORIZADO

Com a entrada dos computadores, foram necessárias técnicas para traduzir necessidades funcionais em um processo adequado para ser codificado sob forma de instruções de computadores. Assim, foram estabelecidas convenções para a criação e o uso destes novos fluxogramas. Uma geração completa de analistas de sistemas tornou-se adepta do uso de gabaritos de fluxogramas, avançando com o crescimento dos bancos de dados, onde o mapeamento de processo apóia o gerenciamento de sistemas de integração de dados e a construção destes bancos (JOHANSSON et al., 1995).

Esta seção abordou a parte de mapeamento de processos e suas definições, os assuntos referentes a desenvolvimento de *software* são abordados na seção seguinte.

2.2 DESENVOLVIMENTO DE *SOFTWARE*

Esta seção dá ênfase à definição de processo, modelos de projeto de desenvolvimento e gerência de *software*, questões consideradas fundamentais para a engenharia de *software* no desenvolvimento de sistemas de *softwares*.

2.2.1 PROCESSO DE *SOFTWARE*

Processo de *software*, segundo Zahran (1998), é um conjunto de atividades que se deve cumprir numa ordem estabelecida, para desenvolver e manter *software* e produtos a ele associados, como: planos de projetos, documentos de projeto, código-fonte, casos de testes, dentre outros. Os processos técnicos e gerenciais relacionados ao desenvolvimento de *software* constituem o enfoque em processos de *softwares*, cujo objetivo é planejar, monitorar, gerenciar, executar, controlar e melhorar atividades relacionadas a *software*.

Jalote (1997, p. 23), apresenta o conceito de processo como sendo o coração da engenharia de *software*. E conclui que um processo de *software* é:

> um conjunto de atividades, ligadas por padrões de relacionamento entre elas, pelas quais se as atividades operarem corretamente e de acordo com os padrões requeridos, o resultado desejado é produzido. O resultado desejado é um *software* de alta qualidade a baixo custo. Obviamente, um processo que não aumenta a produção (não suporta projetos de *software* grandes) ou não pode produzir *software* com boa qualidade não é um processo adequado.

No desenvolvimento de *softwares*, é requerido a execução de várias atividades. Em um projeto de desenvolvimento de *software*, duas atividades no mínimo devem estar presentes: a atividade de desenvolvimento e a de gerenciamento de projeto. A soma destas atividades irá compor o processo de *software*.

Sommerville (2003) alude que, um processo de *software* é um conjunto de atividades e resultados associados que irão gerar um produto de *software*. Citando ainda que há quatro atividades importantes comuns aos processos de *softwares*, que são: especificação do *software*, desenvolvimento do *software*, validação do *software* e evolução do *software*. Esse processo pode envolver o desenvolvimento de *software* desde o início, embora, cada vez mais, ocorra casos de um *software* novo ser desenvolvido mediante a expansão e a modificação de sistemas já existentes.

O processo de *software* pode ser visto como um gerador de produtos, sendo que o produto final, ou principal é o *software* em si. É importante perceber que existem subprodutos que são gerados para cada fase. Como exemplo, ao final da fase de especificação, é comum ter sido desenvolvido e entregue um ou mais documentos que detalham os requisitos do sistema. Todo modelo de *software* deve levar em consideração as fases descritas, no entanto cada um organiza estas fases de acordo com sua filosofia de organização.

2.2.2 PROCESSO E PROJETO DE *SOFTWARE*

Um método para o desenvolvimento de *software* é especificado através de um processo de *software*. Um projeto de *software*, de uma outra forma, é um desenvolvimento no

qual o processo de *software* é usado. E os produtos de *software* são os resultados do projeto de *software* (JALOTE, 1997).

A partir de algumas necessidades, é dado início a um projeto de desenvolvimento de *software* e o mesmo é finalizado com a concepção de um produto de *software* que venha satisfazer estas necessidades. Um processo de *software* especifica, o conjunto de atividades que devem ser executadas para a partir das necessidades do usuário se chegar no produto final. Um projeto de *software* é o ato de executar as atividades para suprir alguma necessidade do usuário. E todas as saídas que são produzidas enquanto as atividades estão sendo executadas são os produtos (um dos quais é o produto final).

Jalote (1997) descreve o processo de *software* como um tipo de abstração. Cada projeto é feito usando o processo como uma instância deste tipo. Em outras palavras, muitos projetos podem ser feitos usando um processo, e podem existir muitos produtos gerados a partir de um projeto. Este relacionamento pode ser melhor observado na figura 2.

De acordo com Sommerville (2003), diferentes organizações podem utilizar processos diferentes para produzir o mesmo tipo de produto. No entanto, alguns projetos são mais adequados do que outros para alguns tipos de aplicação. Porém se um processo inadequado for utilizado, isso provavelmente reduzirá a qualidade do produto de *software* que está sendo desenvolvido.

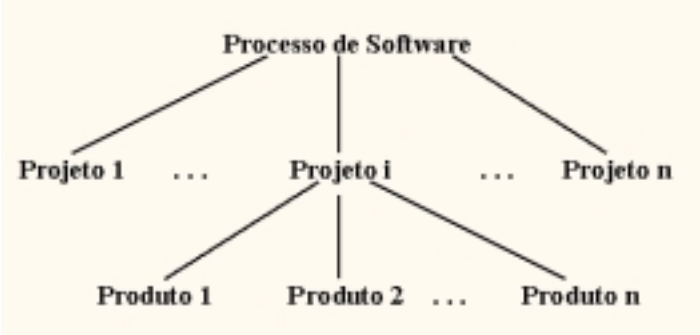


Figura 2 [Jalote (1997)]: Processos, Projetos e Produtos

Esta seção deu ênfase a desenvolvimento de *software*, abordando assuntos referentes a processo, projeto e produto de *software*. O assunto referente aos paradigmas de desenvolvimento de *software* é ênfase da próxima seção.

2.3 PARADIGMAS DE DESENVOLVIMENTO DE *SOFTWARE*

A produção do *software* constitui-se o foco principal no processo de desenvolvimento de *software*. Um modelo de projeto especifica algumas atividades que devem ser executadas e a ordem na qual as mesmas devem ser efetuadas (JALOTE, 1997).

As principais atividades de desenvolvimento e de garantia que precisam ser executadas em um projeto de *software* são especificadas no processo de desenvolvimento, logo, este processo pode ser considerado como o núcleo do processo de *software*. O processo de gerenciamento é decidido com base no processo de desenvolvimento (JALOTE, 1997).

Os paradigmas de desenvolvimento de *software*, de acordo com Sommerville (2003), representam os processos a partir de uma perspectiva particular, de uma maneira que proporcionam apenas informações parciais sobre o processo. Esses paradigmas são abstrações úteis, que podem ser utilizadas para explicar diferentes abordagens do desenvolvimento de *software*, paradigmas diferentes podem ser utilizados para desenvolver diferentes partes do sistema.

Uma série de diferentes paradigmas foi proposto, cada um exibindo potencialidades e fragilidades, mas todos tendo uma série de fases genéricas em comum, sendo assim, o uso dos mesmos se torna importante no desenvolvimento de aplicativos de *software* (PRESSMAN, 1995).

As seções seguintes discutem alguns dos principais paradigmas.

2.3.1 CICLO DE VIDA CLÁSSICO

O ciclo de vida clássico ou modelo cascata foi popularizado em 1970 e é abordado em diversos livros. Dentre os autores pesquisados para a realização deste trabalho, encontra-se o modelo cascata descrito por Von Mayrhauser (1990), Ghezzi (1991), Pressman (1995), Jalote (1997) e Sommerville (2003). A figura 3 retrata o referido modelo.

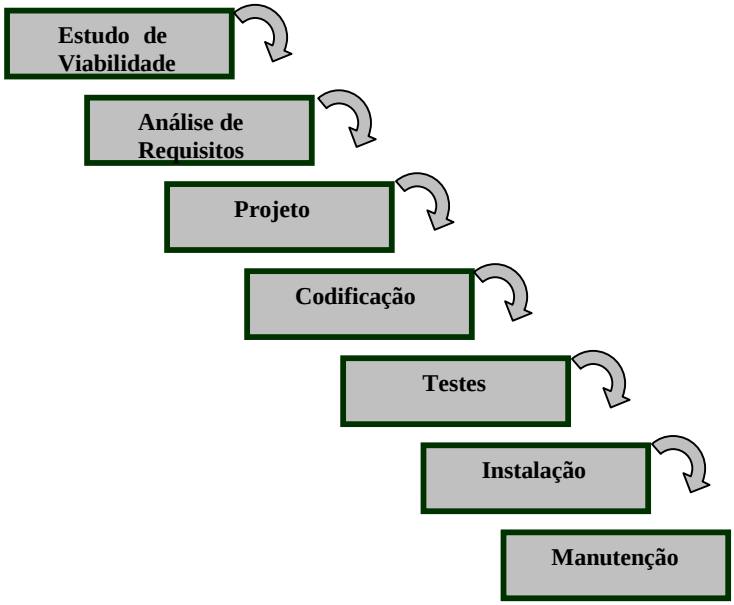


Figura 3 [Ghezzi (1991)]: Modelo Cascata

O modelo é apresentado com algumas variações pelos diferentes autores. No entanto, apesar das variações, o princípio é o mesmo, e o modelo visualizado na figura 3 pode ser considerado um exemplo representativo, bem como as explicações posteriores se aplicam a todas variações.

A seguir é apresentada uma breve explicação das atividades realizadas em cada uma das fases do modelo cascata.

2.3.1.1 FASES

Fase de Estudo de Viabilidade

Segundo Ghezzi (1991), o propósito desta fase é produzir um documento com o estudo de viabilidade dos custos e benefícios da aplicação proposta. Para isto, primeiramente é necessário analisar o problema, pelo menos num nível global.

Obviamente, quanto mais o problema for compreendido, melhor se poderá identificar as alternativas para solução, seus custos, e os seus benefícios para o usuário. Portanto, idealmente, se deveria executar uma análise do problema tão detalhadamente quanto necessário para o estudo de viabilidade. No entanto, o autor destaca que isto raramente ocorre na prática. O estudo é geralmente efetuado com um tempo limitado e sob pressão.

Em suma, o estudo de viabilidade deve antecipar o cenário para o desenvolvimento do *software*, resultando num documento que deve conter, no mínimo os seguintes itens (GHEZZI, 1991):

- Uma definição do problema.
- Alternativas de solução e os benefícios esperados.
- Recursos requeridos, custos e data para entrega de cada alternativa de solução proposta.

Fase de Análise de Requisitos

Esta fase tem como propósito identificar os atributos requeridos pela aplicação, em termos de funcionalidade, performance, facilidade de uso, portabilidade, dentre outros. O responsável por esta fase deve relatar quais atributos a aplicação deve ter e não como obtê-los, ou seja, nesta fase, deve-se definir quais funções o *software* terá (GHEZZI, 1991).

O documento com a especificação dos requisitos possui dois propósitos: um de ser analisado e confirmado com o usuário para verificar se todas as expectativas foram abordadas, e o outro é desenvolver a solução que vá de encontro com os requisitos especificados.

O documento, conforme Ghezzi (1991), pode ser produzido contendo os seguintes itens:

- *Requisitos funcionais*: descrevem o que o produto faz usando uma notação informal, semiformal, formal ou uma mistura delas. Vários livros, como por exemplo, Ghezzi (1991), Von Mayrhauser (1990), Jalote (1997) e Sommerville (2003), apresentam as notações existentes.
- *Requisitos não funcionais*: estes podem ser classificados dentro das seguintes categorias: credibilidade (segurança, integridade, dentre outros), precisão dos resultados, performance, questões de uso da *interface* homem-máquina, portabilidades, dentre outras.
- *Requisitos do processo de desenvolvimento e manutenção*: inclui propriedades de controle de procedimentos em particular, procedimentos para teste, procedimentos para manutenção, dentre outros.

Fase de Projeto

Enquanto a análise de requisitos se detém em o que o *software* a ser implementado fará, a fase de projeto descreve como a solução será implementada. O resultado desta fase é a

arquitetura do *software*: a função de cada módulo e o relacionamento entre os módulos (GHEZZI, 1991; VON MAYRHAUSER, 1990).

Fase de Codificação

O propósito desta fase é traduzir a solução em código. Von Mayrhauser (1990) indica ainda que somente após o código estar todo escrito, documentado e compilado, livre de erros e seguindo o padrão do projeto é que se pode dar esta fase como concluída. A autora cita ainda que até o final desta fase um plano de testes, descrevendo quando e como testar cada parte do código, deve ser traçado.

Fase de Testes

O código gerado deve ser testado rigorosamente baseado nos requisitos analisados, segundo Von Mayrhauser (1990), este é o objetivo desta fase.

Primeiramente os módulos são testados isoladamente, denominados teste de unidade. Posteriormente, os módulos são testados em grupo visando verificar se estão interagindo adequadamente, denominado teste de integração. A partir destes testes executa-se o teste de sistema onde o *software* é testado em ambientes diferentes. O teste de instalação deve testar o *software* em sistemas com configurações diversas. Além destes, ainda é requerido um teste de aceitação, que deve verificar juntamente com o usuário se os requisitos foram atingidos (VON MAYRHAUSER, 1990).

Fase de Implantação

A implantação, geralmente ocorre em dois estágios. Num primeiro momento, a aplicação é distribuída para um grupo selecionado de usuários, para verificar o *feedback* dos mesmos. Num segundo estágio, o produto é distribuído para todos os usuários (GHEZZI, 1991).

Fase de Manutenção

Fase que se inicia a partir da entrega do aplicativo de *software*. É caracterizada pela realização de alterações de diversas naturezas, seja para corrigir erros residuais da fase anterior, para incluir novas funções exigidas pelo cliente, ou para adaptar o *software* a novas configurações de *hardware*.

Sendo assim, Pressman (1995), caracteriza esta fase pelas seguintes atividades:

- *Correção* ou *Manutenção Corretiva*: a qual consiste da atividade de corrigir erros observados durante a operação do sistema;
- *Adaptação ou Manutenção Adaptativa*: a qual realiza alterações no *software* para que ele venha a ser executado sobre um novo ambiente (CPU, arquitetura, novos dispositivos de *hardware*, novo sistema operacional, dentre outros);
- *Melhoramento Funcional* ou *Manutenção Perfectiva*: onde são realizadas alterações para melhorar alguns aspectos do *software*, como por exemplo, o seu desempenho, a sua *interface*, a introdução de novas funções;
- *Manutenção preventiva*: ocorre quando o *software* é modificado para melhorar a confiabilidade ou a manutenibilidade futura, ou para oferecer uma base melhor para futuras ampliações. Esta atividade é caracterizada pelas técnicas de engenharia reversa e reengenharia.

Normalmente é envolvido na manutenção de *software* as etapas de análise de sistema, teste das mudanças, teste das partes já existentes, o que a torna uma etapa complexa e com custo elevado.

2.3.2 PROTOTIPAÇÃO

O objetivo do processo de desenvolvimento baseado na prototipação é superar as duas primeiras limitações do modelo cascata. A idéia básica da prototipação, de acordo com Jalote (1997), é que ao invés de manter inalterado os requisitos durante o projeto e codificação, um protótipo é desenvolvido para ajudar no entendimento dos mesmos. O desenvolvimento do protótipo, obviamente, passa por um projeto, codificação e teste, mas cada uma destas fases não é executada formalmente. E, com a utilização de um protótipo, o cliente é habilitado a entender melhor os requisitos desejados do sistema. Desta forma, resulta em requisitos mais estáveis e assim serão alterados em uma freqüência menor.

A seqüência de eventos no processo de prototipação pode ser observada na figura 4.

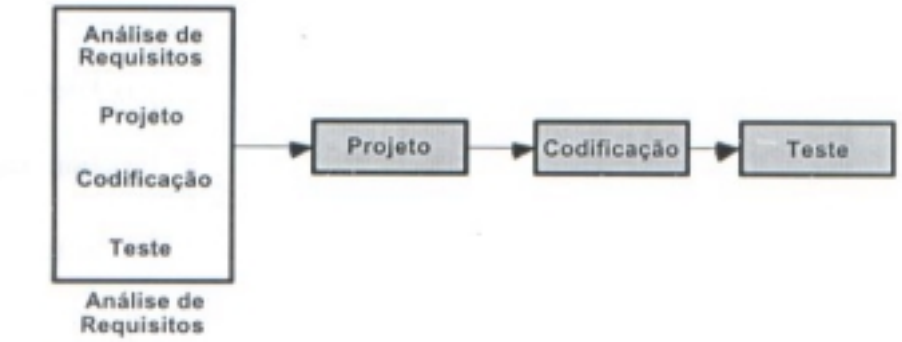


Figura 4 [Jalote (1997)]: Modelo de Prototipação

O desenvolvimento do protótipo tipicamente inicia com uma versão preliminar do documento de especificação dos requisitos a ser desenvolvidos. Depois do protótipo desenvolvido, os usuários finais têm a oportunidade de testar o protótipo.

Baseado na experiência dos usuários é fornecida uma avaliação para os desenvolvedores, informando o que está correto, o que precisa ser modificado, o que está faltando, o que não é necessário, dentre outros. Baseado nesta avaliação, o protótipo é modificado incorporando as sugestões de mudança que podem ser feitas facilmente e então os usuários utilizam novamente o protótipo. Este ciclo é repetido enquanto analistas e projetistas julgarem que o custo em se efetuarem as alterações é válido para a obtenção dos requisitos. Ao final do ciclo, os requisitos iniciais são modificados para produzir a especificação final dos mesmos, os quais são utilizados para a produção do sistema.

O modelo citado é chamado de prototipação descartável, diferente da prototipação evolutiva, em que os desenvolvedores iniciam projetando e implementando as partes essenciais do sistema em um protótipo e vão refinando este protótipo (adicionando novas funcionalidades e melhorias) até ele se tornar o produto de *software* final, que eventualmente será o produto a ser entregue.

A diferença do modelo evolutivo para o de descarte é que no de descarte, faz-se primeiramente um protótipo apenas para auxiliar na elaboração dos requisitos finais do sistema e depois este protótipo é descartado. Na prototipação evolutiva, o próprio protótipo será o produto a ser entregue ao cliente.

2.3.3 MODELO DE DESENVOLVIMENTO ITERATIVO

O modelo tenta combinar os benefícios de ambos os modelos - prototipação e cascata. Segundo Jalote (1997), a idéia básica é que o *software* deve ser desenvolvido

incrementalmente, onde a cada incremento, alguma funcionalidade é adicionada ao sistema, ou seja, modificações são executadas no projeto, até que o sistema completo esteja implementado.

Ghezzi (1991), ressalta que este modelo pode ser denominado também de evolucionário, incremental ou *delivery*. A figura 5 ilustra o referido modelo.

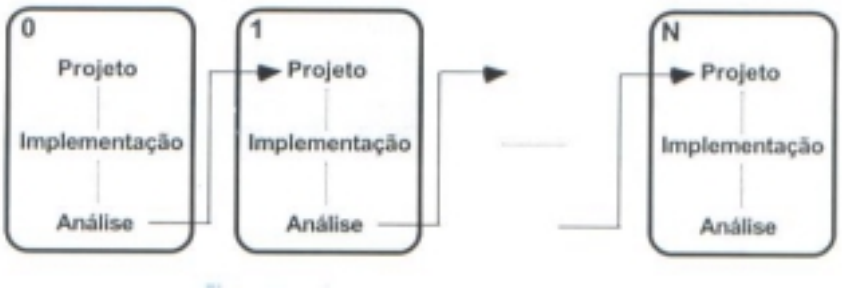


Figura 5 [Jalote (1997)]: Modelo Iterativo

Primeiramente, uma implementação inicial é feita para um subconjunto do problema. Este subconjunto contém alguns aspectos chaves do problema que é fácil de entender e implementar e cuja forma seja útil e usável. Uma lista de controle de projeto é criada. Esta lista contém, em ordem, todas as tarefas que devem ser executadas para obter a implementação final. Através da lista de controle de projeto é possível saber quão longe se está da conclusão do sistema.

Cada passo consiste de remover a próxima tarefa da lista, projetar e implementar a tarefa selecionada, codificar e testar a implementação, efetuar a análise do sistema parcial obtido após este passo e atualizar a lista com o resultado da análise. Estas três fases são denominadas fase de projeto, fase de implementação e fase de análise. O processo é iterativo até que a lista de controle de projeto esteja vazia.

2.3.4 MODELO ESPIRAL

O modelo espiral para o processo de produção de *software* tem o objetivo de prover um *framework* para projetar processos, guiado pelo nível de risco do projeto. O modelo espiral pode ser visto como um metamodelo, pois ele pode acomodar qualquer modelo de processo de desenvolvimento (GHEZZI, 1991). A figura 6 ilustra o modelo espiral, que é descrito na seqüência.

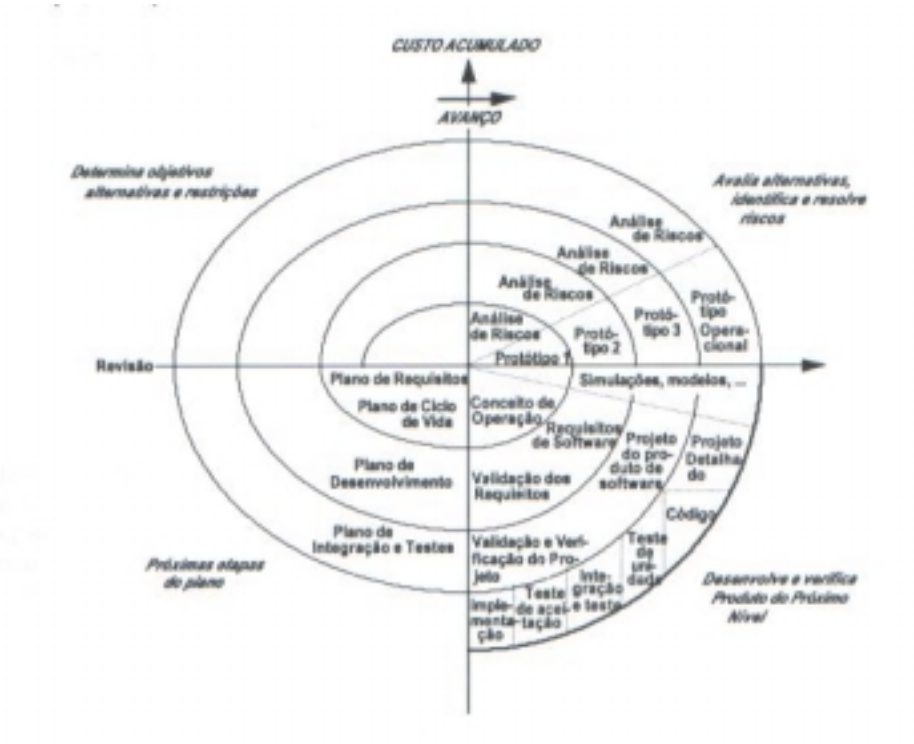


Figura 6 [Jalote (1997)]: Modelo Espiral

Cada ciclo do modelo é iniciado com a identificação dos objetivos para o mesmo, e as diferentes alternativas possíveis para alcançar os objetivos, bem como as restrições que existem. Este é o primeiro quadrante do ciclo (superior esquerdo). O próximo passo no ciclo é avaliar as diferentes alternativas baseado nos objetivos e restrições. O foco da avaliação neste passo é baseado na percepção do risco para o projeto. O próximo passo é desenvolver estratégias que eliminem as incertezas e os riscos. Este passo envolve atividades como *benchmar*k, simulação e prototipação. O próximo passo é desenvolver, mantendo em mente os riscos. Finalmente o próximo estágio é planejado.

Para Jalote (1997), risco reflete as chances de algum dos objetivos do projeto não ser alcançado. Já para Ghezzi (1991), riscos são circunstâncias potencialmente desfavoráveis que podem prejudicar o processo de desenvolvimento e a qualidade do produto.

2.3.5 ANÁLISE CRÍTICA DOS PARADIGMAS DE DESENVOLVIMENTO

Esta seção apresenta uma análise crítica dos modelos estudados para a realização do trabalho.

Análise Crítica do Modelo Cascata

Apesar de ser um modelo bastante popular, pode-se apontar algumas limitações apresentadas pelo modelo, e citadas em Pressman (1995), Ghezzi (1991), Jalote (1997) e Von Mayrhauser (1990):

- O modelo assume que os requisitos são inalterados ao longo do desenvolvimento; isto em boa parte dos casos não é uma condição verdadeira, uma vez que nem todos os requisitos são completamente definidos na etapa de análise;
- Muitas vezes, a definição dos requisitos pode conduzir à definição do *hardware* sobre o qual o sistema vai funcionar; dado que muitos projetos podem levar anos para serem concluídos, estabelecer os requisitos em termos de *hardware* é um tanto temeroso, dadas as freqüentes evoluções no *hardware*;
- O modelo impõe que todos os requisitos sejam completamente especificados antes do prosseguimento das etapas seguintes; em alguns projetos, é as vezes mais interessante especificar completamente somente parte do sistema, prosseguir com o desenvolvimento do sistema, e só então encaminhar os requisitos de outras partes; isto não é previsto ao nível do modelo;
- As primeiras versões operacionais do *software* são obtidas nas etapas mais tardias do processo, o que na maioria das vezes inquieta o cliente, uma vez que ele quer ter acesso rápido ao seu produto.

Em função destas limitações alguns autores apresentam o modelo cascata possuindo *feedback* entre as fases. Esta visão é abordada por Ghezzi (1991) e Pressman (1995).

Análise Crítica do Modelo de Prototipação

Pressman (1995) aponta alguns problemas na utilização do modelo baseado em prototipação:

- O cliente vê aquilo que parece ser uma versão do trabalho do *software*, desconhecendo que o protótipo se mantém unido, sem saber que, na pressa de colocá-lo em funcionamento, não é levado em consideração a qualidade global do *software* e a manutenibilidade a longo prazo. Quando informa-se que o produto precisa ser reconstruído, o cliente exige que “alguns acertos” sejam aplicados para tornar o protótipo um produto de trabalho.
- O desenvolvedor muitas vezes faz concessões de implementação a fim de colocar um protótipo em funcionamento rapidamente. Concessões essas, que podem se tornar parte integrante do sistema.

No entanto, Pressman (1995) e Jalote (1997), destacam alguns benefícios da utilização deste modelo:

- O modelo é interessante para alguns sistemas de grande porte os quais representem um certo grau de dificuldade para exprimir rigorosamente os requisitos.
- Através da construção de um protótipo do sistema é possível demonstrar a realizabilidade do mesmo.
- É possível obter uma versão, mesmo simplificada, do que será o sistema com um pequeno investimento inicial.
- A experiência de desenvolver o protótipo, pode reduzir o custo das fases posteriores.

Análise Crítica do Modelo Iterativo

A vantagem deste paradigma é a facilidade em testar o sistema, uma vez que a realização de testes em cada nível de desenvolvimento é, sem dúvida, mais fácil do que testar o sistema final. Além disto, como na prototipação, a obtenção de um sistema, mesmo incompleto num dado nível, pode oferecer ao cliente interessantes informações que sirvam de subsídio para a melhor definição dos requisitos finais do sistema (JALOTE, 1997).

No entanto, o autor explica que, um problema prático com este tipo de desenvolvimento encontra-se na elaboração do contrato – como o custo das características adicionais será determinado e negociado.

Análise Crítica do Modelo Espiral

Segundo Pressman (1995), as vantagens deste modelo são:

- O paradigma de modelo espiral para a ES atualmente é a abordagem mais realística para o desenvolvimento de sistemas e de *software* em grande escala.
- Usa uma abordagem evolucionária à ES, capacitando o desenvolvedor e o cliente a entender e reagir aos riscos em cada etapa evolutiva.
- O modelo se adequa a sistemas que representem um alto risco de investimento para o cliente.

No entanto, Pressman (1995), observa que alguns cuidados devem ser tomados:

- Pode ser difícil convencer grandes clientes de que a abordagem evolutiva é controlável.
- Exige considerável experiência na avaliação dos riscos e baseia-se nessa experiência para o sucesso.

Foram abordados nesta seção assuntos relevantes aos modelos de desenvolvimento de *software*, a próxima seção dará ênfase ao gerenciamento de projetos.

2.4 GERENCIAMENTO DE PROJETOS

O gerenciamento constitui uma tarefa de fundamental importância no processo de desenvolvimento de um produto e o mesmo não pode ser visto como uma etapa clássica do processo de desenvolvimento, uma vez que ele acompanha todas as etapas do desenvolvimento, da concepção à manutenção do produto (PRESSMAN, 1995). Conforme pode ser observado na figura 7.

Figura 7: Gerenciamento de *Software*

Vários são os problemas relacionados ao desenvolvimento de *software*, os quais na maioria das vezes são resultantes da omissão ou do mau uso de metodologias e técnicas adequadas a esta importante tarefa de engenharia. Chang e Christensen (1999), citam que são muitos os problemas dos gerentes de projetos de *software* para a realização de grandes sistemas, incluindo: (a) reunir, treinar e motivar uma grande equipe; (b) desenvolver ou adotar processos de gerenciamento e engenharia; (c) desenvolver e manter requisitos; (d) planejar, orçar e monitorar o projeto; (e) identificar e resolver conflitos de recursos e (f) monitorar o projeto inteiro continuamente.

Royce (1998) relata o resultado de três análises, realizadas na metade dos anos 90, sobre o estado da engenharia de *software* nas empresas. As três análises chegaram à mesma conclusão, resumida em:

- Desenvolvimento de *software* ainda é altamente imprevisível. Apenas, aproximadamente 10% dos projetos de *software* são entregues com sucesso, considerando o orçamento e cronograma estimado.

- Disciplina de gerenciamento é mais um discriminador do sucesso ou fracasso que uma tecnologia avançada.
- O nível de re-trabalho é um indicativo de um processo imaturo.

Pressman (1995) afirma que boa parte dos fracassos no que diz respeito aos projetos de *software* deve-se, principalmente, a problemas de administração ou gerenciamento do desenvolvimento de *software*.

Sommerville (2003) e O'Connor (2000) atribuem a dificuldade de gerenciamento de projetos de *software* à diferença existente em relação a outros tipos de projetos, podendo-se destacar cinco aspectos:

- *Software* não é "palpável"/visível: quando o produto a ser desenvolvido é uma ponte ou uma estrada, por exemplo, o progresso pode ser claramente verificado. No *software*, o progresso não é visível, o gerente de projeto depende da documentação para verificar o progresso do projeto.
- Não se tem o entendimento claro do processo de *software*: na engenharia de *software* os modelos de desenvolvimento de *software* são representações simplificadas do processo.
- Vários projetos de *software* são "projetos únicos", ou seja, não há semelhança com projetos previamente desenvolvidos. Nestes casos, a experiência histórica é um valor limitante em predizer como o projeto deverá ser gerenciado.
- Complexidade: "Cada dólar, peso ou euro gasto em um produto de *software* contém mais complexidade que outros artefatos da engenharia" (O'CONNOR, 2000).
- Flexibilidade: a facilidade com a qual um *software* pode ser alterado é geralmente visto como uma de suas vantagens.

2.4.1 PARÂMETROS E ATIVIDADES NO GERENCIAMENTO DE *SOFTWARE*

Pressman (1995) e Jalote (1997) definem que, para um projeto de *software* ser bem sucedido é necessário que alguns parâmetros sejam bem analisados, como por exemplo, o escopo do *software*, os riscos envolvidos, os recursos necessários, as tarefas a serem realizadas, os marcos de referência a serem acompanhados, os custos aplicados e a sistemática a ser seguida. A análise de todos estes parâmetros, segundo Pressman (1995) é a função típica do gerenciamento de projetos. Função esta, que se inicia antes do trabalho técnico e que prossegue à medida que o *software* vai se concretizando na forma de um produto.

Sommerville (2003) acrescenta a estes parâmetros algumas atividades que também são de responsabilidade do gerente de projeto, que são a seleção e avaliação da equipe e

elaboração de relatórios. Já Jalote (1997) cita a determinação do cronograma e *milestone* como atividades pertinentes ao gerenciamento.

A seguir, são apresentadas algumas destas atividades:

2.4.1.1 ATIVIDADE INICIAL DO GERENCIAMENTO

Nesta fase do gerenciamento, o objetivo é levantar alguns pontos importantes para a boa condução do projeto como objetivos do *software*, alternativas de solução, restrições administrativas e técnicas.

O conhecimento destas informações vai permitir obter-se uma primeira estimativa do projeto em termos de custo, além de proporcionar uma melhor divisão no que diz respeito às tarefas do projeto, estabelecendo, inclusive, subsídios para que se possa avaliar o estado de evolução do projeto (PRESSMAN, 1995).

É neste momento que as duas partes envolvidas no projeto, o desenvolvedor e o cliente, devem reunir-se para definir os objetivos e o escopo do projeto É importante destacar que o levantamento dos objetivos do projeto não levarão em conta, nesta fase, como eles serão alcançados. A definição do escopo do *software* permitirá obter, num nível elevado de abstração, as principais funções a serem supridas pelo produto (JALOTE, 1997).

Somente após estes dois pontos terem sido levantados é que serão discutidas as alternativas de solução, como forma de melhor selecionar a abordagem a ser adotada de modo que permita respeitar as restrições impostas: prazo de entrega, orçamento, disponibilidade de pessoal, dentre outros.

2.4.1.2. MEDIDAS E MÉTRICAS

Em boa parte dos empreendimentos técnicos, verifica-se que as medições e as métricas permitem um melhor entendimento do processo utilizado para desenvolver um produto, assim como uma melhor avaliação do próprio produto, consenso encontrado em (Candéas e Lopes, 1999; IFPUG, 2003; *Software Productivity Research*, 2003 e Pressman, 1995).

A quantificação dos aspectos relacionados ao processo de obtenção de um produto e do produto em si é importante, pelas seguintes razões:

- No caso do processo de desenvolvimento, as medições podem permitir melhorias no processo, aumentando a sua produtividade;
- No caso do produto, as medições podem proporcionar informações a respeito de sua qualidade.

Apesar de importante, nem sempre a medição é uma tarefa evidente, conduzindo a diversos questionamentos:

- Quais são as métricas mais adequadas para o processo e para o produto?
- Como os dados obtidos serão processados?
- É justo utilizar medições para se comparar pessoas, processos e produtos?

2.4.1.3 ESTIMATIVA

Conforme Candéas e Lopes (1999), a estimativa é um exercício importante, especialmente para o planejamento do projeto de *software*. Fatores como o esforço humano exigido, duração cronológica do projeto, custo e outros, devem ser levantados neste momento. O problema é como definir estes fatores.

Em grande parte dos casos, as estimativas são feitas com base na experiência passada. No caso de se ter um projeto relativamente similar a um projeto já realizado, não fica difícil estimar questões como esforço, cronograma e custo, uma vez que estes serão muito próximos daqueles relativos ao projeto anterior (PRESSMAN, 1995).

Por outro lado, a estimativa através desta abordagem pode tornar-se complexa se o novo projeto apresenta características inovadoras com relação ao(s) projeto(s) anterior(es).

Apesar da existência de diversas técnicas de estimativa, é importante destacar algumas de suas características comuns, conforme encontrado em *Software Productivity Research* (2003):

- O escopo do projeto é estabelecido previamente;
- São utilizadas métricas de *software* e histórico de aferições passadas como base das estimativas;
- O projeto é dividido em pequenas partes as quais são estimadas individualmente.

Em alguns casos, os gerentes de projeto utilizam mais de uma técnica de estimativa, de modo que os resultados obtidos são comparados para verificar a coerência dos cálculos realizados.

2.4.1.4 ANÁLISE DE RISCOS

Em qualquer projeto de engenharia, e os projetos de *software* não fogem a esta regra, existe um conjunto de incertezas, citadas em Pressman (1995):

- A necessidade de um computador é algo realmente claro?
- As funções a serem implementadas estarão prontas antes do prazo final?

- Que problemas técnicos serão enfrentados ao longo do projeto?
- Eles serão contornados de modo a cumprir o cronograma?
- As mudanças de requisitos que eventualmente aparecerão não provocarão um aumento considerável no tempo e custos de desenvolvimento?

Apesar de, em muitos casos, não ser considerada a análise de riscos é uma tarefa de grande importância no gerenciamento de um projeto de *software*. O seu objetivo é, determinar um conjunto de passos a serem seguidos para determinar os riscos envolvidos no projeto: identificação, avaliação, classificação, definição de estratégias para administrar os riscos, resolução dos riscos, dentre outros.

2.4.1.5 DETERMINACÃO DE PRAZOS

Outro aspecto importante, relacionado ao gerenciamento de um projeto de *software*, ressaltado por Pressman (1995), é a definição dos prazos de desenvolvimento. Cada projeto de *software* é caracterizado por um conjunto de atividades a serem executadas.

Como qualquer outro projeto, o projeto de um *software* deve obedecer a um conjunto bem definido de tarefas, ao inter-relacionamento das tarefas, sendo que um determinado esforço será dedicado à realização de cada tarefa envolvendo a alocação de um determinado grupo de pessoas e recursos.

Com base nestas informações, é possível criar uma representação que exprima o tempo dedicado a cada tarefa, determinando assim o prazo final do projeto.

2.4.1.6 MILESTONES

Durante o planejamento de um projeto, Sommerville (2003) comenta que, uma série de *milestones* (marcos de referência) deve ser estabelecida. *Milestone* é um ponto final de uma atividade do processo de desenvolvimento de *software*. Portanto, os *milestones* devem representar o final de um estágio distinto do projeto. Sendo que para cada *milestone* um relatório deve ser elaborado.

Um bom *milestone* é caracterizado pela elaboração da documentação, por exemplo, “plano de testes formulado”. *Milestones* não deve ser indefinido, por exemplo, um *milestone* mal definido seria ‘60% do código concluído’. Pois não há maneira objetiva de se afirmar que ‘60% do código foi concluído’.

Milestones não precisam ser estabelecidos para cada atividade do projeto. Uma regra prática, citada por Sommerville (2003), é que os *miliestones* devem estar agendados em

intervalos de duas a três semanas, podendo variar dependendo do processo de desenvolvimento seguido.

2.4.1.7 CRONOGRAMA

Desenvolver o cronograma de um projeto é uma das tarefas mais difíceis do gerenciamento de *software*, afirma Sommerville (2003), pois envolve estimar o tempo e os recursos necessários para completar as atividades/tarefas e organizá-las em uma seqüência coerente. A menos que o cronograma do projeto seja similar a um projeto prévio, estimativas prévias não são uma boa base. Diferentes projetos usam diferentes linguagens de programação e metodologias, o que complica a tarefa de estimar o cronograma.

O cronograma de projetos separa o trabalho envolvido no projeto em tarefas distintas e avalia quando estas tarefas serão concluídas. Geralmente algumas destas tarefas são realizadas em paralelo. Estas tarefas devem ser coordenadas pelos gerentes de projetos, o qual deve organizar o trabalho a fim de evitar situações as quais o projeto inteiro é atrasado em decorrência a uma tarefa não ser concluída.

A regra prática, citada em Sommerville (2003), para prever o cronograma é estimar como se nada irá dar errado, incrementar a estimativa para cobrir problemas previstos e então adicionar um “fator contingência” para cobrir problemas não previstos.

Este “fator contingência” depende do tipo do projeto, qualidade e experiência da equipe. Para estimar o tempo total requerido pelo projeto, Sommerville (2003) comenta que se pode usar o tamanho estimado do sistema e dividir pela produtividade esperada da equipe.

O resultado do processo de cronograma pode ser expresso através de gráficos relacionando às tarefas, dependências entre tarefas e alocação de pessoal.

2.4.1.8 MONITORAÇÃO E CONTROLE

Após ser definida a programação do desenvolvimento (cronograma), inicia-se a atividade de monitoração e controle do projeto, a qual permanecerá durante todo o projeto, segundo Pressman (1995). O gerente de projeto monitora cada tarefa prevista no programa. Caso o desenvolvimento desta não esteja de acordo com a programação, o gerente pode atuar no sentido de avaliar o impacto do não cumprimento dos prazos para o projeto como um todo, ou atuar sobre ela no sentido de fazer com que a sua realização aproxime-se do que foi programado. Isto pode ser feito pela alocação de mais recursos para o cumprimento da tarefa.

Ainda, em função desta avaliação, tarefas podem ser reorganizadas no sentido de que o projeto global, apesar dos atrasos envolvidos em algumas tarefas, possa ser concluído dentro do prazo final estipulado.

2.4.1.9 SELEÇÃO E AVALIAÇÃO DA EQUIPE

O gerente de projeto geralmente tem a responsabilidade de selecionar a equipe que comporá o projeto. O ideal, segundo Sommerville (2003) é ter pessoal com habilidade e experiência para realizar o projeto. No entanto, o autor assegura que na maioria dos casos isto é inalcançável devido as seguintes razões:

- O orçamento pode tornar impossível o uso de pessoal com salário alto.
- Pessoal com a experiência apropriada pode simplesmente não estar disponível.
- A organização pode exigir que pessoal em treinamento trabalhe no projeto para adquirir experiência.

O gerente de projeto tem que trabalhar com estas hipóteses quando selecionar a equipe.

2.4.2 FASES DO PROCESSO DE GERENCIAMENTO

Um gerenciamento moderno de *software* está baseado em diversos princípios, podendo-se priorizar: processo baseado em uma abordagem em que primeiramente deve-se definir a arquitetura, estabelecer um processo de desenvolvimento iterativo, métodos de projeto devem enfatizar o desenvolvimento baseado em componentes, estabelecer um ambiente de gerenciamento flexível e utilização de ferramentas automatizadas que suportam diferentes formatos (ROYCE, 1998).

Jalote (1997), enquadra estas atividades dentro de três grandes fases do processo de gerenciamento que são: planejamento, monitoramento e controle e análise conclusiva.

2.4.2.1 PLANEJAMENTO

O objetivo desta seção é descrever sobre a importância do planejamento para o gerenciamento de projetos. Pois, conforme citado por Jalote (1997) e Sommerville (2003), “esta atividade freqüentemente absorve a maioria dos esforços gerenciais de um projeto de *software*”. A importância do planejamento também é ressaltada por Wu e Simmons (2000), ao afirmar que “sem um planejamento realista e objetivo do projeto do *software*, o processo de desenvolvimento não pode ser gerenciado de maneira efetiva”.

Esta fase tem como objetivo segundo Ghezzi (1991), identificar todos os requisitos do projeto, para então desenvolver um planejamento para o desenvolvimento do *software* cujos objetivos do projeto possam ser alcançados eficientemente. Sendo desta forma, o planejamento do projeto considerado base fundamental para o gerenciamento.

O gerenciamento efetivo de projeto de *software* depende de um planejamento completo do progresso do projeto, sendo, portanto, desenvolvido antes das atividades de desenvolvimento. Este planejamento, segundo Sommerville (2003), deve antecipar problemas que possam surgir e preparar soluções com antecedência. O plano redigido no início do projeto deve ser usado para dirigir o projeto. Este plano inicial não é estático, pois, deve ser modificado conforme a evolução do projeto (O'CONNOR, 2000).

Uma das questões a ser definida nesta fase é determinar qual o modelo de processo de desenvolvimento é adequado ao projeto. Outra decisão crítica é determinar os recursos necessários para o projeto como, quantidade e habilidade das pessoas a serem envolvidas no processo e recursos computacionais. Por conseguinte, o custo do projeto é diretamente proporcional ao número de pessoas envolvidas no projeto (GHEZZI, 1991).

Jalote (1997) e Wu e Simmons (2000), acrescentam às questões abordadas por Ghezzi (1991), a determinação do *schedule* e *milestones* e a definição da equipe como atividades pertinentes a esta fase do gerenciamento.

Para Ghezzi (1991), um dos requisitos básicos do gerenciamento é medir a produtividade das pessoas e processos envolvidos na produção. As medidas obtidas são usadas durante a fase de planejamento do projeto como base para estimar recursos.

Sommervile (2003), ressalta que estimar a produtividade dos programadores é importante por duas razões:

- Sem uma estimativa de produtividade é impossível prever um cronograma confiável;
- Os benefícios de usar técnicas da ES e ferramentas podem demonstrar aos superiores que o gerente usa resultados para melhorar a produtividade em todo o ciclo de vida do *software*.

Devido ao fato do *software* não ser "palpável" não é possível medir a produtividade diretamente. Produtividade em sistemas manufaturados pode ser medida contando o número de unidades produzidas dividido pelo número de horas trabalhadas.

Em sistemas computacionais, o que se espera é estimar o custo de um sistema dado a sua funcionalidade.

No entanto, Sommerville (2003) ressalta que “métricas de produtividade são somente um guia subjetivo e relativo de produtividade. Os gerentes de projeto devem acrescentar percepção e intuição para estimar a real produtividade”.

2.4.2.2 MONITORAMENTO E CONTROLE

Segundo Jalote (1997), trata-se da fase mais longa do processo de gerenciamento, abrangendo a maior parte do processo de desenvolvimento. Para Ghezzi (1991) o propósito desta fase é monitorar o progresso das atividades planejadas e garantir que os objetivos estão sendo executados.

Custo, *schedule* e tarefas são os principais fatores a serem gerenciados, portanto, o maior esforço desta fase envolve monitorar e controlar estes fatores. Outra importante atividade desta fase é monitorar os riscos potenciais do projeto. Se as informações obtidas pelo monitoramento sugerem que os objetivos do projeto podem não ser alcançados, faz parte das tarefas desta fase planejar as ações para controlar as atividades de desenvolvimento (JALOTE, 1997).

O monitoramento e controle do processo requerem informações referentes ao projeto. Tais informações são obtidas pelo processo de gerenciamento a partir do processo de desenvolvimento, sendo a interpretação das informações parte do monitoramento e controle (JALOTE, 1997).

2.4.2.3 ANÁLISE CONCLUSIVA

Considerando que a fase de monitoramento e controle permanece durante toda a execução do projeto, a última fase do processo de gerenciamento, é executada quando o processo de desenvolvimento termina. A razão básica para realizar a análise conclusiva é prover informação sobre o processo de desenvolvimento. Esta informação se faz necessária para compreender as características do processo, pois as informações retiradas desta análise darão suporte e estimativas para futuros projetos. Além disso, esta fase prevê realizar uma análise sobre o próprio processo utilizado (JALOTE, 1997).

Esta seção deu ênfase aos conteúdos de atividades e fases do processo de gerenciamento, o assunto referente à métrica de *software* será abordado a seguir.

2.5 MÉTRICA DE *SOFTWARE*

Na área de engenharia, a medição tem sido um aspecto de grande importância, sendo que pode-se citar uma gama interminável de grandezas as quais sofrem este tipo de

tratamento: peso, temperatura, tensões e correntes elétricas, dentre outros. Na computação, alguns parâmetros são quantificados como forma de expressar as potencialidades de determinadas máquinas, tais como a capacidade de um processador de executar um certo número de instruções por segundo (MIPS), a capacidade de armazenamento (Mbytes), a freqüência do *clock* do processador (MHz), dentre outros.

A engenharia de *software* aplica uma abordagem de engenharia para construção e suporte ao produto de *software*, envolvendo atividades de gerenciamento, custo, planejamento, modelagem, especificação, projeto, implementação, teste e manutenção. Por abordagem de engenharia compreende-se que cada atividade é entendida e controlada, ocorrendo poucas surpresas na especificação do *software*, projeto, construção e manutenção. Considerando que a ciência da computação fornece fundamentação teórica para construção de *software*, sendo a engenharia de *software* focada na implementação de *software* de maneira controlada e científica (FENTON, 1997).

Fenton (1997) lista os tipos de informações necessárias para entender e controlar um projeto de desenvolvimento de *software*, sobre a perspectiva do gerente e do desenvolvedor. Como abordou-se a utilização de métrica para o gerenciamento de projetos, a seguir encontram-se algumas informações úteis a um gerente de projeto de *software*:

- Medir o tempo e esforço envolvido nas diferentes fases do processo de produção de *software*. Por exemplo, pode-se identificar o custo da especificação de requisitos, o custo do projeto do sistema e o custo de implementação e teste do *software*. Podendo-se assim identificar não somente o custo total do projeto, mas também o custo envolvido em cada etapa;
- Medir o tempo levado pela equipe para especificar o sistema, projetá-lo, implementá-lo e testá-lo, determinando assim a produtividade da equipe em cada atividade. Esta informação é útil quando mudanças são solicitadas, pois pode-se utilizar a produtividade para estimar o custo e duração das alterações;
- Avaliar a qualidade do *software* desenvolvido. Armazenando as falha, erros e mudanças, pode-se medir a qualidade do *software* permitindo, comparar diferentes produtos, predizer o efeito de alterações e avaliar o efeito da aplicação de novas práticas;
- Medir a funcionalidade, determinando se todos os requisitos foram implementados apropriadamente. Pode-se também avaliar a usabilidade, confiabilidade, tempo de resposta, dentre outras características, para garantir a satisfação do cliente;

- Medir o tempo para executar cada atividade principal do desenvolvimento e calcular seu efeito na qualidade e produtividade. Então pode-se avaliar a relação custo/benefício de cada prática. Viabilizando assim, testar várias práticas, medir os resultados e decidir qual a melhor. Por exemplo, pode-se comparar dois métodos de projeto e verificar qual apresenta maior qualidade no código.

Esta listagem mostra, segundo Fenton (1997), como a medição faz–se importante para três atividades básicas.

- *Primeiro*: existem medidas que auxiliam entender o que está ocorrendo durante o desenvolvimento e manutenção. As medidas tornam os aspectos do processo e produto mais visíveis, dando um melhor entendimento do relacionamento entre atividades;
- *Segundo*: a medição permite controlar o que está acontecendo nos projetos. Por exemplo, pode-se monitorar a complexidade dos módulos e efetuar uma rigorosa revisão somente naqueles que excedem uma certa medida;
- *Terceiro*: medições encorajam a melhorar o processo de desenvolvimento e o produto. Por exemplo, pode-se aumentar a quantidade ou tipos de revisão de projetos baseado nas medidas obtidas na especificação de requisitos.

Pressman (1995) e Braga (1996), acrescentam diversas razões para se considerar as medições um item de importância para o gerenciamento:

- Quantizar a qualidade do *software* como produto;
- Avaliar a produtividade dos elementos envolvidos no desenvolvimento do produto;
- Estimar o tamanho de um sistema antes de desenvolvê-lo;
- Avaliar os benefícios de métodos e ferramentas para o desenvolvimento de *software*;
- Formar uma base de dados para as estimativas;
- Justificar o pleito e aquisição de novas ferramentas e/ou treinamento adicional para membros da equipe de desenvolvimento.

A seguir é apresentado as categorias principais de métricas.

2.5.1 TIPOS DE MÉTRICAS

As medições de *software* podem ser classificadas em duas categorias principais, segundo Bomfim (2003) e Pressman (1995):

- A*s medições diretas*: por exemplo, o número de linhas de código (LOC) produzidas, o tamanho de memória ocupado, a velocidade de execução, o número de erros registrados num dado período de tempo, dentre outros;

- *As medições indiretas*: permitem quantizar aspectos como a funcionalidade, complexidade, eficiência, manutenibilidade, dentre outros.

As medições diretas, tais quais aquelas exemplificadas acima, são de obtenção relativamente simples, desde que estabelecidas as convenções específicas para isto. Por outro lado, aspectos como funcionalidade, complexidade e eficiência são bastante difíceis de quantizar.

As medições de *software* podem ser organizadas em outras classes, as quais serão definidas a seguir, conforme citado por Bomfim (2003) e Pressman (1995):

- *Métricas da produtividade*: baseadas na saída do processo de desenvolvimento do *software* com o objetivo de avaliar o próprio processo;
- *Métricas da qualidade*: indicam o nível de resposta do *software* às exigências explícitas e implícitas do cliente;
- *Métricas técnicas*: nas quais encaixam-se aspectos como funcionalidade, modularidade, manutenibilidade, dentre outros.

Sob uma outra ótica, definida por Pressman (1995) é possível definir uma nova classificação das medições:

- *Métricas orientadas ao tamanho*: baseadas nas medições diretas da engenharia de *software*. Esta classe abrange todas as possíveis medidas obtidas diretamente do *software*, um exemplo é a utilização de linhas de código;
- *Métricas orientadas à função*: oferecem medidas indiretas. Esta classe é baseada em medidas indiretas do *software* e do processo utilizado para obtê-lo. Esta métrica leva em conta aspectos como, a funcionalidade e a utilidade do programa;
- *Métricas orientadas às pessoas*: as quais dão indicações sobre a forma como as pessoas desenvolvem os produtos de *software*.

Foi apresentado nesta seção, algumas definições e tipos de métrica de *softwares*. A próxima seção visa abordar o assunto de *Interface* homem-computador.

2.6 INTERFACE HOMEM-COMPUTADOR

Esta seção visa definir *interface* de usuário, seus objetivos e apresenta o estilo de interação *WIMP* (*Windows, Icons, Menus e Pointers*).

2.6.1 ALGUMAS CONSIDERAÇÕES SOBRE *HCI*

A capacidade e limitação física do ser humano requerem o desenvolvimento de um *hardware* de *interface* específico. Estudos de como o ser humano enxerga ou escuta e suas habilidades manuais são fontes de conhecimento para a tecnologia de entrada e saída. Da mesma forma, o projeto do modelo de interação requer conhecimento proveniente de estudos de psicologia que indiquem capacidades de memorização, raciocínio e aprendizado, dentre outras faculdades mentais. A figura 8 ilustra a interação do usuário com o sistema.

Figura 8: Interação Usuário-Sistema

A *HCI* visa fornecer aos pesquisadores e desenvolvedores de sistemas explicações e previsões para fenômenos de interação usuário-sistema e resultados práticos para o projeto da *interface* de usuário, buscando desenvolver modelos teóricos da performance e cognição humana, bem como técnicas efetivas para avaliar a usabilidade (LINDGARD, 1994).

Um dos objetivos dos pesquisadores envolvidos com *HCI* é chamar a atenção dos desenvolvedores, para considerar no desenvolvimento de sistemas, não apenas o sistema e sua *interface*, mas também, usuários envolvidos e o processo de interação entre eles. Portanto, a *HCI* é multidisciplinar e está relacionada com áreas tais como: ciência da computação, que sustenta o conhecimento tecnológico para a engenharia de *software* e *hardware*; ergonomia, para o estudo do desempenho físico do usuário; psicologia e ciência cognitiva, que oferecem conhecimento sobre comportamento e habilidades perspectivas e cognitivas, bem como técnicas de análise e avaliação empírica; dentre outras (CYBIS, 2003).

Norman (1986) é um pesquisador da área que vem influenciando o trabalho de vários outros pesquisadores. O autor apresentou um modelo das atividades do usuário na interação com o sistema que funciona como um quadro teórico de referência, uma vez que vários outros modelos podem ser alocados em sua moldura.

O modelo de Norman (1986), permite ao usuário mapear suas metas em comandos e funções do sistema. Segundo esta proposta, o papel do *designer* está em desenvolver uma *interface* que permita ao usuário, durante o processo de interação, adquirir um modelo mental correspondente ao modelo do sistema. Para isto são necessários modelos cognitivos das atividades mentais. Ele considera que este modelo mental do usuário é adquirido durante a interação com a *interface*, o sistema de ajuda e toda a documentação do sistema.

Como fundamentação teórica, o modelo de Norman (1986) revela as atividades desempenhadas pelo usuário, permitindo identificar quais elementos devem ser projetados e qual o impacto que eles terão no desempenho do usuário:

- *Estabelecimento da meta*: o usuário utiliza o sistema como ferramenta para realizar uma tarefa. Para isso ele deve estabelecer uma meta. Então, o usuário deve ter conhecimento a respeito dos objetos do sistema e como eles podem ser modificados. O *designer* deve preocupar-se em deixar claro para o usuário quais são os objetos do sistema e que tarefas podem ser realizadas;
- *Planejamento ou formulação da intenção*: a meta que o usuário deseja, apenas pode ser atingida por alguma das funções do sistema. Caso exista esta função, a meta será atingida diretamente por ela. Senão, o usuário deve realizar um planejamento mental no qual a meta inicial deve ser dividida em submetas, sucessivamente, até que para cada uma delas exista uma função do sistema que permita atingí-la. Este planejamento só é possível se o usuário conhecer as funções que o sistema oferece (denominado de funcionalidade);
- *Especificação da estrutura de ações*: uma vez realizado o planejamento, o usuário precisa especificar as ações que controlam a função planejada. Estas ações podem estar estruturadas em seqüência, combinação simultânea, repetição, dentre outras;
- *Execução*: as atividades anteriores são a preparação mental para a execução física propriamente. Nesta etapa existe o esforço físico de acionar o *hardware* e os *widgets* da *interface*. Cabe ao *designer* minimizá-lo, bem como aumentar a produtividade do usuário através de um projeto de comandos e de *hardware* de *interfaces* adequados;
- *Percepção*: escolher o canal sensorial adequado (normalmente o visual e/ou auditivo) e qual a distinção naquele canal chamará a atenção do usuário;
- *Interpretação*: é o que o usuário percebe, precisa ser interpretado para que possa ser avaliado. Cada distinção num canal sensorial - símbolo, palavra escrita, ruídos, palavra

sonora, ícone - quando interpretado pelo usuário é um signo que representa algo sobre o sistema e deve permitir ao usuário avaliar se a meta foi atingida;

- *Avaliação*: é realizada através da comparação daquilo que foi interpretado com a meta que havia sido estabelecida inicialmente.

A perspectiva cognitiva mostra que a facilidade de aprendizado deve ser solucionada com o *design* de *interfaces* que proporcionem o menor esforço cognitivo para as atividades e na aquisição do modelo conceitual. O objetivo destas abordagens é descobrir o conhecimento teórico a respeito das atividades mentais que orientam o projeto da *interface* para usabilidade.

2.6.2 *INTERFACE DE USUÁRIO*

O termo *interface* é aplicado normalmente àquilo que interliga dois sistemas. Tradicionalmente, considera-se que uma *interface* homem-máquina é a parte de um artefato que permite a um usuário controlar e avaliar o funcionamento do mesmo através de dispositivos sensíveis às suas ações e capazes de estimular sua percepção. No processo de interação usuário-sistema, a *interface* é o combinado de *software* e *hardware* necessário para viabilizar e facilitar os processos de comunicação entre o usuário e a aplicação. A *interface* entre usuários e sistemas computacionais diferencia-se das *interfaces* de máquinas convencionais por exigir dos usuários um maior esforço cognitivo em atividades de interpretação e expressão das informações que o sistema processa (NORMAN, 1986).

A *interface* é tanto um meio para a interação usuário-sistema, quanto uma ferramenta que oferece os instrumentos para este processo comunicativo. Desta forma, a *interface* é um sistema de comunicação. Quando se considera a aplicação como máquina(s) virtual(is), a *interface* pode ser considerada ainda como um ambiente virtual para ações.

A *interface* possui componentes de *software* e *hardware*. Os componentes de *hardware* compreendem os dispositivos com os quais os usuários realizam as atividades motoras e perceptivas. Entre eles estão a tela, o teclado, o mouse e vários outros. O *software* da *interface* é a parte do sistema que implementa os processos computacionais necessários para controle dos dispositivos de *hardware*, para a construção dos dispositivos virtuais (os *widgets*) com os quais o usuário também pode interagir, para a geração dos diversos símbolos e mensagens que representam as informações do sistema e para a interpretação dos comandos dos usuários.

Segundo Pressman (1995), a *interface*, é, de muitas maneiras, a embalagem do *software*, se ela for amigável ao usuário este não terá dificuldades em adaptar-se ao aplicativo. Caso contrário, poderá tornar-se um grave problema. Se o projeto da *interface* for bom, o usuário cairá num ritmo natural de interação, mas se ela for ruim, o usuário imediatamente

ficará insatisfeito com o modelo de interação “não amigável”. Trazendo desta forma sérios problemas no uso do aplicativo e até abandono do mesmo.

Norman (1986) alude que, para o usuário poder utilizar o sistema com sucesso, ele deve saber quais as funções são oferecidas pelo sistema e como ele pode interagir com cada uma delas, isto é, qual o modelo conceitual da aplicação o projetista concebeu para tal.

A *interface* de usuário tem dois objetivos fundamentais:

- Determinar como o usuário pode efetivamente interagir com o sistema, desenvolvendo-se uma *interface* que permita ao usuário manipulá-la de acordo com o modelo conceitual de interação.
- Mostrar para o usuário o que ele pode fazer, isto é, quais as funções da aplicação o sistema oferece, e quais os comandos de funções e mensagens auxiliares que compõem o modelo de interação.

Pois estas definições devem ficar bem claras tanto para o projetista como para o usuário final, pois se as informações forem incompletas, ambíguas ou ininteligíveis, a aplicação deixará de satisfazer as necessidades do usuário (PRESSMAN, 1995).

A importância da *interface* de um programa de computador é avaliada por Heckel (1993), como sendo a parte que o projetista deva dar muita atenção, pois não deve preocupar-se somente com a aparência da *interface* e sim com a funcionalidade da mesma, pois se a preocupação for maior com a aparência do que com a funcionalidade, o que se pode obter é a frustração do usuário.

Conforme Rezende (1999), a elaboração de *interfaces* é uma atividade multidisciplinar que aplica conhecimentos derivados da psicologia e da tecnologia para projetar uma *HCI* de alta qualidade.

2.6.3 MODELOS DE PROJETO DE *INTERFACES*

Pressman (1995) relata que quatro modelos entram em cena quando uma *HCI* (*Human-Computer Interface*) vai ser projetada, a figura 9 apresenta os quatro modelos que serão descritos na seqüência.

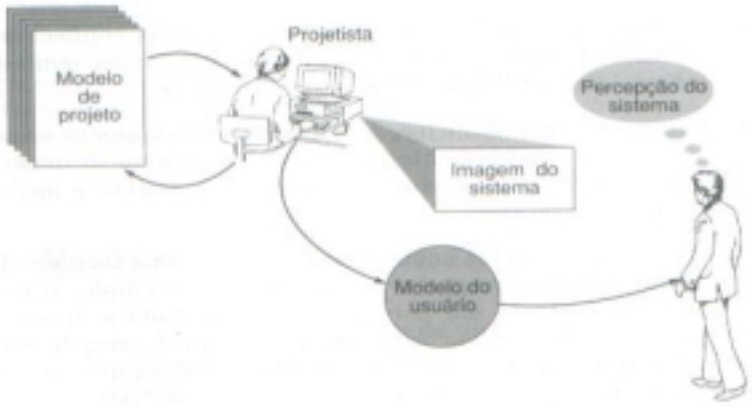


Figura 9 [Pressman (1995)]: Modelos

- *Modelo de projeto*: através do projeto de todo o sistema que incorpora representações procedimentais, arquiteturais e de dados do *software*, pode-se estabelecer certas restrições que ajudem a definir o usuário do sistema e assim começar a projetar a *interface*;
- *Modelo de usuário*: este descreve o perfil dos usuários finais, como idade, sexo, educação e outros. Além disso, os usuários podem ser classificados como *principiantes*, *intermitentes* e *instruídos* sobre o funcionamento do sistema;
- *Percepção do sistema*: é a imagem que o usuário carrega em sua mente, ou seja, é o conhecimento que o usuário tem de algum aplicativo;
- *Imagem do sistema*: é o que o usuário vê na tela, ou seja, o que realmente será usado pelo mesmo. Quando a imagem do sistema e a percepção do sistema são coincidentes, os usuários geralmente sentem-se à vontade com o sistema e usam-no efetivamente.

2.6.4 *WIMP* - (*Windows, Icons, Menus e Pointers*)

O estilo de interação *WIMP*, que é a abreviatura para Janelas, Ícones, Menus e Apontadores, permite a manipulação através de componentes de interação virtuais denominado de *widgets*. Este estilo é implementado com o auxílio das tecnologias de *interfaces* gráficas, que proporcionam o desenho de janelas e o controle de entradas através do teclado e do *mouse* em cada uma destas janelas. Os *softwares* de *interfaces* que implementam estes estilos permitem a construção de ícones que suportam a interação através do *mouse*, comportando-se como dispositivos virtuais de interação.

O *WIMP* não deve ser considerado um único estilo, mas a junção de uma tecnologia de *hardware* e *software*, associado aos conceitos de janelas e de *widgets* que permite a

implementação de vários estilos. Nas *interfaces X Windows* é possível ter os estilos de menus, manipulação direta, preenchimento de formulário e linguagem de comandos. *WIMP* pode ser considerado um modelo ou um *framework* de *interface* apoiado pela tecnologia de *interfaces* gráficas (NORMAN, 1986; PRESSMAN, 1995).

A grande vantagem é que as diversas *interface*s construídas obedecendo a um padrão serão consistentes umas com as outras. Entretanto, o uso de determinado padrão não garante que a *interface* tenha alta usabilidade.

Existem diversas maneiras pelas quais o usuário pode interagir com o sistema e diversos tipos de *interfaces* que podem viabilizar este modelo de interação, a seguir é citado outros estilos.

Linguagens de comandos

A *interface* baseada em linguagens de comandos, proporciona ao usuário a possibilidade de enviar instruções diretamente ao sistema através de comandos específicos. As linguagens de comandos foram o primeiro estilo de interação a ser usado amplamente. Este estilo caracteriza-se por possibilitar ao usuário construir comandos através do teclado (*hardware* da *interface*) que devem ser interpretados pelo *software* da *interface* para que funções específicas da aplicação sejam ativadas.

Menus

Nas *interfaces* orientadas por menus, o conjunto de comandos de funções oferecido pela aplicação é mostrado ao usuário através da tela e cabe ao usuário selecionar uma delas através do *mouse*, de teclas alfanuméricas ou de teclas especiais. Como as funções e a maneira de acioná-las estão visíveis na forma de opções para o usuário selecionar, existe uma demanda maior pelo processo de reconhecimento ao invés de recordação como no caso das *interfaces* baseadas em comandos.

Nesta parte do trabalho foi abordado a parte de *interface* com o usuário, citando-se algumas definições e tipos de *interfaces*. A seguir, será apresentado algumas considerações sobre qualidade de *software*.

2.7 QUALIDADE DE *SOFTWARE*

A engenharia de *software* é a responsável pelo controle da qualidade, fazendo com que o sistema atenda a todos os requisitos e atributos, assumindo assim o papel crítico na produção dos sistemas. A garantia de qualidade de *software* (*Software Quality Assurance* –

*SQ*A) é uma atividade que deve ser aplicada ao longo de todo o processo de desenvolvimento, envolvendo revisões técnicas formais, múltiplas fases de teste, controle da documentação de *software* e das mudanças, procedimentos para garantir a adequação aos padrões e mecanismos de medição e divulgação (PRESSMAN, 1995).

Um dos métodos mais modernos que visam o aumento da qualidade de *software* é o modelo proposto pelo *Software Engineering Institute* (SEI) da *Carnegie Mellon University* conhecido como *Capability Maturity Model* (CMM) que visa a melhoria da qualidade de serviços de desenvolvimento de aplicativos de *software*.

Segundo Gomes (2003), *software* de qualidade é fácil de usar, funciona corretamente, é de fácil manutenção e mantém a integridade dos dados em falhas do ambiente ou outras fora do seu controle. No entanto, em sua grande maioria, os *softwares* requerem conhecimentos técnicos especiais na sua utilização, são difíceis de alterar para modificar função existente ou implementar novos processos ou facilidades, e, para desespero de seus usuários, apresentam falhas sem um prévio aviso e não preservam a integridade dos dados.

A autora ainda relata os custos resultantes dos defeitos ou erros provocados por falha de *softwares*, tanto para desenvolvedores quanto para usuários. O *bug* do milênio, causado pelos erros que os computadores teriam ao confundir o ano 2000 com o ano 1900, consumiu bilhões de dólares para evitar um colapso mundial. Este é um exemplo recente e dimensiona o quanto se depende das máquinas e de seus *softwares*.

Para Rezende (1999), um *software* tem qualidade quando está adequado à empresa, ao cliente e/ou usuário e atende a padrões de qualidade predefinidos. Após estar pronto, o mesmo terá qualidade se gerar informações com qualidade, ou seja, adequada, útil, precisa, clara e oportuna ao usuário.

2.7.1 QUALIDADE DO *SOFTWARE* NA VISÃO DO USUÁRIO

Para os usuários interessa o uso do aplicativo, o seu desempenho e os efeitos que o seu uso possa produzir a ele. Eles não valorizam conhecer aspectos internos do *software* ou como o mesmo foi desenvolvido.

O que realmente interessa ao usuário é que o *software* desempenhe as funções para as quais foi projetado da melhor maneira possível e com segurança. Sendo assim o mesmo deve manter a integridade no caso de falhas, ser portável, seguro e principalmente fácil de usar (GOMES, 2003).

2.7.2 FATORES DE QUALIDADE DE *SOFTWARES*

Gomes (2003) cita que um grande problema que a engenharia de *software* tem é a dificuldade de medir a qualidade de *software*. A qualidade de um dispositivo mecânico é freqüentemente medida em termos de tempo médio entre suas falhas, que é uma medida da capacidade de o dispositivo suportar desgaste. O *software* não se desgasta, portanto tal método de medição de qualidade não pode ser aproveitado.

Pressman (1995) define a qualidade de *software* como sendo a "conformidade aos requisitos de desempenho e de funcionalidade que foram explicitamente definidos, aos padrões de desenvolvimento documentados e às características implícitas que são esperadas por todo *software* desenvolvido por profissionais". O autor cita ainda os onze fatores que diferenciam os sistemas quanto à qualidade de *software*. O quadro 1 visa ilustrar estes fatores.

Fator	Característica
Manutenibilidade	Pode ser concertado?
Flexibilidade	Pode ser mudado?
Capacidade de teste	Pode-se testá-lo?
Portabilidade	Pode ser usado em outra máquina?
Reusabilidade	Pode-se reutilizar parte do *software*?
Interoperabilidade	Pode-se compor uma *interface* com outro sistema?
Corretitude	Faz aquilo que se quer?
Confiabilidade	Comporta-se com precisão o tempo todo?
Eficiência	Rodará em um *hardware* tão bem quanto possível?
Interidade	Ele é seguro?
Usabilidade	Foi projetado para o usuário?

Quadro 1 [Pressman (1995)]: Fatores de Qualidade.

A ISO/IEC 9126 (NBR 13596) fornece um modelo de propósito geral o qual define seis amplas categorias de características de qualidade de *software* que são, por sua vez, subdivididas em subcaracterísticas. O quadro 2 ilustra estas categorias.

CARACTERÍSTICAS	SUBCARACTERÍSTICAS	SIGNIFICADO
Funcionalidade O conjunto de funções satisfazem as necessidades explícitas e implícitas para a finalidade a que se destina o produto?	Adequação	Propõe-se a fazer o que é apropriado?
	Acurácia	Gera resultados corretos ou conforme acordados?
	Interoperabilidade	É capaz de interagir com os sistemas especificados?
	Segurança de acesso	Evita acesso não autorizado, acidental ou deliberado a programas e dados?
	Conformidade	Está de acordo com normas e convenções previstas em leis e descrições similares?
Confiabilidade O desempenho se mantém ao longo do tempo e em condições estabelecidas?	Maturidade	Com que freqüência apresenta falhas?
	Tolerância a falhas	Ocorrendo falhas como ele reage?
	Recuperabilidade	É capaz de recuperar dados após uma falha?
Usabilidade É fácil utilizar o software?	Inteligibilidade	É fácil entender os conceitos utilizados?
	Apreensibilidade	É fácil aprender a usar?
	Operacionalidade	É fácil de operar e controlar a operação?
Eficiência Os recursos e os tempos utilizados são compatíveis com o nível de desempenho requerido para o produto?	Comportamento em relação ao tempo	Qual é o tempo de resposta e de processamento?
	Comportamento em relação aos recursos	Quanto recurso utiliza?
Manutenibilidade Há facilidade para correções, atualizações e alterações?	Analisabilidade	É fácil encontrar uma falha quando ocorre?
	Modificabilidade	É fácil modificar e remover defeitos?
	Estabilidade	Há grandes riscos de *bugs* quando se faz alterações?
	Testabilidade	É fácil testar quando se faz alterações?
Portabilidade É possível utilizar o produto em diversas plataformas com pequeno esforço de adaptação?	Adaptabilidade	É fácil adaptar a outros ambientes sem aplicar outras ações ou meios além dos fornecidos para esta finalidade no software considerado?
	Capacidade para ser instalado	É fácil instalar em outros ambientes?
	Capacidade para substituir	É fácil substituir por outro software?
	Conformidade	Está de acordo com padrões ou convenções de portabilidade?

Quadro 2 [Gomes (2003)]: Características de Qualidade de *Software*.

O modelo proposto pela ISO/IEC 9126 (NBR 13596) tem por objetivo servir de referência básica na avaliação de produto de *software*. Além de ter força de norma internacional, ela cobre os aspectos mais importantes para qualquer produto de *software*.

Nesta seção deu-se ênfase a qualidade de *software*, o assunto relativo a testes de *software* é abordado na seção seguinte.

2.8 TESTE DE *SOFTWARE*

Apresenta-se nesta seção os métodos para a realização de testes de *software*, embora menos eficazes que as revisões para a remoção de erros, os testes são indispensáveis para remover os erros que ainda escapam das revisões e para avaliar-se o grau de qualidade de um produto e de seus componentes.

2.8.1 ATIVIDADES DE TESTES

A atividade de testes é uma etapa crítica para o desenvolvimento de *software*, freqüentemente, a atividade de testes insere tantos erros em um produto quanto a própria implementação. Por outro lado, o custo para correção de um erro na fase de manutenção é de sessenta a cem vezes maior que o custo para corrigi-lo durante o desenvolvimento (PRESSMAN, 1995).

Embora as revisões técnicas sejam mais eficazes na detecção de erros, os testes são importantes para complementar estas revisões e aferir o nível de qualidade do *software*. A realização de testes é, quase sempre, limitada por restrições de cronograma e orçamento. É importante que os testes sejam bem planejados e desenhados, para conseguir-se o melhor proveito possível dos recursos alocados para eles. “Um bom caso de teste é aquele que tem uma elevada probabilidade de revelar um erro ainda não descoberto. Um teste bem sucedido é aquele que revela um erro ainda não descoberto” (PRESSMAN, 1995).

Várias estratégias de testes podem ser implementadas para assegurar que o *software* esteja de acordo com suas especificações e livre de erros. Teste de unidade, teste de integração, teste de sistema, teste de instalação e teste de aceitação são exemplos de técnicas que podem ser utilizadas, segundo Von Mayrhauser (1990). Os mais conhecidos são: o *alpha-test*, no qual o *software* é testado num ambiente controlado por alguns usuários e na presença dos desenvolvedores; e o *beta-test*, no qual o *software* é testado por um grupo maior de usuários, que se propõem a dar um *feedback* aos desenvolvedores, caso alguma irregularidade seja encontrada.

Para Rezende (1999), a atividade de testes é de fundamental importância no desenvolvimento de sistemas, pois ela pode vir a garantir o sucesso do *software*, sendo assim, se todos os requisitos funcionais forem testados, provavelmente o cliente estará satisfeito.

Um objetivo central de toda a metodologia dos testes é maximizar a cobertura destes. Deseja-se conseguir detectar a maior quantidade possível de defeitos que não foram apanhados pelas revisões, dentro de dados limites de custo e prazos (JALOTE, 1997).

Pressman (1995), alude quanto as atividades de teste que:

- Se a atividade de teste for conduzida com sucesso, ela descobrirá erros no *software*;
- A atividade de teste não pode mostrar a ausência de *bugs*; ela só pode, mostrar se defeitos de *software* estão presentes;
- Se erros graves forem encontrados com regularidade, a qualidade e a confiabilidade do *software* são suspeitas;
- Se erros facilmente corrigíveis forem encontrados a qualidade e a confiabilidade do *software* estão aceitáveis ou os testes são inadequados para revelar erros graves;
- Se não for encontrado erro, a configuração de teste não foi suficientemente elaborada e erros estão escondidos no *software*.

2.8.2 MÉTODOS DE TESTES

Existe basicamente duas maneiras de se construírem testes:

- *Método da caixa branca*: tem por objetivo determinar defeitos na estrutura interna do produto, através do desenho de testes que exercitem suficientemente os possíveis caminhos de execução. Os testes de unidade são geralmente de caixa branca;
- *Método da caixa preta*: tem por objetivo determinar se os requisitos foram, totalmente ou parcialmente satisfeitos pelo produto, como erros de *interface*, erros de inicialização e término, dentre outros.

Os testes de caixa preta não verificam como ocorre o processamento, mas apenas os resultados produzidos. Os testes de integração, que são técnicas sistemáticas para a construção da estrutura do programa, realizando-se, ao mesmo tempo, testes para descobrir erros associados a *interfaces*, estes testes geralmente misturam testes de caixa preta e de caixa branca (PRESSMAN, 1995).

Esta seção abordou testes e métodos de testes de *software*, a seguir é abordado o assunto sobre manutenção de *software*.

2.9 MANUTENÇÃO

Segundo Pressman (1995) a manutenção de *software* usualmente consome mais de 60% do custo no ciclo de vida de um *software*, apesar de ser a última fase do processo de ES. Isso ocorre devido ao fato dos programadores, freqüentemente, serem negligentes durante as fases anteriores a implementação do *software*.

Dessa forma, para se obter uma freqüência menor de manutenção, deve-se utilizar extensamente os conceitos que a engenharia de *software* propõe para desenvolvimento de

aplicativos, a qual garantirá um projeto adequado e escalável para futuras modificações. Durante a manutenção, são realizadas atividades corretivas, adaptativas e preventivas, segundo Pressman (1995). A manutenção de um sistema de *software* deve ser feita de maneira criteriosa, para evitar que alterações inseridas venham a prejudicar o funcionamento do sistema e implique em erros que possam causar sérios danos, principalmente aos resultados, como por exemplo, na disponibilização de informações incorretas.

Segundo Sommerville (2003), a manutenção de *software* é o processo geral de modificação de um sistema depois que ele foi colocado em uso. Estas modificações podem ser simples, destinadas a corrigir erros de código, mais extensas, a fim de corrigir erros de projeto, ou significativas, que tem a função de corrigir erros de especificação ou acomodar novos requisitos. O autor ainda ressalta que a manutenção não é um processo para efetuar mudanças maiores na arquitetura do sistema, mas sim para corrigir ou alterar componentes de sistema já existentes.

2.9.1 TIPOS DE MANUTENÇÃO

Para Pressman (1995) e Sommerville (2003) existem diferentes tipos de manutenção, apesar dos autores usarem nomes um pouco diferentes para classificá-las versam sobre o mesmo princípio.

Manutenção para reparar os defeitos no software

Em geral, é a correção de erros de codificação, um processo relativamente barato. Erros de projeto e de especificação de requisitos são muito dispendiosos de se corrigir, sendo assim, muitas vezes é melhor refazer o projeto do que tentar corrigir os mesmos.

Manutenção para adaptar o software a um ambiente operacional diferente

Esse tipo de manutenção é necessário quando algum aspecto do sistema é modificado, como *hardware*, sistema operacional, dentre outras. A aplicação deve ser modificada, para que seja adaptada, a fim de atender tais mudanças.

Manutenção para fazer acréscimo à funcionalidade do sistema

Esse tipo de manutenção é necessário quando os requisitos de sistema são modificados, em resposta a mudanças organizacionais ou de negócios.

De acordo com Sommerville (2003), não há uma distinção entre esses diferentes tipos de manutenção. Pois os defeitos de um *software* podem ser revelados porque um sistema foi utilizado de maneira imprevista, e a melhor maneira de reparar estes defeitos pode ser

acrescentar nova funcionalidade, a fim de ajudar os usuários com o sistema. O autor ainda cita que (p. 518):

> embora esses diferentes tipos de manutenção sejam reconhecidos, algumas vezes, lhes é dado nomes diferentes. A manutenção corretiva é universalmente utilizada para se referir à manutenção para o reparo de defeito. Contudo a manutenção adaptativa, as vezes, significa adaptação a um novo ambiente e, outras vezes, significa adaptar o *software* a novos requisitos. A manutenção evolutiva, às vezes, significa aperfeiçoar o *software* implementando novos requisitos e, em outros casos, se refere a manter a funcionalidade do sistema.

Neste tópico abordou-se a manutenção e suas fases, a pesar de ser o último tópico da ES é um dos mais importantes na implantação de sistemas de *software*. A seguir será abordado o assunto referente a banco de dados.

2.10 SISTEMA DE BANCO DE DADOS

Um dos recursos mais utilizados pelos sistemas de gerenciamento de informações são os sistemas de banco de dados. Esta seção enfatiza alguns conceitos sobre estes sistemas.

2.10.1 CONCEITOS

Para Navathe (2002), um sistema de banco de dados consiste em uma coleção de dados-relacionados e um conjunto de programas para acessá-los. O autor ainda cita que um sistema de banco de dados é uma coleção de programas que permitem ao usuário criar e manter o banco de dados. O sistema de banco de dados é um *software* de finalidade genérica que facilita o processo de definição, construção e manipulação de BD para várias aplicações.

Segundo Date (1991) os sistemas de bancos de dados são basicamente um sistema de manutenção de registros por computador, ou seja, um sistema que objetiva manter as informações e torná-las disponíveis quando solicitadas. As vantagens do uso de um sistema de banco de dados em relação aos métodos tradicionais, baseados em papéis e arquivos, são de ser mais compacto, mais rápido, importar menos serviço braçal e disponibilizar informações certas e atualizadas a qualquer momento. Para Korth (1999), a finalidade de um sistema de banco de dados é simplificar e facilitar o acesso aos dados.

O uso de um sistema de banco de dados se torna necessário quando se deseja obter um ambiente eficiente, para a consulta e armazenamento de informações, sendo este responsável pela manutenção, segurança e integridade dos dados. Os bancos de dados são ferramentas projetadas para gerenciar grandes grupos de informações, o gerenciamento de dados envolve a

definição de estruturas para armazenamento de informações e o desenvolvimento de mecanismos para manipulá-las.

Recorrendo novamente a Navathe (2002), o autor cita que poderia-se construir um próprio conjunto de programas para criar e manter o banco de dados, efetivamente criando-se um próprio *software* de gerenciamento do banco de finalidade específica. Pois o autor afirma, que tanto usando um sistema de banco de dados de finalidade especifica como um de finalidade genérica, ter-se-á que geralmente usar uma quantidade considerável de *software* para manipular o banco de dados. O autor chama de sistema de banco de dados o *software* do SGBD e o BD conjuntamente. A figura 10 ilustra esta definição.

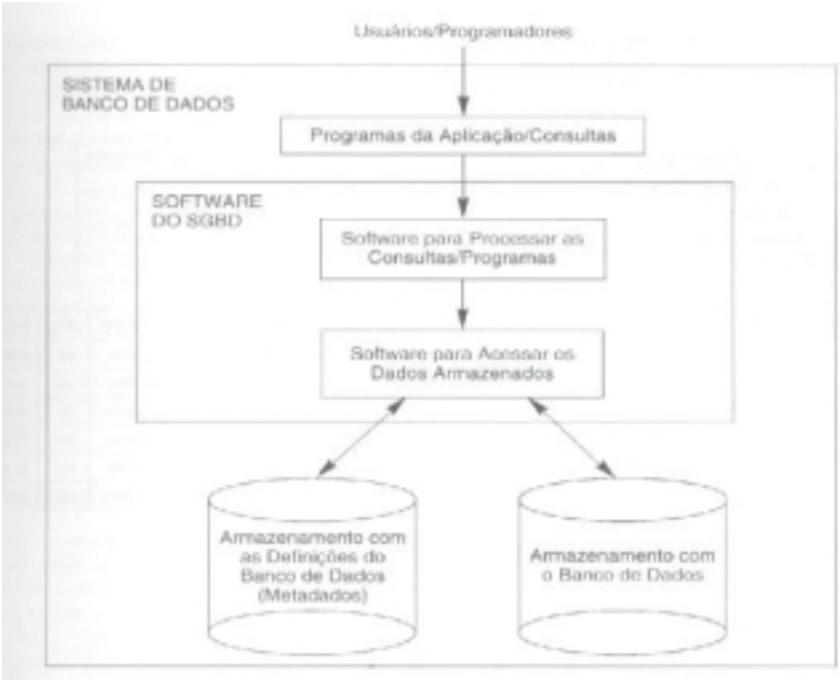


Figura 10 [Navathe (2002)]: Sistema de Banco de Dados

A abstração é uma característica dos bancos de dados, pois permite a independência e a operação dos dados, assim o BD deve ter uma representação conceitual dos dados, não tendo um aprofundamento nos detalhes dos dados armazenados. Navathe (2002), define que a coleta e análise de todos os requisitos e a criação de um esquema conceitual para o BD seriam a etapa de projeto conceitual, “pois o esquema conceitual é uma descrição concisa dos requisitos de dados feitos pelo usuário e inclui descrições detalhadas dos tipos de entidades, relacionamentos e restrições do sistema”.

Heuser (1999), cita que o modelo conceitual é uma descrição do BD de forma independente de implementação em um sistema de banco de dados. O modelo registra que

dados podem aparecer no BD, mas não registra como os mesmos estão armazenados em nível de sistema.

2.10.2 ARQUITETURA DO SISTEMA DE BANCO DE DADOS

Navathe (2002) apresenta como arquitetura de sistema a arquitetura em três camadas, que tem como objetivo separar as aplicações do usuário e o banco de dados. Ela é definida em três níveis, segundo o autor. A figura 11 ilustra a arquitetura de três camadas.

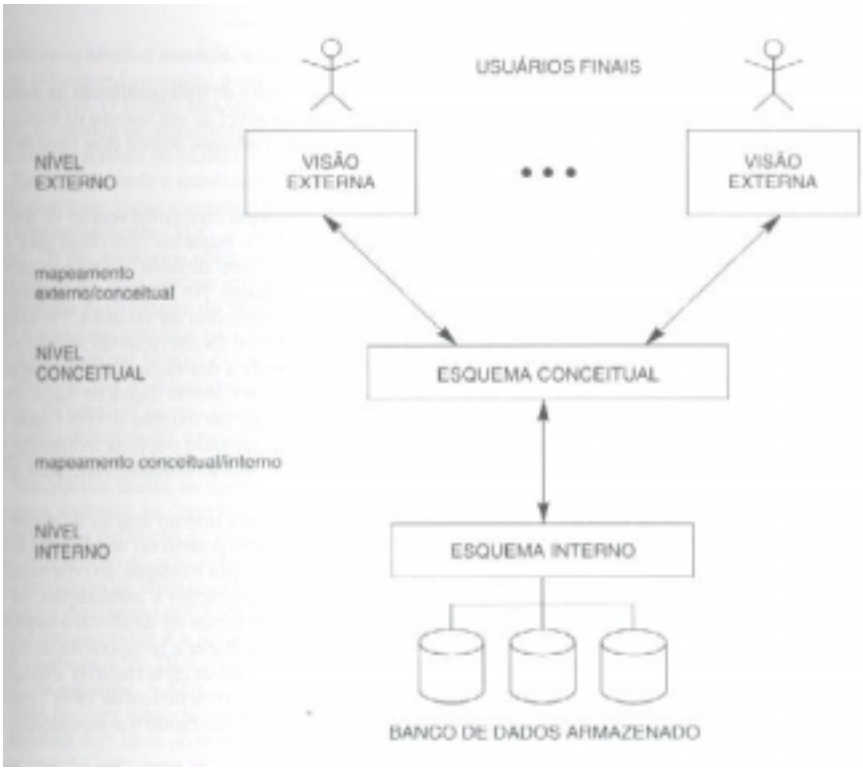


Figura 11 [Navathe (2002)]: Arquitetura de Três Camadas

1. Nível interno: descreve a estrutura de armazenamento físico do BD;
2. Nível conceitual: descreve a estrutura de todo o BD para uma comunidade de usuários;
3. Nível externo ou de visões: mostra aos usuários apenas o que lhes interessa, e esconde o restante do banco daqueles usuários.

A arquitetura de três camadas é uma ferramenta conveniente para o usuário visualizar os níveis de esquema num sistema de banco de dados.

Navathe (2002), define a independência de dados como sendo, a capacidade de alterar o esquema em um nível de um sistema de BD sem ter que alterar o esquema num nível mais elevado. Pode-se definir dois tipos de independência, a independência lógica dos dados em

que se poder alterar o esquema conceitual sem ter de alterar esquemas externos ou programas da aplicação, e a independência física dos dados que é a capacidade de se poder alterar o esquema interno sem precisar alterar os esquemas conceituais (ou externos).

2.10.3 VISÕES DO BANCO DE DADOS

Korth (1999) afirma que, o maior beneficio de um BD é proporcionar ao usuário a visão abstrata dos dados, isto é, o sistema oculta do usuário determinados detalhes referentes a armazenamento e manutenção dos dados.

Navathe (2002) baseia-se, no fato de um sistema de banco de dados geralmente possuir muitos usuários e cada um podendo exigir uma diferente perspectiva ou visão do BD, logo um sistema de banco de dados deve permitir que cada usuário visualize os dados de forma diferente daquela existente previamente no BD. Uma visão consiste de um subconjunto de dados do BD, necessariamente derivados dos existentes no banco de dados, porém estes não deverão estar explicitamente armazenados.

Para Date (1991), banco de dados tem implícito a idéia de abstração das informações, desta forma o autor divide a arquitetura de visões em três níveis gerais:

- *Visão interna*: é a vista pelo responsável pela manutenção e desenvolvimento do sistema;
- *Visão conceitual*: é a que é vista pelo analista de desenvolvimento e pelo administrador das bases de dados. Aqui há a definição das normas e procedimentos para manipulação dos dados. Na visão conceitual, existem duas linguagens de operação que são:

  a) Linguagem de Definição dos Dados (DDL): define as aplicações, arquivos e campos que irão compor o banco de dados (comandos de criação e atualização da estrutura dos campos dos arquivos);

  b) Linguagem de Manipulação dos Dados (DML): define os comandos de manipulação e operação dos dados (comandos de consulta e atualização dos dados dos arquivos);
- *Visão externa*: é a que é vista pelo usuário que opera os sistemas aplicativos, através de *interfaces* desenvolvidas pelo analista (programas), buscando o atendimento e suas necessidades.

2.10.4 ESTRUTURA DO SISTEMA DE BANCO DE DADOS

Um sistema de banco de dados está dividido em módulos específicos, de modo a atender a todas as funções do sistema. Os componentes funcionais do sistema de banco de dados podem ser divididos pelos componentes de processamento de consultas e pelos componentes de administração de memória (KORTH, 1999).

Os componentes de processamento de consulta são:

- *Compilador DML* (*Data Manipulation Language*): transforma a solicitação do usuário em uma solicitação equivalente, mas mais eficiente, buscando, assim, uma boa estratégia para a execução da consulta;
- *Pré-compilador para comandos DML* (*Data Manipulation Language*): interage com o compilador DML de modo a gerar o código apropriado;
- *Interpretador DDL* (*Data Definition Language*): interpreta os comandos DDL e registra-os em um conjunto de tabelas que contêm metadados;
- *Componentes para tratamento de consultas*: executam as instruções geradas pelo compilador DML;

Os componentes de administração de dados são:

- *Gerenciamento de autorizações e integridade*: testam o cumprimento das regras de integridade e a permissão ao usuário no acesso aos dados;
- *Gerenciamento de transações*: garante que o BD permanecerá consistente no caso de falhas no sistema e que as transações serão executadas sem conflitos em seus procedimentos;
- *Administração de arquivos*: gerencia a alocação de espaço no armazenamento em disco;
- *Administração de buffer*: responsável pela intermediação de dados do disco para a memória principal.

A figura 12 ilustra estes componentes e a conexão entre eles.

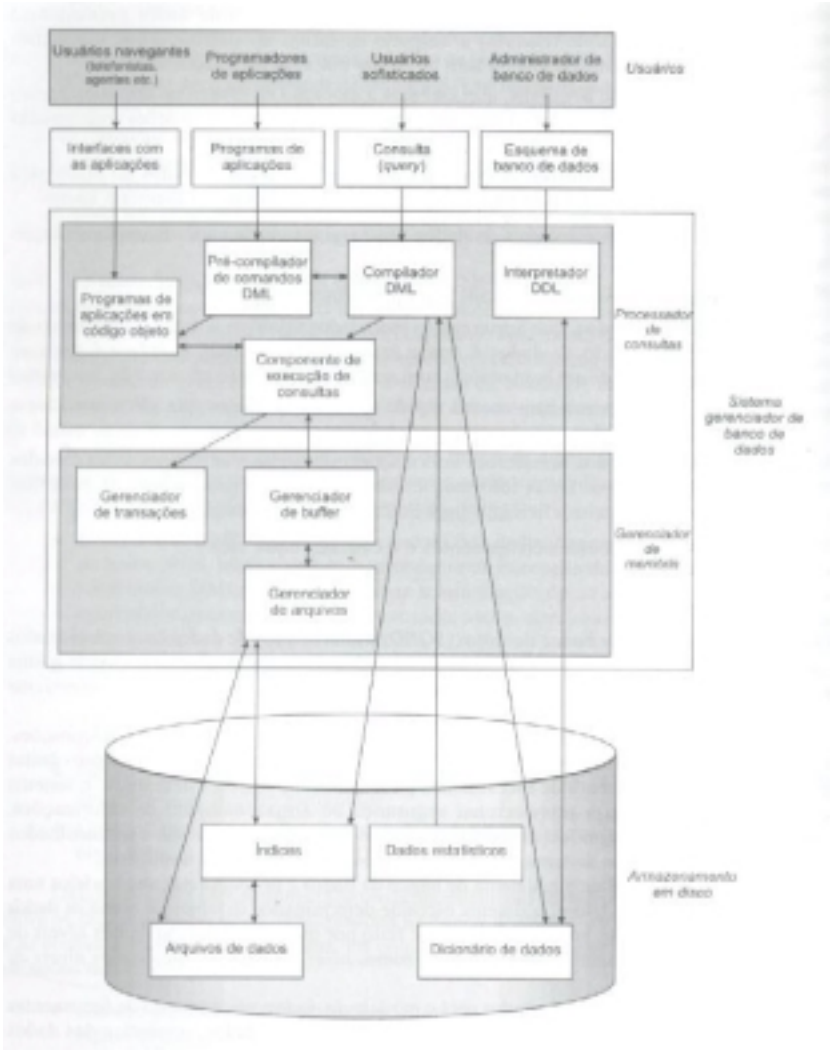


Figura 12 [Korth (1999)]: Estrutura do Sistema

2.10.5 OBJETIVOS DE UM SISTEMA DE BANCO DE DADOS

Segundo as definições de Korth (1999) e Navathe (2002), esta seção visa descrever algumas das vantagens do sistema de banco de dados perante arquitetura tradicional.

Apesar de Korth (1999) tratar esta parte como "*Objetivos de um sistema de banco de dados*" e Navathe (2002) como "*Vantagens do uso de banco de dados*", os autores referenciam as mesmas questões.

- *Controle de redundâncias*: a redundância consiste no armazenamento de uma mesma informação em locais diferentes, provocando inconsistências. Em um banco de dados as informações só se encontram armazenadas em um único local, não existindo duplicação descontrolada dos dados. Quando existem replicações dos dados, estas são decorrentes do processo de armazenagem típica do ambiente Cliente-Servidor, totalmente sob controle do sistema;

• *Eliminação de inconsistências*: através do armazenamento da informação em um único local com acesso descentralizado e, sendo compartilhada a vários sistemas, os usuários estarão utilizando uma informação confiável. A inconsistência ocorre quando um mesmo campo tem valores diferentes em sistemas diferentes. Exemplo, o estado civil de uma pessoa é solteiro em um sistema e casado em outro. Isto ocorre porque esta pessoa atualizou o campo em um sistema e não o atualizou em outro. Quando o dado é armazenado em um único local e compartilhado pelos sistemas, este problema não ocorre;

• *Compartilhamento dos dados*: permite a utilização simultânea e segura de um dado, por mais de uma aplicação ou usuário, independente da operação que esteja sendo realizada. Deve ser observado apenas o processo de atualização concorrente, para não gerar erros de processamento (atualizar simultaneamente o mesmo campo do mesmo registro). Os aplicativos são por natureza multiusuário;

• *Restrições de segurança*: define para cada usuário o nível de acesso a ele concedido (leitura, leitura e gravação ou sem acesso) ao arquivo e/ou campo. Este recurso impede que pessoas não autorizadas utilizem ou atualizem um determinado arquivo ou campo;

• *Padronização dos dados*: permite que os campos armazenados na base de dados sejam padronizados segundo um determinado formato de armazenamento (padronização de tabela, conteúdo de campos, etc) e ao nome de variáveis seguindo critérios padrões pré-estabelecido pela empresa. Ex. Para o campo "Sexo" somente será permitido armazenamento dos conteúdos "M" ou "F";

• *Manutenção de integridade*: exige que os conteúdos dos dados armazenadas no banco de dados possuam valores coerentes ao objetivo do campo, não permitindo que valores absurdos sejam cadastrados. Exemplo: Um funcionário que faça no mês 500 horas extras, ou um aluno que tenha nascido no ano de 1860;

• *Evitar necessidades conflitantes*: representa a capacidade que o administrador de banco de dados deve ter para solucionar "prioridades sempre altas" de todos os sistemas, tendo ele que avaliar a real necessidade de cada sistema para a empresa para priorizar a sua implantação;

• *Independência dos dados*: representa a forma física de armazenamento dos dados no BD e a recuperação das informações pelos programas de aplicação. Esta recuperação deverá ser totalmente independente da maneira com que os dados estão fisicamente

armazenados. Quando um programa retira ou inclui dados, o sistema, compacta-os para que haja um menor consumo de espaço no disco. Este conhecimento do formato de armazenamento do campo é totalmente transparente para o usuário. A independência dos dados permite os seguintes recursos:

a) Os programas de aplicação definem apenas os campos que serão utilizados independente da estrutura interna dos arquivos;

b) Quando há inclusão de novos campos no arquivo, será feita manutenção apenas nos programas que utilizam esses campos, não sendo necessário mexer nos demais programas. Nos sistemas tradicionais este tipo de operação requer a alteração no esquema de todos os programas do sistema que utilizam o arquivo.

Backups: capacidade do sistema de recuperar falhas de *hardware* e *software*, através da existência de arquivos de "pré-imagem" ou de outros recursos automáticos, exigindo minimamente a intervenção de pessoal técnico.

Interfaceamento: poder disponibilizar formas de acesso gráfico, em linguagem natural, em SQL ou ainda via menus de acesso, não sendo uma "caixa-preta" somente sendo passível de ser acessada por aplicações.

Navathe (2002), cita ainda outras implicações a respeito de sistemas de banco de dados que podem beneficiar a maior parte dos usuários, que são: potencial para impor padrões, tempo reduzido para o desenvolvimento de aplicações, flexibilidade, disponibilidade de informações atualizadas para todos os usuários e economias de escalas. O autor aponta ainda algumas situações em que o uso de banco de dados pode não ser o ideal, que são:

- Quando os investimentos iniciais são muito elevados em *hardware*, *software* e treinamento;
- A generalidade que um sistema de banco de dados fornece para definir e processar dados;
- Custos adicionais para fornecer segurança, controle de concorrência, recuperação e funções de integridade.

Outros problemas podem surgir se não tiver projetistas de banco de dados competentes, ou se as aplicações dos sistemas de banco de dados não forem implementadas de forma correta. Desta forma seria mais apropriado utilizar arquivos regulares sob as seguintes circunstâncias:

- O banco de dados e as aplicações são simples, bem definidos e não se espera que sejam alterados;

- Existem exigências de tempo real rigorosas para alguns programas que podem não ser cumpridas devido aos custos adicionais (*overhead*) do sistema;
- O acesso aos dados por múltiplos usuários não é necessário.

2.10.6 MODELO RELACIONAL

Conforme as definições de Heuser (1999), Korth (1999) e Navathe (2002) um banco de dados relacional consiste em uma coleção de tabelas, cada qual designada por um único nome. O modelo relacional é claramente baseado no conceito de matrizes, onde as chamadas linhas seriam os registros e as colunas seriam os campos. Os nomes das tabelas e dos campos são de fundamental importância para a compreensão entre o que se esta armazenando, onde se esta armazenando e qual a relação existente entre os dados armazenados.

As relações não podem ser duplicadas, a ordem de entrada de informações no banco de dados não deverá ter qualquer importância para as relações no que concerne ao seu tratamento. Conceitualmente, entidades individuais e relacionamentos são distintos, mas numa perspectiva de banco de dados, a diferença entre eles precisa ser expressa em termos de seus atributos. Chama-se chave primária ao atributo que define um registro, este campo será a chave de acesso ao registro da tabela.

2.10.7 INTEGRANDO BANCO DE DADOS E WEB

Atualmente existem diversas aplicações de banco de dados que podem ser transportadas para o ambiente *Web* e várias aplicações *We*b que podem usar bancos de dados, como mecanismos mais eficientes para o armazenamento de informações. Neste sentido, a *Web* está passando rapidamente da condição de troca irrestrita de documentos, como projetado inicialmente, para a condição de se tornar uma plataforma de desenvolvimento de inúmeras aplicações baseadas em banco de dados.

Além disso, vêm crescendo as soluções intranet integradas a banco de dados: uma rede corporativa de computadores baseada nos padrões de comunicação da internet pública e da *Web*, cujos dados são armazenados e gerenciados por sistemas de banco de dados.

De modo geral, pode-se enumerar as seguintes vantagens da integração *Web* e banco de dados:

Os mecanismos de armazenamento e acesso de dados da *Web* não são muito eficientes nem padronizados. Muitas aplicações baseadas na *Web* armazenam dados em arquivos convencionais e, na maioria das vezes, usam linguagens de propósito geral como C, C++ ou PERL, para armazenamento e acesso aos dados. Quando o volume de informações previsto

cresce muito, estas linguagens podem se tornar inadequadas. Como se sabe, os banco de dados são providos de mecanismos muito mais eficientes como estruturas de índices automáticos, execução otimizada de consultas, autorização de acesso, suporte a transações em ambientes multiusuário (como consistência e controle de concorrência), mecanismos de recuperação de dados quanto à falhas, facilidades para modelar e organizar tipos de dados complexos e relacionamento entre eles (segundo o modelo relacional, por exemplo), além de usarem linguagens específicas de consulta que independem de aplicação (como por exemplo, SQL – *Structured Query Language*).

Existe atualmente uma grande massa de dados residindo e sendo gerenciada por SGBDs. A *Web* passa a proporcionar a possibilidade de dispor estas informações a um número ilimitado de usuários em redes locais ou remotas. A *Web* também vem possibilitando o desenvolvimento de aplicações baseadas em SGBDs capazes de conectar membros de uma organização sob uma rede diferente, local ou remotamente.

Um dos grandes problemas dos atuais SGBDs é que o desenvolvimento de aplicações dirigidas a usuário é limitada e muito dependente das ferramentas proprietárias distribuídas pelos fabricantes. Em particular, há o problema das interfaces do usuário para banco de dados. A *Web* proporciona uma interface gráfica relativamente amistosa para aplicações de banco de dados multiplataforma.

A *Web* está possibilitando o desenvolvimento e expansão de novas aplicações baseadas em SGBDs. São exemplos dessas aplicações compras eletrônicas (aplicação da idéia de comércio eletrônico) via internet e bibliotecas virtuais.

A figura 13 mostra o ambiente de integração de banco de dados com *Web*.

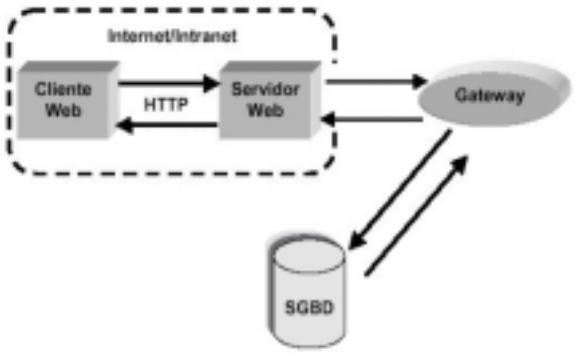


Figura 13: Ambiente de Integração de Banco de Dados e *Web*

O cliente e o servidor *Web*, podem estar dentro dos limites de uma Intranet sendo somente usados por membros de uma organização, ou fazer parte de uma Internet global. O *gateway* é o *software* responsável pelo gerenciamento da comunicação e por proporcionar serviços de aplicação entre o servidor *Web* e o servidor de banco de dados. Ele reside tipicamente no lado do servidor *Web*, e é composto por um ou mais programas *scripts*. Ao servidor de banco de dados cabe a tarefa de gerenciar os dados armazenados no banco de dados. O servidor *Web* tem a tarefa de receber os pedidos dos clientes e retornar os dados enviados pelo *gateway* para serem exibidos pelo cliente *Web* ao usuário final.

Os *gateways* podem ser implementados através de uma grande variedade de soluções. A solução mais utilizada é implementada através da interface CGI. Outras soluções incluem APIs de servidores *Web* ou através de linguagens de programação como java, php e dll's.

Baseado na figura anterior é apresentado a seguir o fluxo de dados de uma aplicação no ambiente de integração:

Inicialmente o cliente *Web* solicita um pedido ao servidor *Web* via protocolo HTTP.

O servidor *Web* dispara um processo no *gateway* enviando os parâmetros recebidos do cliente *Web*.

O *gateway* trata os parâmetros recebidos, formula o comando SQL, abre uma conexão com o SGBD e espera pela resposta.

O SGBD atende a solicitação e retorna os dados ao *gateway*.

O *gateway* trata os dados recebidos e os repassa ao servidor *Web* num formato que o cliente *Web* entenda.

O servidor *Web* retorna os dados enviados pelo *gateway* ao cliente *Web*.

O cliente *Web* identifica o formato dos dados e os exibe ao usuário cliente.

Algumas considerações a respeito das linguagens de programação utilizadas em SGBD:

Os benefícios proporcionados pela integração *Web* banco de dados também são significativamente afetados pelo tipo de linguagem que pode ser usado para o desenvolvimento. Algumas linguagens vêm crescendo a sua utilização para aplicações *Web* banco de dados como, Java, C, C++, JavaScript, VbScript, Php e PERL.

Muitos fabricantes de sistemas de banco de dados continuam a usar suas linguagens proprietárias (PL da Oracle, por exemplo) como linguagens de desenvolvimento para aplicações *Web* banco de dados. Toda essa variedade de linguagens cria problemas de portabilidade e complexidade da aplicação *Web* banco de dados. Porém hoje em dia as linguagens *open source* tem ganhado muito espaço em aplicações para a *Web* banco de dados.

2.10.7.1 HTTP

HTTP é o protocolo utilizado na *Web* para transferência de informações. Essas informações ou dados transferidos podem ser: texto não estruturado, hipertextos, imagens, ou qualquer outro tipo de dado.

No protocolo *HTTP* existe dois itens distintos: um deles é um conjunto de solicitações dos *browsers* aos servidores, o outro item é um conjunto de respostas que retornam dos servidores para os *browsers*.

Para transferência de informações ou dados, o protocolo *HTTP* utiliza uma conexão *TCP* para cada transferência de objeto. A conexão permanece ativa enquanto durar a transferência de informação entre os pontos cliente e servidor.

Em relação as solicitações, protocolo *HTTP* aceita dois tipos: simples e completas. Uma solicitação simples é apenas uma linha *GET* que identifica a página desejada, sem cabeçalhos, sem códigos entre outros (TANEMBAUM, 1997).

Já as solicitações completas, são indicadas pela presença da versão do protocolo de linha de solicitação *GET*. Estas solicitações podem apresentar várias linhas e seguidas de uma linha em branco para indicar o fim de uma solicitação.

Apesar de ter sido projetado para a utilização na *Web*, o *HTTP* foi, intencionalmente, criado para ser mais genérico do que o necessário com vistas a futuras aplicações orientadas a objetos. Por esse motivo, a primeira palavra na linha de solicitação completa é o nome do método (comando) a ser executado na página *Web* (ou objeto genérico). Quando você acessa

objetos genéricos, outros métodos específicos dos objetos podem se tornar disponíveis. Os nomes fazem distinção entre maiúsculas/minúsculas; portanto *GET* é um método válido, mas *get* não é (TANEMBAUM, 1997).

A solicitação *GET* nada mais é do que um pedido feito ao servidor *Web*, para que o cliente solicitante possa ler a página pedida.

Apesar da habilidade do *HTTP* em negociar formatos, a *WWW* possui uma linguagem básica de intercâmbio de hipertextos, denominada *HTML* (*Hypertext Markup Language*). A linguagem *HTML* é compatível com o padrão *SGML* (*Standard General Markup Language*). Um documento *HTML* válido, acompanhado das declarações *SGML* que definem o *HTML DTD* (*Document Type Definition*), pode ser analisado sintaticamente por um *parser SGML*.

2.10.8 SQL (*Structured Query Language)*

Embora se fale que a linguagem *SQL* é uma *linguagem de consulta*, essa linguagem possui outras capacidades além de realizar consultas em um banco de dados. A linguagem *SQL* possui recursos para definição da estrutura de dados, para modificar dados no banco de dados e recursos para especificar restrições de segurança e integridade.

A versão original da linguagem *SQL* foi desenvolvida no laboratório de pesquisa da IBM. Esta linguagem, originalmente chamada *SEQUEL*, foi implementada como parte do projeto *System* R no início dos anos 70. A linguagem *SEQUEL* evoluiu e seu nome foi mudado para *SQL* (*Strutured Query Language*). Numerosos sistemas gerenciadores de banco de dados suportam a linguagem *SQL*. A *SQL* estabeleceu-se como a linguagem padrão de banco de dados relacional. Embora existam diversas versões, com algumas diferenças entre elas, a estrutura da *SQL* se mantém inalterada desde a sua criação (KORTH, 1999).

A *SQL* apresenta uma série de comandos que permitem a definição dos dados, chamada de DDL (*Data Definition Language*), composta entre outros pelos comandos *Create*, que é destinado à criação do banco de dados, das tabelas que o compõe, além das relações existentes entre as tabelas. Como exemplo de comandos da classe DDL temos os comandos *Create*, *Alter* e *Drop*.

Os comandos da série DML (*Data Manipulation Language*), destinados a consultas, inserções, exclusões e alterações em um ou mais registros de uma ou mais tabelas de maneira simultânea. Como exemplo de comandos da classe DML temos os comandos *Select*, *Insert*, *Update* e *Delete*.

Uma subclasse de comandos DML, a DCL (*Data Control Language*), dispõe de comandos de controle como *Grant* e *Revoke*.

A linguagem *SQL* tem como grande virtude sua capacidade de gerenciar índices, sem a necessidade de controle individualizado de índice corrente, algo muito comum nas linguagens de manipulação de dados do tipo registro a registro. Outra característica muito importante disponível em *SQL* é sua capacidade de construção de visões, que são formas de visualizarmos os dados na forma de listagens independente das tabelas e organização lógica dos dados.

Outra característica interessante na linguagem *SQL* é a capacidade que dispomos de cancelar uma série de atualizações ou de as gravarmos, depois de iniciarmos uma seqüência de atualizações. Os comandos *Commit* e *Rollback* são responsáveis por estas facilidades.

Deve-se notar que a linguagem *SQL* consegue implementar estas soluções, somente pelo fato de estar baseada em Banco de Dados, que garantem por si mesmo a integridade das relações existentes entre as tabelas e seus índices.

2.11 CONSIDERAÇÕES FINAIS

Conforme exposto na fundamentação teórica, não existe método certo ou errado no processo de *software*. Diferentes processos de *software* organizam as atividades de diferentes maneiras. Diferentes organizações usam diferentes processos para produzir o mesmo tipo de produto. E, diferentes tipos de produtos podem ser produzidos utilizando o mesmo tipo de processo.

Desta forma, os modelos apresentados servem como guia para o processo de desenvolvimento, ainda vale ressaltar que o gerenciamento de projetos é uma tarefa complexa que envolve a análise de diversos fatores, tais como tempo, recursos e tarefa.

Sendo que, o principal objetivo de um gerenciamento de projeto é a busca pela qualidade do produto e produtividade, vale ressaltar que o processo de *software* compreende o processo de desenvolvimento de *software*, o processo de gerenciamento de projeto e as medições, não esquecendo-se questões relevantes à *interface*, qualidade de *software* e banco de dados, no desenvolvimento de projetos de *softwares*, ênfases deste capítulo. A metodologia e a descrição do planejamento do projeto serão ênfases do capítulo seguinte.

3. DESCRIÇÃO DO PROJETO

Este capítulo apresenta a descrição do mapeamento do processo de orientação, bem como, os passos que foram seguidos para o levantamento dos dados e a definição do modelo de protótipo que é proposto neste trabalho.

3.1 MAPEAMENTO DO PROCESSO DE ORIENTAÇÃO

Esta seção vem apresentar o mapeamento do processo de orientação, com base na fundamentação teórica, bem como, suas fases e respectivas funções em cada uma das mesmas. Este mapeamento tem como referência as entrevistas efetuadas e as respectivas informações prestadas pelas pessoas que foram entrevistadas.

3.1.1 RESULTADOS DA PESQUISA COM OS ORIENTADORES

A lista de verificação que foi usada para as entrevistas contém treze perguntas, esta seção visa relatar as questões que considera-se mais importante no mapeamento do processo de orientação, sendo assim, aqui é relatado alguns dos resultados e respostas obtidas pelos entrevistados.

a) Atividades envolvidas no processo de orientação: segundo os orientadores, as atividades mais importantes que envolvem este processo são a seleção do aluno, que é feito uma análise em seu perfil de formação e currículo, a definição do tema do trabalho do aluno, a metodologia a ser seguida durante o processo, indicação de bibliografia, aprendizado de como desenvolver a pesquisa, definição dos cronogramas que o aluno terá que cumprir, revisões no documento escrito pelo aluno, abrir canais de contato para o aluno e um acompanhamento a todos os passos do aluno referente ao seu processo de formação.

b) Critérios usados para aceitar um aluno como orientado: esta é uma das partes do processo em que cada orientador tem os seus próprios critérios para tal, desta forma, são relatados aqui os critérios comuns as pessoas que foram entrevistadas, que são o currículo do aluno, identificação com a área escolhida, formação básica na área, disponibilidade de tempo, afinidade pessoal, produção científica e possibilidade de se manter financeiramente durante o curso.

c) A definição do tema a ser desenvolvido: os temas são definidos com base na área de pesquisa do orientador, perfil do orientado ou os encaixando em projetos anteriores, sendo que alguns relataram que os temas são propostos numa lista dentro da área de conhecimento do orientador e o aluno escolhe o que deseja trabalhar.

d) O acompanhamento das atividades desenvolvidas pelos alunos: esta pergunta também obteve respostas muito diferentes. É citado aqui as mais relevantes para o processo de orientação como, reuniões periódicas com os orientados, leitura do material enviado pelo aluno, *papers* enviados pelo aluno em cada fase de desenvolvimento do trabalho e os cronogramas do aluno.

e) O que usa para arquivar e controlar o andamento do aluno: os orientadores nesta parte usam várias formas para tal controle. Porém, nenhum usa uma ferramenta específica para controlar este andamento. As respostas foram o uso de editores de textos, anotações em cadernos, pastas de e-mail em seus correios eletrônicos e anotações em papéis após as reuniões com seus orientados.

f) Forma de tirar dúvidas de seus orientados: nesta questão a semelhança é bastante comum, pois quase todos as tiram pessoalmente as mais importantes, por e-mail e por telefone as menos importantes.

Tendo como referência as respostas e conversas efetuadas durante as entrevistas, pode-se dizer que os resultados obtidos através do processo de mapeamento que foi utilizado para entender-se e definir o que seria necessário para desenvolver a ferramenta proposta foram positivos. Pois com o mapeamento, conseguiu-se entender como funciona o processo de orientação e atingir os objetivos esperados. Ressalta-se ainda que, com o entendimento de como funciona o processo de orientação, ficou mais fácil planejar e implementar a ferramenta proposta, pois todos os entrevistados relataram claramente o que desejavam que a ferramenta contivesse.

Sendo assim, conforme os relatos dos entrevistados os módulos trabalho, e-mail, cronograma e arquivos são muito importantes para a ferramenta. Pois através do módulo trabalho o orientador pode ter um controle de toda a vida acadêmica do orientado, com o módulo e-mail é possível manter o aluno informado das atividades que o mesmo tem que desenvolver no curso e lembrá-lo de seus compromissos agendados com o orientador e desta forma não ter que o orientador estar se preocupando com estas datas, pois a ferramenta irá informar o aluno de suas atividades, já o módulo cronograma irá manter um controle das tarefas que o aluno deve entregar a seu orientador com as datas de início e término, sendo o mesmo avisado das mesmas por e-mail, livrando também o orientador de ter que cobrar do aluno tais tarefas. Com o módulo arquivos o orientador terá na pasta do aluno todos os arquivos que o mesmo lhe enviar, com um controle de data que o mesmo foi recebido e de data de leitura do mesmo, facilitando assim para o orientador o controle destes documentos, sem contar que esta parte pode ser usada como uma base bibliográfica no módulo *Web*.

Esta seção apresentou os resultados da pesquisa com os orientadores. A seção seguinte apresenta o processo de orientação.

3.1.2 O PROCESSO DE ORIENTAÇÃO

Com base nas entrevistas elaborou-se o mapeamento do processo de orientação. A figura 14 descreve sucintamente o processo de orientação, a seguir é feita uma descrição das atividades realizadas em cada uma das fases do processo de orientação, segundo os entrevistados.

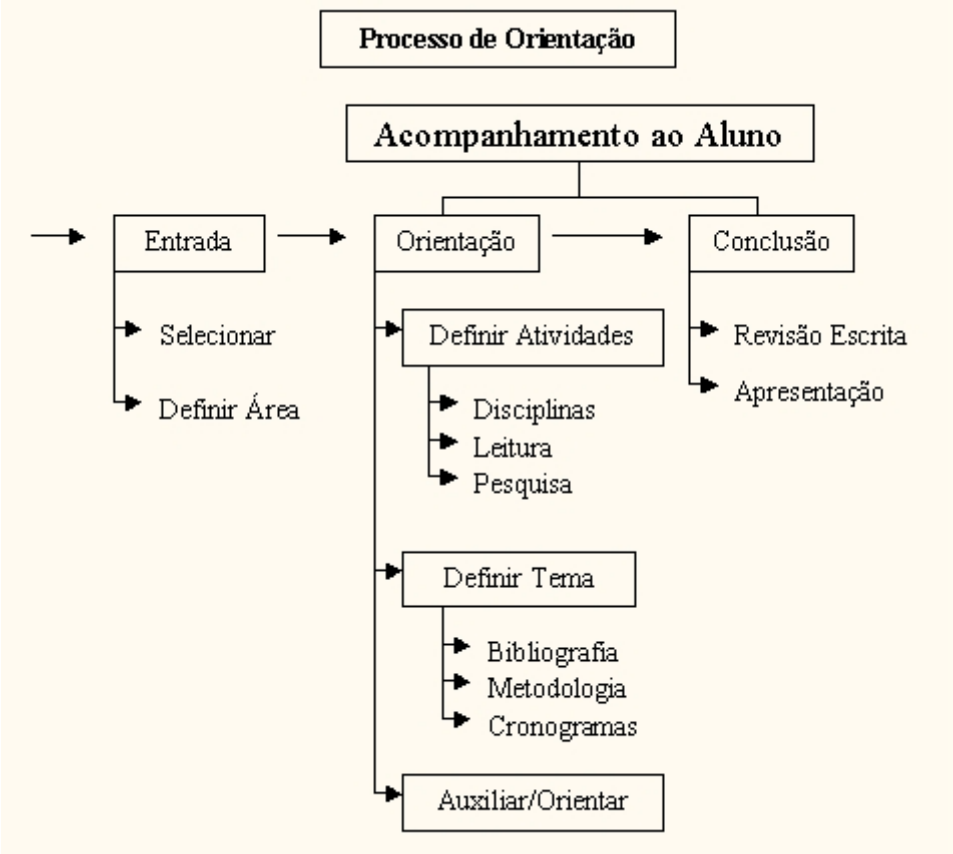


Figura 14 – Processo de Orientação

Descrição das Fases do Processo de Orientação

- *Entrada*: seleção do aluno com base nos requisitos do curso e de cada orientador. Após a aprovação do aluno é definida a área de pesquisa do orientando conforme o orientador que o selecionou e suas atividades iniciam no curso.

- *Orientação*: nesta fase são planejadas as atividades que o aluno terá que cumprir dentro do período de orientação, como disciplinas que deverá cursar para ajudar-lhe a ter uma melhor formação ao desenvolver seu trabalho, leitura e elaboração de artigos, pesquisas sobre a área escolhida. Há também a definição do tema de sua pesquisa e da bibliografia que lhe é recomendada e os cronogramas que terá que cumprir.

  É nesta fase que entra realmente a orientação por parte do professor que o selecionou, pois é em toda esta fase que o orientador irá acompanhar e auxiliar o aluno realmente, dando-lhe dicas e tirando possíveis dúvidas que irão surgir no decorrer do processo.

- *Conclusão*: nesta fase o orientador fará a revisão da escrita de seu aluno e indicará onde o mesmo terá que melhorar e fazer as alterações finais para encaminhá-lo para a defesa de seu trabalho que é a apresentação do mesmo.

- *Acompanhamento ao aluno*: este processo se inicia a partir do momento em que o aluno ingressa no curso, até o momento em que entrega seu trabalho para a secretaria após ter feito as correções indicadas pela banca examinadora e assim estará apto a receber seu diploma.

A seção seguinte apresenta o projeto conceitual e ferramentas usadas para o projeto e desenvolvimento do sistema.

3.2 PROJETO CONCEITUAL

Nesta fase do objeto, tem-se uma visão global do sistema, pois são irrelevantes para o desenvolvedor, os detalhes da estrutura do banco de dados. Deve-se observar quais dados são relevantes e como eles estão relacionados. Para realização desta etapa, foi adotado o MER, que será descrito a seguir no item 3.2.3.

3.2.1 FERRAMENTAS PARA O DESENVOLVIMENTO DO SISTEMA

Esta seção descreve as ferramentas que foram usadas para o projeto e implementação do aplicativo que é proposto neste trabalho.

Para elaborar-se os diagramas de classe na versão *desktop* e o de caso de uso na especificação *Web*, utilizou-se à ferramenta *CASE Rational Rose* 4.0, desenvolvida pela *Rational Software Corporation*. Já, para a modelagem do banco de dados, usou-se a ferramenta de diagramação *ERwin* 4.0, desenvolvida pela *Computer Associates*.

O banco de dados usado para servir de *back-end* para o aplicativo foi o Interbase 6.0, uma ferramenta *Open Source*, desenvolvida pela *Borland Software Corporation*. E para o desenvolvimento da interface de acesso a este banco, ou seja, o *front-end* do aplicativo usou-se a ferramenta de programação Delphi 5.0, desenvolvida pela *Inprise Corporation*.

Algumas Considerações Sobre UML (*Unified Modeling Language*)

O objetivo da *UML* é prover uma linguagem padrão que permita modelar um sistema. De acordo com Larman (2000), com a *UML* é possível descrever eficazmente requisitos de *software*, caracterizar a arquitetura (lógica e física) de um sistema, focalizar na arquitetura em vez da implementação e direcionar programadores, aumentando a produtividade e diminuindo os riscos.

Por outro lado, a *UML* não se restringe a diagramas, ela apresenta uma série de conceitos e recursos que facilitam a identificação de objetos e classes, associando-os aos requisitos do sistema, bem como oferece formas de planejar e gerenciar projetos baseados nestes requisitos (FURLAN, 1998).

A *UML* apresenta os seguintes diagramas que, em conjunto, modelam todo o sistema (FURLAN, 1998):

- *Diagrama de Classe*: utilizado para representar as diversas classes de objetos do sistema, seus atributos e operações, bem como a associação entre cada uma delas (herança, generalização, composição, agregação, dentre outras.);
- *Diagrama de Caso de Uso*: usado para demonstrar o relacionamento entre atores e casos de uso;
- *Diagramas de Seqüência*: tipo de diagrama de interação que apresenta a interação de seqüência de tempo dos objetos que participam na interação;
- *Diagrama de Colaboração*: tipo de diagrama de interação que mostra uma interação dinâmica de um caso de uso e seus objetos relacionados;
- *Diagrama de Estado*: utilizado para demonstrar as seqüências de estados que um objeto assume em sua vida, em função do seu uso no sistema;
- *Diagrama de Atividade*: tipo de diagrama de estado no qual a maioria dos estados são ações. Descreve o fluxo interno de uma operação;
- *Diagrama de Componente*: usado para representar os diversos componentes dos sistemas e suas dependências;

- *Diagrama de Implantação*: utilizado para demonstrar elementos de configuração de processamento *run-time*.

O uso de um tipo ou outro de diagrama depende, muitas vezes, do grau de detalhamento necessário para o desenvolvimento do sistema. Os diagramas de classe, de casos de uso e de seqüência são os mais utilizados. Há ainda diversos outros conceitos, dentre tantos que a *UML* possui, fugindo ao escopo desta dissertação a explicação de cada um destes. Para um bom uso da *UML*, recomenda-se a utilização de ferramentas *CASE*, que ajudam na construção dos diagramas, dando suporte automatizado à notação. Hoje a maioria das ferramentas *CASE* já suportam a *UML*, sendo algumas totalmente direcionadas a esta.

Nesta seção fez-se a apresentação das ferramentas usadas para o desenvolvimento do sistema. A seguir será abordada a análise estuturada.

3.2.2 ANÁLISE ESTRUTURADA

A análise estruturada de sistemas compõe-se de um conjunto de técnicas e ferramentas, nascida do sucesso da programação e do projeto estruturado. Seu conceito fundamental é a construção de um modelo lógico (não físico) de um sistema, utilizando técnicas gráficas capazes de levar usuários, analistas e projetistas a formarem um quadro claro e geral do sistema e de como suas partes se encaixam para atender as necessidades daqueles que dele precisam (DEMARCO, 1989).

A análise de sistemas é a parte mais difícil no desenvolvimento de um sistema de processamento de dados. São vários fatores que, combinados, tornam essa tarefa bastante difícil e exigente.

A identificação da necessidade do sistema é uma tarefa que envolve o trabalho conjunto do analista e do cliente e/ou usuário final, sendo o ponto de partida para o desenvolvimento de um sistema computacional.

A questão econômica também é o ponto essencial a ser analisado em grande parte dos sistemas. Permite verificar se o sistema pode ser desenvolvido de modo que a relação custo/benefício esteja dentro de padrões aceitáveis de desenvolvimento. A viabilidade técnica é, sem dúvida, o ponto com maior dificuldade de análise, a qual permite obter informações sobre a possibilidade de se construir um sistema considerando os requisitos funcionais e de desempenho. O analista deve combinar o que é atualmente possível e o que vale a pena ser feito para a empresa, em relação à maneira como é administrada.

Uma vez definida a base de dados e as funções de um sistema, é possível criar um modelo que represente o sistema, incluindo seus diferentes elementos e as suas inter-relações.

Do ponto de vista clássico, qualquer sistema computacional pode ser modelado na forma de uma transformação de informações (arquitetura - entrada - processamento - saída) (DEMARCO, 1989).

3.2.3 MODELO ENTIDADE-RELACIONAMENTOS

O esquema de banco de dados, em nível conceitual, foi definido com base no modelo entidade-relacionamento (ER) criado por Peter Chen em 1976, sendo este baseado na teoria relacional de Cood (HEUSER, 1999).

Este modelo baseia-se na representação do sistema por um conjunto de objetos do mundo real (entidades) e por relacionamentos entre estas entidades. O modelo ER nada mais é do que uma visão de uma dada realidade e está baseado no relacionamento entre entidades, e que cada entidade ou relacionamento pode possuir atributos qualificadores.

Este modelo foi desenvolvido para otimizar o projeto de banco de dados, pois tem como característica principal à facilidade de projetar esquemas de banco de dados. O modelo ER tornou-se o modelo de maior capacidade semântica para representação de bancos de dados em nível conceitual, permitindo que seja feita a especificação do esquema da organização, que representa toda a estrutura lógica do banco de dados. O modelo ER é extremamente útil para mapear, sobre um esquema conceitual, o significado e interações das empresas reais. Devido a essa utilidade, muitas das ferramentas de projeto foram concebidas para o modelo ER (NAVATHE, 2002).

A figura 15 expressa o MER do banco de dados proposto a implementar-se neste trabalho, o dicionário de dados da figura será apresentado no apêndice B deste trabalho. Este modelo foi desenvolvido com base nas informações coletadas através de conversas com os entrevistados, e para tal modelagem, usou-se a ferramenta de diagramação ErWin citada no item 3.2.1, que é própria para este tipo de modelagem em banco de dados.

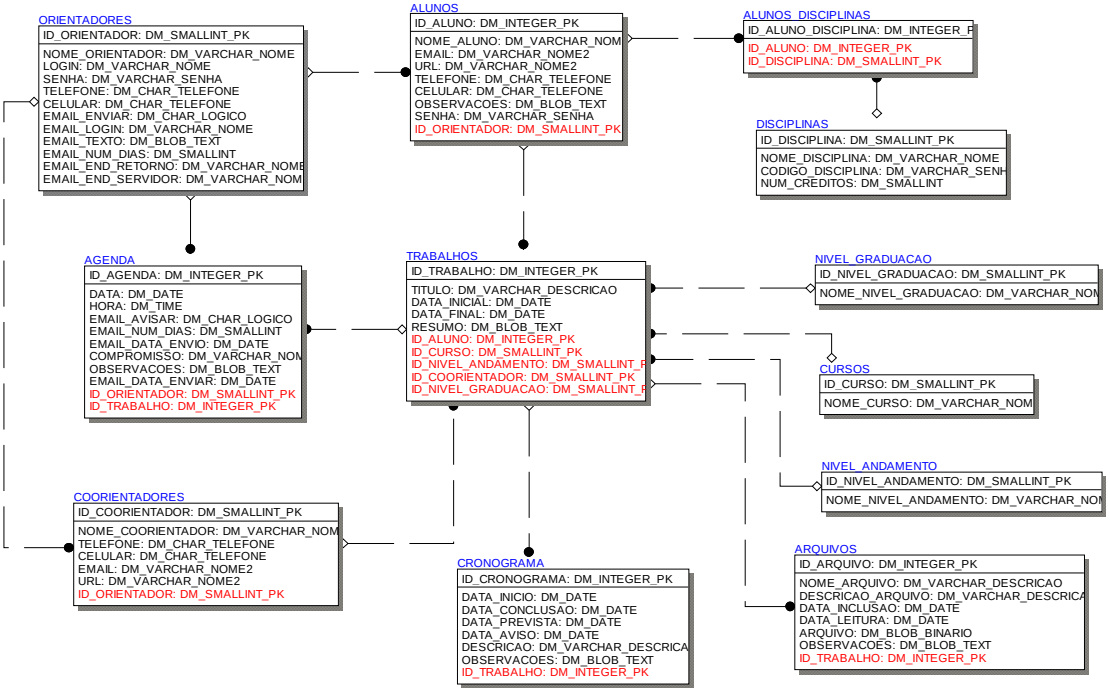


Figura 15 – Modelo Entidade-Relacionamentos

Esta seção de ênfase ao modelo entidade-relacionamentos. A seção seguinte apresenta a especificação do sistema proposto.

3.3 ESPECIFICAÇÃO DO SISTEMA

Esta seção apresenta uma breve especificação do sistema proposto, pois a descrição detalhada do mesmo, será abordada no capítulo 4. A seguir serão apresentados o diagrama de classe e o diagrama de caso de uso (especificação) do SOA para *desktop* e o diagrama de caso de uso do SOA para *Web*.

3.3.1 DIAGRAMA DE CLASSES DO SOA *DESKTOP*

De acordo com Bezerra (1997), o diagrama de classes expressa de uma forma geral a estrutura estática de um sistema, contém classes e associações entre elas. Uma classe descreve um conjunto de elementos. Uma associação é um conjunto de ligações. Objetos são instâncias das classes. O relacionamento entre as classes é chamado de associação e o relacionamento entre objetos é chamado de ligação. A figura 16 apresenta o diagrama de classes do aplicativo proposto.

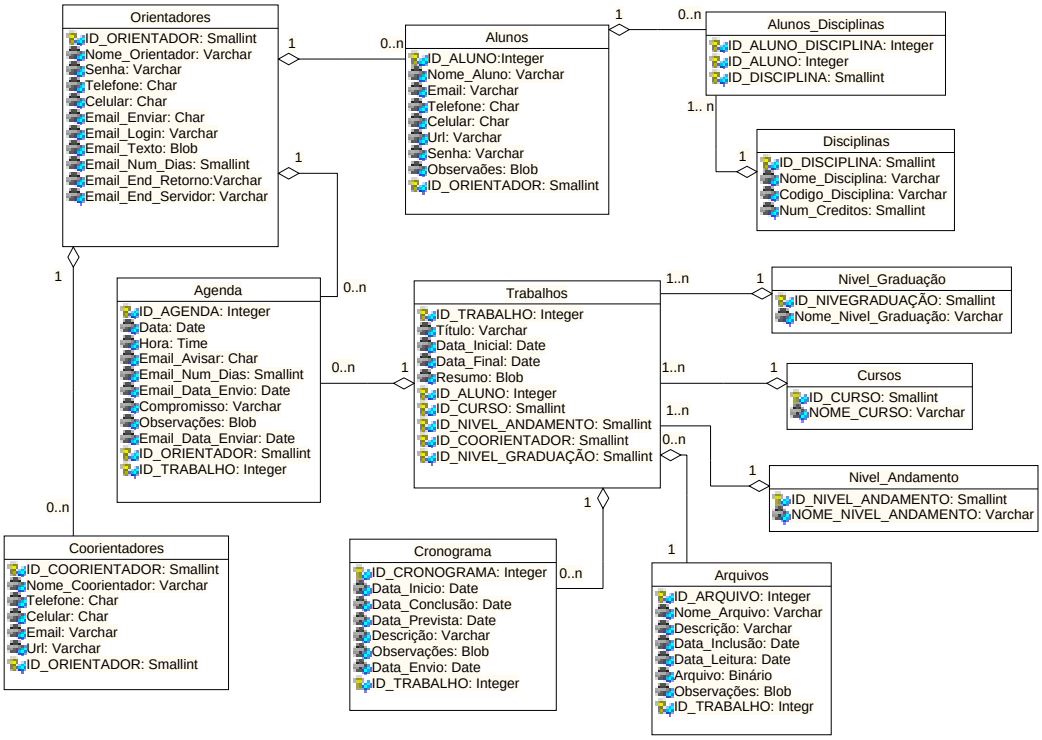


Figura 16: Diagrama de Classes do SOA

3.3.2 DIAGRAMAS DE CASO DE USO DO SOA *DESKTOP*

Neste item é apresentada a interação que os usuários terão com o sistema, sendo assim, utilizou-se o diagrama de *use case* para ilustrar estas seqüências de ações e variações. Pois este tipo de diagrama representa os requisitos funcionais do sistema como um todo, e descreve "o que" o sistema faz, mas não "o como" ele faz. As figuras 17 e 18 ilustram as ações dos usuários no sistema.

SOA *Desktop* Admin

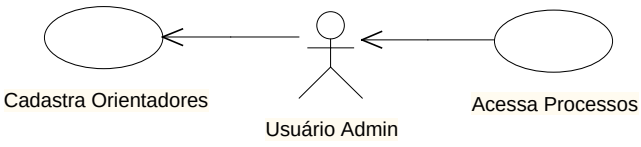


Figura 17: Modelo de Contexto do Admin

SOA *Desktop* Orientador

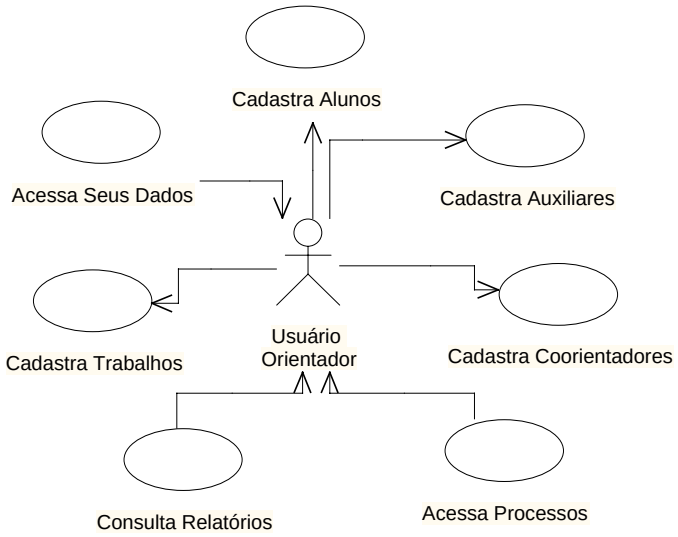


Figura 18: Modelo de Contexto do Orientador

3.3.3 DIAGRAMA DE CASO DE USO DO SOA *WEB*

Nesta seção é apresentado o diagrama de *use case* do SOA *Web*, apesar deste módulo não ser implementado neste trabalho, far-se-á nesta seção uma ilustração dos acessos aos módulos que comporão esta parte. A figura 19 ilustra o diagrama.

Legenda da figura 19:

- Usuário Admin: cadastra orientadores
- Usuário Orientador: cadastra alunos
- Usuário Orientador em Mensagens: envia e recebe a todos seus alunos.
- Usuário Orientador em Disciplinas: cadastra e consulta disciplinas de seus alunos.
- Usuário Orientador em Cronograma: cadastra e consulta o cronograma de seus alunos.
- Usuário Orientador em Arquivos: envia e recebe arquivos a seus alunos ou todos.
- Usuário Aluno em Mensagens: envia e recebe de seu orientador.
- Usuário Aluno em Disciplinas: cadastra e consulta disciplinas suas.
- Usuário Aluno em Cronograma: só pode consultar seu cronograma de atividades.
- Usuário Aluno em Arquivos: envia só para seu orientador e pode receber de todos.

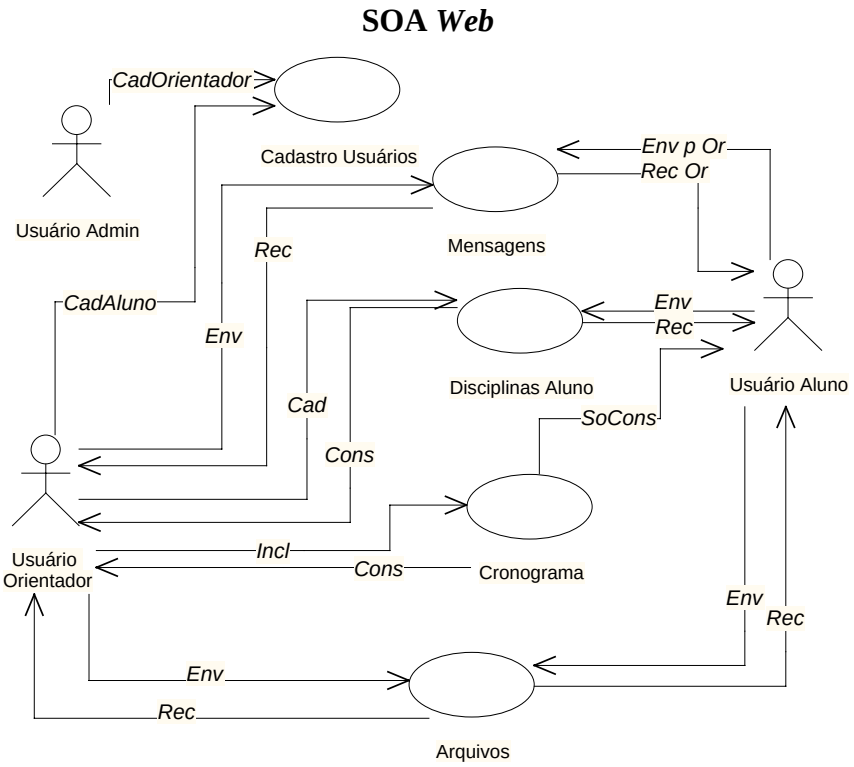


Figura 19: Modelo de Contexto do SOA *Web*

3.4 CONSIDERAÇÕES FINAIS

Através do mapeamento de processo é que se pôde compreender o que seria necessário para o projeto que se está propondo implementar para este trabalho, pois normalmente usa-se apenas a modelagem tradicional para tal. Neste trabalho, usam-se as duas técnicas, a de modelagem e a de mapeamento, para desta forma se conseguir um número menor de erros de projeto e uma melhora contínua no processo de orientação.

Como é citada anteriormente, a compreensão dos processos é importante, pois é a chave para o sucesso em qualquer negócio, e com o mapeamento foi possível compreender melhor o processo de orientação e suas fases.

A modelagem do BD é a parte mais importante de qualquer projeto, pois é através dela que se tem noção do que realmente será o BD, já que os vários recursos oferecidos pelo modelo ER proporcionam aos projetistas de banco de dados numerosas opções para a escolha da melhor representação do projeto que esta sendo modelado (KORTH, 1999).

Este capítulo deu ênfase a descrição do mapeamento e as técnicas usadas para coleta de dados e ferramentas utilizadas para a modelagem e implementação do protótipo. A descrição do sistema constitui o foco principal do próximo capítulo.

4. A FERRAMENTA COMPUTACIONAL PROPOSTA

Este capítulo apresenta o aplicativo de auxílio a orientadores, implementado a partir das especificações descritas no capítulo 3. Nesta fase é efetuada a codificação dos resultados da análise na linguagem de programação escolhida. É aqui que se pode visualizar os conceitos definidos na etapa de projeto e análise, bem como as definições dos módulos do protótipo que é proposto para este trabalho.

Para a implementação do aplicativo foi utilizada a linguagem de programação Delphi 5.0 em conjunto com o sistema de banco de dados Interbase. Para o detalhamento do sistema, denominado Sistema de Auxílio a Orientadores (SOA), este capítulo foi dividido de acordo com os itens descritos:

Primeiro: Paradigma usado;
Segundo: Fases de desenvolvimento;
Terceiro: Descrição dos módulos;
Quarto: Requisitos de *hardware* e software;
Quinto: Desenvolvimento do protótipo.

4.1 PARADIGMA USADO

O paradigma de desenvolvimento usado para projetar e implementar o sistema, foi o de prototipação, descrito no item 2.3.2. Sendo aplicado o desenvolvimento em módulos do aplicativo.

A escolha por este paradigma deu-se devido à forma de desenvolvimento e a implementação rápida do projeto, pois o uso da prototipação permite que se faça um modelo de projeto inicial e a partir deste se vá incrementando o mesmo, bem como também permite a integração entre os diversos módulos do sistema.

Esta seção apresentou o paradigma usado, a seguir são descritas as fases de desenvolvimento do protótipo.

4.2 FASES DE DESENVOLVIMENTO DO SOA

Inicialmente foram realizadas a análise de requisitos, definição dos módulos funcionais, definição e implementação do banco de dados e a codificação em si, com a criação das telas e do código para cada módulo. Posteriormente foram realizados os testes e

validações dos módulos desenvolvidos, sendo aplicado tanto os testes de caixa branca quanto de caixa preta.

Estas fases de desenvolvimento podem ser classificadas como:

Fase 1: Modelagem e implementação do banco de dados;

Fase 2: Desenvolvimento dos módulos e telas;

Fase 3: Implementação das técnicas de *HCI* e geração do arquivo de ajuda;

Fase 4: Testes e correção de erros de sistema.

4.3 DESCRIÇÃO DOS MÓDULOS

Esta seção apresenta os módulos do protótipo e suas funcionalidades dentro do contexto global do aplicativo. A figura 20 ilustra um modelo geral do sistema, a seguir é feita também uma descrição detalhada dos módulos e apresenta-se as principais características funcionais de cada módulo.

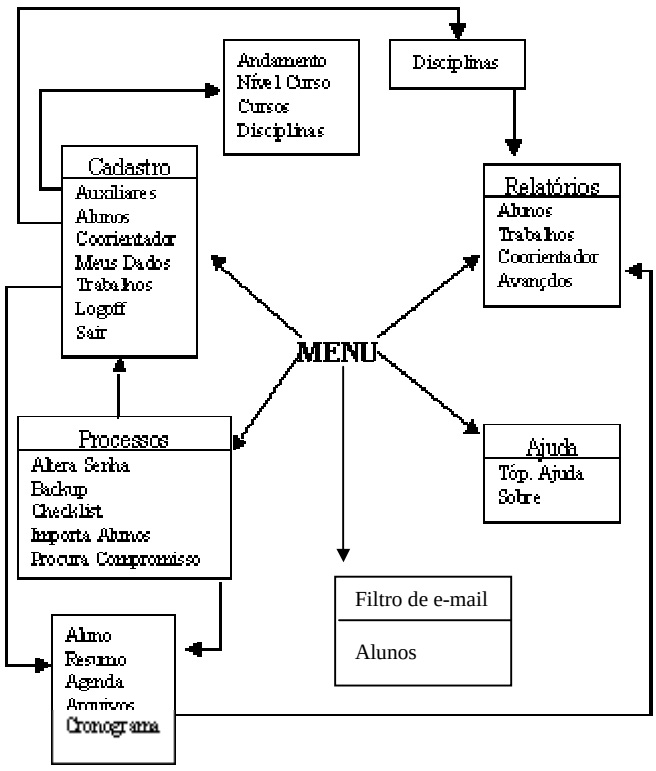


Figura 20: Módulos do Sistema

A seguir é feita a descrição dos principais módulos do aplicativo, que são o módulo de cadastro, processos e relatórios. É feita aqui também, uma breve descrição do módulo trabalhos.

Visando facilitar a compreensão dos módulos que compõem o aplicativo desenvolvido, esta seção é descrita na forma de um tutorial passo-a-passo das etapas envolvidas nos módulos.

Módulo Cadastro

- *Auxiliares*: aqui é feito o cadastro de andamento do aluno, ou seja, em que fase do trabalho o mesmo se encontra, se está em disciplina, em escrita ou defesa. É feito também o cadastro do nível do curso que o aluno freqüenta, se é mestrado, doutorado ou outro.

Este item também permite, o cadastro de cursos e disciplinas que serão usados por outros módulos do sistema. Ressalta-se aqui, que todos estes cadastros ficaram abertos para a inserção do nome que o usuário desejar dar aos itens, desta forma deixando o usuário personalizar os nomes conforme seu uso.

- *Alunos*: este item chama o módulo cadastro do aluno, onde são cadastrados os dados pessoais referentes ao aluno, tendo como campo obrigatório o e-mail do mesmo, bem como as disciplinas que o aluno cursou são cadastradas aqui.
- *Coorientadores*: aqui é feito o cadastro dos coorientadores, sendo os mesmos usados no módulo de cadastro de trabalhos.
- *Meus Dados*: permite ao orientador visualizar e alterar os seus dados pessoais cadastrados.
- *Trabalhos*: este item chama o módulo trabalho, no qual é cadastrados o trabalho do aluno, o resumo de seu trabalho, os compromissos, os arquivos enviados ao orientador e seus cronogramas a cumprir.
- *Logoff*: permite a troca de usuário no sistema sem ter que encerrar o mesmo.

Módulo Processos

- *Alterar Senha*: permite que o usuário altere sua senha no sistema.
- *Backup*: acessa o módulo de *backup* do aplicativo, permitindo efetuar-se uma cópia dos dados em disquete ou em outro computador.
- *Checklist*: permite o acesso do usuário ao *checklist* do programa, pois o mesmo é o responsável pelo envio de mensagens aos orientados do professor logado ao sistema.
- *Importar Alunos*: permite a importação dos dados pessoais dos alunos para o sistema.

- *Procurar Compromisso*: faz uma busca por compromissos agendados, sendo que o mesmo pode ser por título do compromisso ou período do mesmo.

Módulo Relatórios

- *Alunos por Curso*: permite a emissão de relatórios dos alunos por curso escolhido ou todos os cursos.
- *Trabalhos por Curso*: permite a emissão de relatórios de trabalhos por curso ou de todos os cursos.
- *Por Coorientador*: emite o relatório para o coorientador selecionado sobre o andamento do trabalho do aluno ao qual é coorientador.
- *Procura Avançada*: aqui é feita uma pesquisa avançada na base de dados, podendo-se nesta, fazer-se concatenação de vários campos das tabelas. Esta concatenação faz uso de recursos Sql para sua implementação.

Módulo Trabalho

- *Cadastro do Trabalho*: nesta ficha é feito o cadastro do trabalho do aluno, usando-se para este os dados pré-cadastrados nos módulos anteriores.
- *Resumo*: aqui é colocado o resumo do trabalho efetuado pelo aluno.
- *Agenda*: nesta parte são armazenados os compromissos do aluno, bem como os avisos que o mesmo receberá de seus cronogramas e anotações enviadas pelo orientador.
- *Arquivos*: aqui são armazenados os arquivos enviados pelo aluno, os mesmos são marcados com a data de importação e de leitura dos mesmos, podendo o orientador enviar comentários ao aluno quando efetuar a leitura do arquivo em questão.
- *Cronograma*: é nesta ficha que será armazenado o cronograma de tarefas do aluno, tendo a data de início, data prevista de conclusão, data que houve a conclusão e uma descrição do mesmo. Sendo que a data de conclusão é quem irá definir se o aluno deve ser avisado ou não do cronograma.

Esta seção apresentou os módulos principais da ferramenta de auxílio a orientadores, as telas referentes aos módulos são apresentadas e descritas no apêndice D deste trabalho. A seguir é feita uma descrição dos requisitos de sistema para um funcionamento adequado da ferramenta.

4.4 REQUISITOS DE SISTEMA

Nesta seção serão descritos os requisitos mínimos para que se obtenha um bom funcionamento do programa, bem como uma satisfação melhor de uso por parte dos usuários.

4.4.1 REQUISITOS DE *HARDWARE* E *SOFTWARE*

Não há uma definição específica de *hardware* para o funcionamento do programa, sugere-se, no entanto, que a máquina tenha uma configuração mínima de um PC 486, 32 *Mbytes* de memória RAM, e espaço disponível em disco de 30 *Mbytes*. Em máquinas com configurações 'melhores', o aumento da performance do programa será significativamente melhor.

Quanto a softwares, exige-se que o SO em uso na maquina seja o *Windows* 95 ou superior juntamente com gerenciador de banco de dados Interbase.

Aqui foram apresentados os requisitos de sistema para o funcionamento da ferramenta, a seguir é descrito o desenvolvimento da mesma.

4.5 DESENVOLVIMENTO DO PROTÓTIPO

Esta seção apresenta como foi desenvolvido o sistema e as dificuldades enfrentadas para planejar e implementar o protótipo, bem como as principais telas do aplicativo.

4.5.1 CONSIDERAÇÕES SOBRE O DESENVOLVIMENTO

O tempo destinado ao desenvolvimento do projeto foi dividido em partes como segue: primeiro houve um estudo de coleta de dados e planejamento do que uma ferramenta de auxílio a orientadores deveria no mínimo possuir, para desta forma auxiliar os professores no controle do processo de orientação de seus alunos, após este estudo e definição de funções, houve uma pesquisa literária para decidir-se qual o paradigma de desenvolvimento aplicar-se-ia para gerenciar e desenvolver o projeto da ferramenta proposta, bem como as técnicas de engenharia de software que seriam envolvidas neste projeto. Após estas definições iniciou-se o projeto e modelagem do banco de dados que servirá de base para a ferramenta, baseando-se em conceitos e estudos relativos ao desenvolvimento de bancos de dados como consta na fundamentação teórica deste trabalho. Somente após se ter toda a fundamentação teórica revisada e o banco modelado e implementado é que se começou a implementar as telas da ferramenta e a parte de ajuda da mesma, baseando-se sempre em conceitos bibliográficos e técnicas de desenvolvimento de software.

Diversas foram as dificuldades encontradas na fase de codificação do protótipo, entre elas pode-se citar as mudanças que ocorreram do projeto inicial até o momento, tendo que se implementar novos campos também no banco de dados para atender as alterações efetuadas. Outro ponto a ser considerado é o de racionalização do acesso a informação, visto que quanto maior a facilidade que se implementa para o usuário, maior é a complexidade do algoritmo a ser implementado.

Finalmente, a unificação dos módulos, exigiu que algumas alterações fossem realizadas nos mesmos, devido, principalmente às alterações em telas e ao surgimento de novas instâncias de programa que exigiram uma nova codificação e acertos.

4.5.2 CUMPRIMENTO DAS METAS

Os resultados alcançados relativos ao processo de orientação permitem não só a utilização da ferramenta em um nível de gerenciamento diferenciado, baseando-se em níveis de complexidade (questão da interface), bem como possibilita a utilização efetiva destas informações no âmbito da orientação.

Finalmente, o protótipo proposto cumpre seu objetivo ao manipular as principais informações necessárias ao apoio e gerenciamento do processo de orientação, de forma diferenciada e considerando a problemática envolvida no contexto do segmento educacional.

4.5.3 QUANTO AOS RESULTADOS EXPERIMENTAIS

Para os resultados experimentais, a metodologia escolhida foi o teste da caixa preta, onde são realizadas entradas e analisadas as saídas do protótipo.

No decorrer do desenvolvimento dos módulos, testes foram realizados a fim de validar o módulo em desenvolvimento. Inicialmente os testes ignoraram o relacionamento do módulo testado com os demais módulos, sendo validados somente as funções no âmbito interno ao módulo em teste. Em um segundo instante foram verificados os relacionamentos entre o módulo testado e os demais módulos os quais este interaja, sendo então analisadas as instâncias criadas.

Como era de se esperar, falhas foram detectadas, tanto nos testes iniciais (internos) quanto na integração entre os módulos. Dependendo da falha ocorrida sua resolução exigiu a realização de alterações no código. A princípio as falhas encontradas eram tratadas exclusivamente internamente ao módulo, ignorando seu relacionamento com os demais módulos. No instante em que esta falha estivesse corrigida e o módulo estivesse funcionando corretamente, eram então realizados testes onde fossem verificados os relacionamentos entre

os módulos os quais ocorresse alguma interação em nível de informação, tal como descreve Pressman (1995).

O relacionamento entre módulos com compartilhamento de informações exigiu, em alguns casos, a realização de novos testes nos módulos relacionados ao módulo alterado. O fato de trabalhar-se com um modelo de desenvolvimento em módulos possibilitou uma grande coesão entre as informações gerenciadas, ou seja, o relacionamento da informação entre os módulos poderia ser analisado internamente ao código do programa, e não estaria condicionada ao acesso direto à informação.

Desta forma, os erros puderam ser minimizados, visto que o relacionamento funcional entre os diversos módulos estava condicionado às informações pertinentes ao módulo que estivesse sendo analisado. Os procedimentos de testes e validações não foram documentados em detalhes, sendo que as alterações no projeto, tanto em nível de código quanto em nível de estrutura de banco de dados foram diretamente alteradas no projeto lógico.

Finalmente, realizou-se testes sobre o protótipo como um todo, onde a consistência do protótipo exigiu novamente alguns ajustes, devido ao surgimento de instâncias imprevistas, as quais foram corrigidas e testadas novamente.

4.5.4 ARQUIVOS DE AJUDA

Um paradigma relacionado à apresentação de um programa de computador é a questão do treinamento e a ajuda aos usuários.

Diversas técnicas de ajuda são utilizadas por empresas desenvolvedoras de softwares com a finalidade de solucionar a problemática da exposição das potencialidades e funções disponibilizadas pelo programa implementado. A metodologia convencional retratada por MacConell (2000) expõe a utilização de ajudas baseadas em índices, assistentes, tutoriais e frames de auxílio.

No decorrer do desenvolvimento da ferramenta foram utilizadas diversas metodologias de auxílio ao usuário, tais como o *Tooltipbox* (caixa dinâmica que aparece junto ao cursor indicando a finalidade do objeto), arquivo de ajuda e o nome do objeto. No entanto, a racionalização da interface ao usuário está intimamente relacionada ao desenvolvimento de uma tela amigável, do ponto de vista funcional. Assim, a validação do próprio programa está diretamente relacionada a concepção de uma interface amigável, que exija poucos conhecimentos de informática, o que, dispensaria a busca de soluções em metodologias de ajuda mais elaboradas.

As ajudas por tópicos foram o principal método de auxílio a ser implementado. Assim, ao clicar em “F1”, um arquivo contendo os principais tópicos será disponibilizado, para acesso aleatório do usuário. O arquivo de ajuda foi desenvolvido em um programa auxiliar, sendo este específico para o desenvolvimento de arquivos de ajuda.

O objetivo do arquivo de ajuda é esclarecer ou solucionar prováveis dúvidas que o usuário possa encontrar no decorrer da utilização da ferramenta. O fato da ajuda apresentar-se de forma semelhante a outros programas comerciais, pode ser visto como uma grande vantagem ao considerar que este é um método amplamente difundido e aceito como ajuda ao usuário pelos desenvolvedores de softwares comerciais.

4.5.5 DIAGRAMA DE FLUXO DE DADOS

O diagrama de fluxo de dados (DFD) é uma das ferramentas utilizadas na modelagem de sistemas, principalmente para sistemas em que suas funções sejam de fundamental importância, mais complexa do que os dados manipulados pelo sistema.

Os diagramas foram utilizados pela primeira vez na área da engenharia de software como uma representação para o estudo dos problemas do projeto de sistemas. A representação, por sua vez, foi trazida de antigos trabalhos sobre a teoria dos grafos e continua a ser usada como uma forma cômoda de notação por engenheiros de software interessados na implementação direta de modelos dos requisitos do usuário (DEMARCO, 1989).

Segundo Yourdon (1992), o diagrama de fluxo de dados consiste em processos, depósitos de dados, fluxos e terminais, onde:

- Processo: demonstra as diferentes funções individuais que o sistema executa, ou seja, os mecanismos que transformam as entradas em saídas.
- Fluxo: mostram as informações que são exigidas pelos processos como entrada e/ou as informações geradas como saídas, em outras palavras, são as conexões existentes entre os processos.
- Depósito de Dados: representam as informações armazenadas no sistema na forma de arquivos ou banco de dados, as quais residem temporariamente na memória enquanto o sistema processa a consulta.
- Terminais: referem-se a entidades externas as quais o sistema se comunica, são usuários que acessam o sistema.

Segundo Pressman (1995), o diagrama de fluxo de dados representa o fluxo dos dados dentro do programa. O DFD é utilizado para entendimento de um sistema, sendo a principal

ferramenta de análise estruturada e de especificação estruturada, sendo assim, o DFD visa apresentar os componentes e fluxos de forma sucinta.

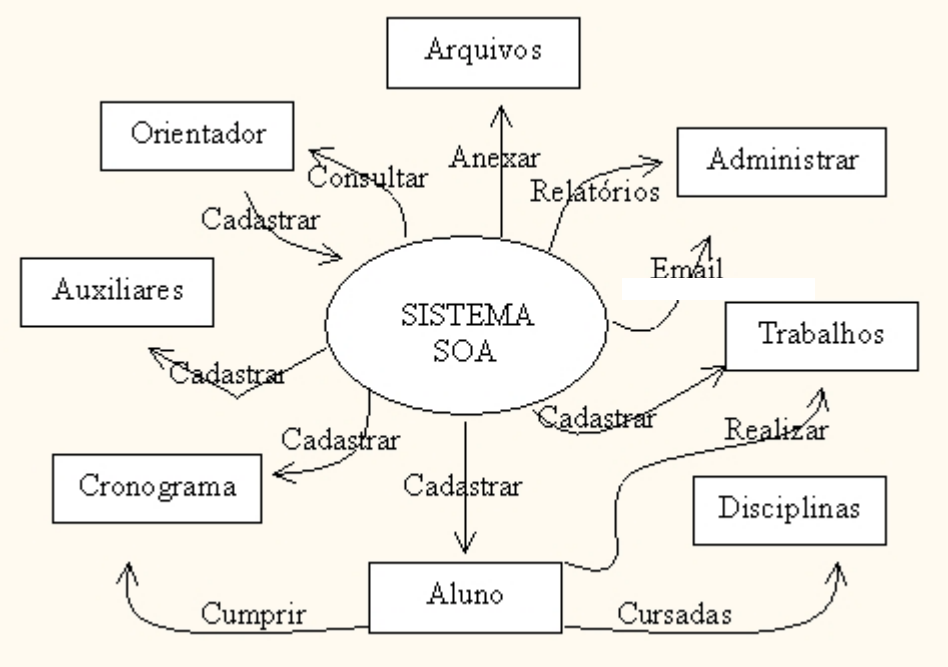


Figura 21: Visão Geral do SOA

4.5.6 DICIONÁRIO DE DADOS

Um dicionário de dados é um conjunto ordenado de definições, uma parte da especificação estrutural de um sistema. Sem ele, o MER é apenas uma imagem bonita que transmite alguma idéia dos campos de um sistema. O papel mais importante de qualquer dicionário de dados é fornecer-lhe um único lugar para as definições de termos e elementos utilizados dentro do MER, papel este de grande importância para a análise estruturada (DEMARCO, 1989).

O dicionário de dados das tabelas do sistema é definido e mostrado no apêndice B.

4.5.7 ALGUMAS TELAS DO SISTEMA

Esta seção tem a finalidade de apresentar algumas telas da ferramenta. É apresentada aqui, a tela principal do aplicativo, a tela do módulo cadastro de trabalhos e a tela do módulo cadastro de alunos, sendo que, o restante das telas do aplicativo bem como suas funções são descritas e mostradas no apêndice D deste trabalho.

A tela principal da ferramenta permite o acesso a todas as funções e módulos da mesma, através dela o usuário poderá cadastrar alunos, coorientadores, trabalhos, dentre outros. A figura 22 demonstra a tela principal.

Figura 22: Tela Principal da Ferramenta

A figura 23 ilustra a tela trabalhos, que permite o acesso a todas as funções e módulos relacionados ao controle do processo de orientação do aluno, bem como a parte de controle através de e-mail.

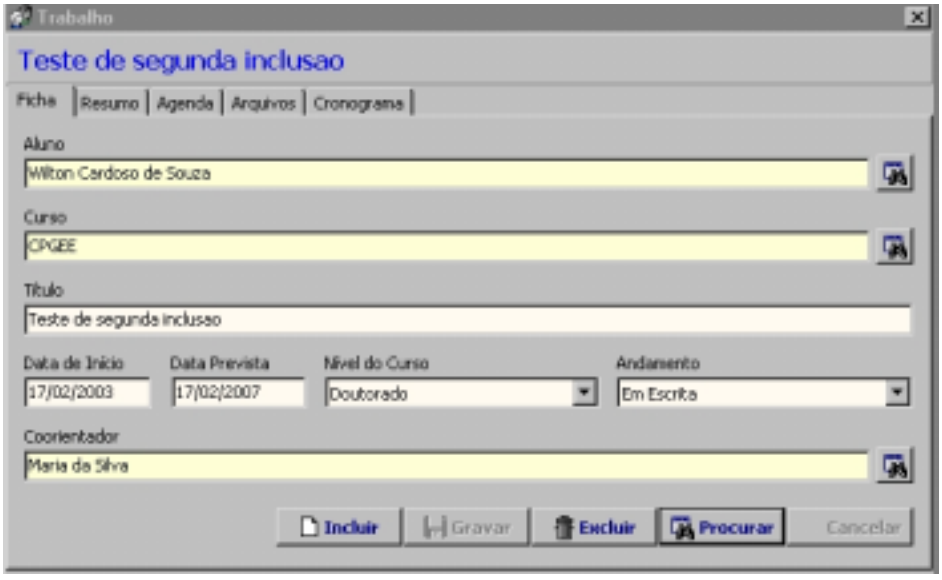


Figura 23: Tela Trabalho

A figura 24 demonstra a tela aluno, a qual permite o acesso a todas as funções e módulos relacionados a mesma, constando neste módulo além dos dados pessoais do aluno os dados mais relevantes sobre seu histórico acadêmico.

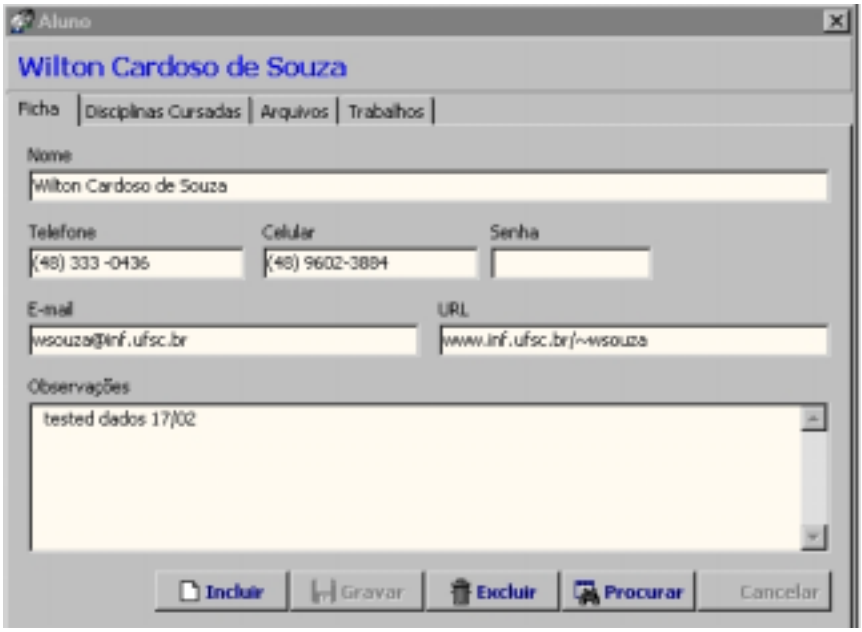


Figura 24: Tela Aluno

Esta seção apresentou o desenvolvimento do aplicativo, cumprimento das metas, os resultados experimentais e algumas considerações sobre os arquivos de ajuda da ferramenta proposta. A seguir será exposta a consideração final deste capítulo.

4.6 CONSIDERAÇÕES FINAIS

Com a codificação da ferramenta é que realmente se pode ter noção de sua utilidade, sendo assim as técnicas utilizadas para definição e implementação demonstraram-se eficientes e adequadas para o projeto proposto neste trabalho. Desta forma conseguiu-se desenvolver um protótipo de uma ferramenta de auxílio a orientadores consistente e que realmente cumpre com as funções pré-estabelecidas no projeto conceitual da mesma.

Este capítulo deu ênfase a ferramenta proposta e suas definições e experimentos realizados. A conclusão e trabalhos futuros são ênfase principal do capítulo seguinte.

5. CONCLUSÕES E RECOMENDAÇÕES

Esse capítulo encerra o presente trabalho e tem como finalidade apresentar as contribuições oferecidas pelo estudo realizado, suas dificuldades e sugestões para que este possa ser melhorado.

5.1 CONCLUSÕES

A partir da contextualização obtida pela revisão bibliográfica, envolvendo engenharia de software, mapeamento de processos e processo de orientação, partiu-se para a definição e implementação da ferramenta de auxílio proposta.

Sendo assim, primeiramente objetivou-se o projeto e a implementação de um sistema capaz de automatizar o processo de orientação e a sumarização de idéias com foco neste processo, utilizando-se de uma *interface desktop* para o usuário final, um processo de ligação entre esta e um sistema de banco de dados relacional, finalizando-se com a construção e/ou utilização de um módulo *Web*.

Nas primeiras etapas do desenvolvimento desta pesquisa, ainda durante o levantamento bibliográfico complementar, notou-se que a implementação de uma ferramenta para *desktop* para auxiliar orientadores seria viável, porém o módulo *Web* não seria possível, devido à complexidade envolvida em tal tipo de implementação e ao pouco tempo disponível para sua programação. Ainda nesta fase, optou-se por descartar a implementação do módulo *Web*, limitando-se apenas a sua definição e especificação do modelo, bem como suas telas.

Quanto as contribuições deste trabalho, ressalta-se além do material científico e bibliográfico exposto nesta dissertação e a ferramenta proposta, o fato de agrega-se ao trabalho proposto o uso de mapeamento de processos em paralelo as técnicas de desenvolvimento de software, pois como cita a bibliografia, a compreensão dos processos é a chave para qualquer processo.

Sobre as contribuições dos entrevistados, ressalta-se que foram de grande importância as respostas, sugestões e dicas dos mesmos, pois sem tais informações seria quase que impossível atingir-se o objetivo proposto. Além de servirem para entender e mapear o processo de orientação, também foram muito importantes para a definição dos módulos que compõem a ferramenta e assim, definir as tabelas e campos no banco de dados.

Dentre as limitações encontradas para a realização deste trabalho, cita-se a falta de bibliografia referente ao processo de orientação, definições de como funciona realmente o processo de orientação acadêmica. As poucas referências encontradas tratavam apenas de orientação pedagógica na escola. Outra limitação foi com relação às pessoas entrevistadas,

pois as mesmas não seguem um padrão para orientação de seus alunos. Desta forma, o que foi conseguido de informações está baseado em entrevistas e conversas com os orientadores, não obtendo-se assim, informações bem definidas pelos mesmos.

Dentro dos objetivos iniciais, o trabalho realizado nesta dissertação mostrou como contornar os problemas que os orientadores enfrentam quanto à falta de uma ferramenta para monitorar e armazenar os dados e andamentos de seus orientados. O mesmo foi desenvolvido para tentar suprir esta falta e auxiliar os mesmos na obtenção de tais informações e assim lhes proporcionar uma melhora contínua em suas atividades de orientação e pesquisa. Pois com base nas necessidades expressas pelas pessoas que foram entrevistadas e visando também atender de uma forma genérica um maior número de orientadores, este trabalho propõe um aplicativo que auxilie os orientadores. Eliminando assim, o uso de anotações em papéis e o uso de outros softwares que não sejam projetados para tal. Assim, o projeto tem como objetivo, dar mais tempo ao orientador para atender seus orientados e proporcionar um controle maior e mais centralizado de seus orientados e quem sabe até proporcionar um aumento no número dos mesmos, visto que a proposta é concentrar todo o histórico de orientação ao aluno num único lugar, provendo desta forma um acesso rápido e eficiente as informações.

5.2 RECOMENDAÇÕES PARA TRABALHOS FUTUROS

A ferramenta de auxílio a orientadores foi proposta em dois módulos, o módulo *desktop*, o qual foi modelado e implementado neste trabalho e o módulo *Web*, no qual fez-se apenas a definição do modelo para tal neste trabalho, ficando sua implementação para trabalhos futuros devido a sua complexidade e falta de tempo para realizar-se a implementação da mesma.

Ressalta-se ainda que pode haver um estudo mais aprofundado no processo de orientação, bem como em seu mapeamento, gerando desta forma um outro trabalho na área.

Uma outra questão que poderá ser implementada e integrada à ferramenta é o uso de técnicas de busca, pois como o aplicativo pode rodar em máquinas separadas, o banco de dados do mesmo não é compartilhado. Sendo assim alguma técnica de busca pode ser implementado nos mesmos para a retirada de informações e desta forma disponibilizá-las no meio *Web*.

Na parte do módulo *Web* pode-se fazer uso de alguma ferramenta para a tradução de textos e artigos, pois como o mesmo suportará o compartilhamento de uma base bibliográfica, tal técnica seria de grande valia para o trabalho.

REFERÊNCIAS

BEZERRA, E.; **Princípios de análise e projeto de sistemas com UML**. Rio de Janeiro: Campus, 2001.

BOMFIM, F.; AZEVEDO M.; HUDSON S. Métricas de software. Disponível em: <http://www.internext.com.br/mssa/medidas.html>. Acesso em: 15 mar. 2003.

BRAGA, A. **Análise de pontos de função**. IBPI Press, 1996.

CANDÉAS, A. J.; LOPES, C.C.N. Estimativa: Uma ferramenta para agilizar o dimensionamento de projetos no SERPRO com base na metodologia de análise de pontos por função. In: SIMPÓSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE. Caderno de Ferramentas, 13, 1999, Florianópolis. **Anais...** Florianópolis: UFSC, 1999.

CARVALHO, A.M.B.R.; CHIOSSI, T.C.S. **Introdução à engenharia de software**. Campinas: Unicamp Editora, 2001.

CHANG, C. K.; CHRISTENSEN, M. A. Net Practice for Software Project Management. IEEE SOFTWARE. V. 16, n. 6, p. 80 -88, nov./dec. 1999.

CYBIS, W. A. **Engenharia de usabilidade: Uma abordagem ergonômica**. Florianópolis, Laboratório LabIUtil, UFSC, 2003.

DATE, C. **Introdução a sistemas de banco de dados**. 4 ed., Rio de Janeiro: Campus, 1991.

DAVENPORT, T. H. **Reengenharia de processos**. Rio de Janeiro: Campus, 1994.

DAVIS, M. R.; WECKLER, D. A. **A pratical guide to organization design**. Los Altos, CA: Crisp Publications, 1996.

DEMARCO, T. **Análise estruturada e especificação de sistema**. Rio de Janeiro: Campus, 1989.

FENTON, N. E.; PFLEEGER, S. L. **Software metrics: A rigorous & practical approach.** 2 ed. Boston: PWS Publishing, 1997.

FURLAN, J.D. **Modelagem de objetos através da UML**. São Paulo: Makron Books, 1998.

GHEZZI, C.; JAZAYERI, M.; MANDRIOLI, D. **Fundamentals of software engineering.** New Jersey: Prentice-Hall, 1991.

GOMES, N. S. Qualidade de Software - Uma Necessidade. Disponível em: <http://www.esaf.fazenda.gov.br/cst/arquivos/Qualidade_de_Soft.pdf> Acesso em: 05 mai. 2003.

HARRINGTON, J. **Aperfeiçoando processos empresariais**. São Paulo: Makron Books, 1993.

HECKEL, P. **Software amigável**. Rio de Janeiro: Campus, 1993.

HEUSER, C. A. **Projeto de banco de dados**. Porto Alegre: Sagra Luzzato, 1999.

HUNT, V. D. **Process mapping**: how to reengeneer your business processes. New York: John Wiley & Sons, Inc., 1996.

IFPUG - International function point users group. Disponível em: <http://www.ifpug.org/>. Acesso em 12 mar. 2003

JALOTE, P. **An integrated approach to software engineering.** 2 ed. New York: Springer-Verlag, 1997.

JOHANSSON, H. J. et al. **Processos de negócios**. São Paulo: Pioneira, 1995.

KORTH, H. F.; SILBERSCHATS, A.; SUDARSHAN S. **Sistemas de banco de dados**. São Paulo: Makron Books, 1999.

LARMAN, C. **Utilizando UML e padrões: uma introdução à análise e ao projeto orientados a objetos**. Porto Alegre: Bookman, 2000.

LINDGARD, G. **Usability Testing and system evaluation.** Chapman & Hall: London (UK), 1994.

MACCONELL, S. The Best Influences in Software Engineering. IEEE Software Eletronic Magazine, New York, v.11, n.9, February, 2000.

MARTIM, J., McCLURE, C. **Structured techniques: The basis for case**. New Jersey: Prentice Hall, 1988.

NAVATHE, S.; ELMASRI, R. **Sistemas de banco de dados**. Rio de Janeiro: LTC, 2002.

NBR 13596, Tecnologia de Informação – Avaliação de produto de software – Características de Qualidade e Diretrizes para o seu Uso; ABNT – Abril de 1996 (versão brasileira da Norma ISSO/IEC 9126, 1991).

NORMAN, D.A. **Cognitive engineering**, in User Centered System Design, Hillsdale, New Jersey: Lawrence Erlbaum. 1986.

O'CONNOR, R. **An architecture for an intelligent assistant system for use in software project planning.** PhD Thesis, City University. Londres, 2000.

PEZZIN, M. Z. **Programa computacional baseado em conceitos financeiros e contábeis para o gerenciamento de pequenas empresas.** Florianópolis, 2001. 103 f. Dissertação (Mestrado) - Universidade Federal de Santa Catarina.

PIDD, M. **Modelagem empresarial: ferramentas para tomada de decisão**. Porto Alegre: Artes Médicas, 1998.

PRESSMAN, R. S. **Engenharia de software.** São Paulo: Makron Books, 1995.

REZENDE, D. A. **Engenharia de software e sistemas de informação**. Rio de Janeiro: Brasport, 1999.

ROYCE, W. **Software project management: A unified framework**. USA: Addison-Wesley Publishers, 1998.

RUMMLER, G. A. BRACHE, A. P. **Melhores desempenhos das empresas**. São Paulo: Makron Books, 1994.

SILVA, L.E.; MEZES, M. E.. **Metodologia da pesquisa e elaboração de dissertação 3 ed**. Florianópolis: Laboratório de Ensino à Distância da UFSC, 2001.

SOARES, L.F.G., LEMOS, G., COLCHER, S. **Redes de computadores: das LANs, MANs e WANs as redes ATM. 2**. Rio de Janeiro: Campus, 1995.

SOMMERVILLE, I. **Engenharia de software**. 6 ed. São Paulo: Addison-Wesley, 2003.

SOFTWARE PRODUCTIVITY RESEARCH. Disponível em: <http://www.spr.com/>. Acesso em: 05 abr. 2003.

TANEMBAUM, A. S., **Redes de computadores**. Rio de Janeiro: Campus, 1994.

VON MAYRHAUSER, A. **Software engineering: Methods and management**. San Diego: Academic Press, 1990.

WU, C.; SIMMONS, D. B. Software project planning associate (SPPA): A Knowledge – Based Approach for Dynamic Software Project Planning and Tracking In: IEEE Proceedings of the Twenty-Fourth Annual International Computer Software and Applications Conference, 2000.

YOURDON, E. **Análise Estruturada Moderna**. Rio de Janeiro: Campus, 1992.

ZAHRAN, S. **Software process improvement: Practical guidelines for business success**. Edinburgh: Addison-Wesley,1998.

APÊNDICE A

Neste apêndice apresenta-se lista de verificação usada para iniciar-se as entrevistas para a coleta dos dados da pesquisa.

Lista de Verificação

1. Identificação do Orientador:

Nome:_____

Departamento em que orienta: _____

2. Que atividades estão envolvidas no processo de orientação?

3. Que critérios usa para selecionar o orientando?

4. Como define o tema de pesquisa do orientando?

5. Que atividades costuma repassar a seus orientandos?

6. Como acompanha o desenvolvimento do trabalho de seus orientandos?

7. Que forma usa para anotar/arquivar e controlar o andamento dos orientandos?

8. Usa algum tipo de software como apoio à orientação?

9. Que recursos seriam necessários para ajuda no controle da orientação?

10. Um aplicativo de apoio a orientadores deveria possuir que características?

11. Este aplicativo o ajudaria/facilitaria a ter um controle melhor, um maior tempo e até proporcionar um aumento no número de orientandos?

12. De que forma tira as dúvidas de seus orientandos?

13. Características do protótipo a ser proposto: Um aplicativo com estas características serviria como ajuda no controle do processo de orientação?

- Multi-usuário;
- Cadastro de alunos tanto em graduação, mestrado e doutorado;
- Segurança dos dados protegidos por senha;
- Consulta e relatórios de todas as atividades e cronogramas;
- Cadastro e consulta de todas as atividades do aluno desenvolvidas no processo;
- Parte inteligente do sistema: será através de e-mail e um checklist na inicialização;
    - Pré-configuração de e-mail e datas definidas pelo orientador;
    - Envio de avisos aos orientado automaticamente pelo aplicativo;

    - Armazenamento dos mesmos automaticamente pelo aplicativo;
    - Controle de cronogramas que o orientado tem a cumprir;
    - Checklist informando o orientador das atividades no dia e por e-mail como configurado pelo mesmo.
    - Todo o evento de cada aluno será armazenado em sua conta para posterior consulta ou relatório.

- O aplicativo terá outras funcionalidades que não estão citas aqui, por não terem sido definidas ainda.

14. Sugestões:

APÊNDICE B

Neste apêndice é apresentado o dicionário de dados das tabelas do banco de dados usado neste trabalho, bem como seus comentários. O dicionário foi gerado a partir da ferramenta ERwin.

Dicionário de Dados do SOA

Name	Datatype	Null Option	Commentários
ARQUIVO	BLOB	NULL	Arquivo importado da tabela ARQUIVOS
CELULAR	CHAR(10)	NULL	Celular do Orientador da tabela ORIENTADORES
CELULAR	CHAR(10)	NOT NULL	Celular do aluno na tabela ALUNOS
CELULAR	CHAR(10)	NOT NULL	Celular do coorientador da tabela COORIENTADORES
CODIGO_DISCIPLINA	VARCHAR(10)	NULL	Código da disciplina da tabela DISCIPLINAS
COMPROMISSO	VARCHAR(50)	NOT NULL	Nome ou tipo de compromiso da tabela AGENDA
DATA	DATE	NULL	Data do compromisso da tabela AGENDA
DATA_CONCLUSAO	DATE	NULL	Data de conclusão do cronograma da tabela CRONOGRAMA
DATA_FINAL	DATE	NULL	Data de previsao da conclusão do curso da tabela TRABALHOS
DATA_INCLUSAO	DATE	NULL	Data em que o arquivo foi importado da tabela ARQUIVOS
DATA_INICIAL	DATE	NULL	Data que aluno inicio curso da tabela TRABALHOS
DATA_INICIO	DATE	NULL	Data de início do cronograma da tabela CRONOGRAMA
DATA_LEITURA	DATE	NULL	Data de leitura do arquivo da tabela ARQUIVOS
DATA_PREVISTA	DATE	NULL	Data prevista para a conclusão do cronograma da tabela CRONOGRAMA
DATA_AVISO	DATE	NULL	Data de aviso através de e-mail via agenda, da tabela CRONOGRAMA
DESCRICAO	VARCHAR(200)	NULL	Descrição do cronograma a ser cumprido da tabela CRONOGRAMA
DESCRICAO_ARQUIVO	VARCHAR(250)	NULL	Descrição do arquivo da tabela ARQUIVOS
EMAIL	VARCHAR(50)	NOT NULL	E-mail do coorientador da tabela COORIENTADORES
EMAIL	VARCHAR(50)	NOT NULL	E-mail do aluno na tabela ALUNOS
EMAIL_AVISAR	CHAR(1)	NULL	Opção que marcada envia e-mail na data determinada da tabela AGENDA
EMAIL_DATA_ENVIAR	DATE	NULL	Data que será enviado da tabela AGENDA
EMAIL_DATA_ENVIO	DATE	NULL	Data que o e-mail foi enviado da tabela AGENDA
EMAIL_END_RETORNO	VARCHAR(50)	NULL	Endereço de e-mail do orientador da tabela ORIENTADORES
EMAIL_END_SERVIDOR	VARCHAR(50)	NULL	Enderreço do servidor de e-mail do orientador da tabela

Name	Datatype	Null Option	Commentários
			ORIENTADORES
EMAIL_ENVIAR	CHAR(1)	NULL	Opção que será marcada para envio de e-mail da tabela ORIENTADORES
EMAIL_LOGIN	VARCHAR(50)	NOT NULL	Login do orientador no servidor de e-mail da tabela ORIENTADORES
EMAIL_NUM_DIAS	SMALLINT	NULL	Número de dias antes que os e-mail's serão enviados da tabela ORIENTADORES
EMAIL_NUM_DIAS	SMALLINT	NULL	Número de dias antes que o e-mail será enviado da tabela AGENDA
EMAIL_TEXTO	BLOB SUB_TYPE 1	NULL	Texto pré-definido para os e-mail's da tabela ORIENTADORES
HORA	DATE	NULL	Hora do compromisso da tabela AGENDA
ID_AGENDA	INTEGER	NOT NULL	Chave primária da tabela AGENDA
ID_ALUNO	INTEGER	NOT NULL	Chave primária (PK) da tabela ALUNOS
ID_ALUNO	INTEGER	NOT NULL	Chave primária (PK) da tabela ALUNOS
ID_ALUNO	INTEGER	NULL	Chave primária (PK) da tabela ALUNOS
ID_ALUNO_DISCIPLINA	INTEGER	NOT NULL	Chave primária (PK) da tabela ALUNOS_DISCIPLINAS
ID_ARQUIVO	INTEGER	NOT NULL	Chave primária (PK) da tabela ARQUIVOS
ID_COORIENTADOR	SMALLINT	NOT NULL	Chave primária (PK) da tabela COORIENTADORES
ID_COORIENTADOR	SMALLINT	NULL	Chave primária (PK) da tabela COORIENTADORES
ID_CRONOGRAMA	INTEGER	NOT NULL	Chave primária (PK) da tabela CRONOGRAMA
ID_CURSO	SMALLINT	NOT NULL	Chave primária (PK) da tabela CURSOS
ID_CURSO	SMALLINT	NULL	Chave primária (PK) da tabela CURSOS
ID_DISCIPLINA	SMALLINT	NOT NULL	Chave primária (PK) da tabela DISCIPLINAS
ID_DISCIPLINA	SMALLINT	NULL	Chave primária (PK) da tabela DISCIPLINAS
ID_NIVEL_ANDAMENTO	SMALLINT	NOT NULL	Chave primária (PK) da tabela NIVEL_ANDAMENTO
ID_NIVEL_ANDAMENTO	SMALLINT	NULL	Chave primária (PK) da tabela NIVEL_ANDAMENTO
ID_NIVEL_GRADUACAO	SMALLINT	NOT NULL	Chave primária (PK) da tabela NIVEL_GRADUACAO
ID_NIVEL_GRADUACAO	SMALLINT	NULL	Chave primária (PK) da tabela NIVEL_GRADUACAO
ID_ORIENTADOR	SMALLINT	NULL	Chave primária (PK) da tabela ORIENTADOES
ID_ORIENTADOR	SMALLINT	NOT NULL	Chave primária (PK) da tabela ORIENTADOES
ID_ORIENTADOR	SMALLINT	NOT NULL	Chave primária (PK) da tabela ORIENTADOES
ID_TRABALHO	INTEGER	NULL	Chave primária (PK) da tabela TRABALHOS
ID_TRABALHO	INTEGER	NOT NULL	Chave primária (PK) da tabela TRABALHOS
ID_TRABALHO	INTEGER	NOT NULL	Chave primária (PK) da tabela TRABALHOS
ID_TRABALHO	INTEGER	NULL	Chave primária (PK) da tabela TRABALHOS
LOGIN	VARCHAR(20)	NOT NULL	Login do orientador da tabela ORIENTADORES
NOME_ALUNO	VARCHAR(50)	NOT NULL	Nome do aluno na tabela ALUNOS
NOME_ARQUIVO	VARCHAR(250)	NULL	Nome do arquivo importado da

Name	Datatype	Null Option	Commentários
			tabela ARQUIVOS
NOME_COORIENTADOR	VARCHAR(50)	NOT NULL	Nome do coorientador da tabela COORIENTADORES
NOME_CURSO	VARCHAR(50)	NOT NULL	Nome do curso da tabela CURSOS
NOME_DISCIPLINA	VARCHAR(50)	NOT NULL	Nome da disciplina da tabela DISCIPLINAS
NOME_NIVEL_ANDAMENTO	VARCHAR(50)	NOT NULL	Indice do nível de andamento do trabalho da tabela NIVELANDAMENTO
NOME_NIVEL_GRADUACAO	VARCHAR(50)	NOT NULL	Identifica o nível de graduação do aluno da tabela NIVEL_GRADUACAO
NOME_ORIENTADOR	VARCHAR(50)	NOT NULL	Nome orientador da tabela ORIENTADORES
NUM_CREDITOS	SMALLINT	NULL	Número de créditos da disciplina da tabela DISCIPLINAS
OBSERVACOES	BLOB SUB_TYPE 1	NULL	Observações sobre o cronograma da tabela CRONOGRAMA
OBSERVACOES	BLOB SUB_TYPE 1	NULL	Observações sobre o compromisso da tabela AGENDA
OBSERVACOES	BLOB SUB_TYPE 1	NULL	Observações sobre o arquivo da tabela ARQUIVOS
OBSERVACOES	BLOB SUB_TYPE 1	NULL	Observações sobre o aluno na tabela ALUNOS
RESUMO	BLOB SUB_TYPE 1	NULL	Resumo do trabalho do aluno da tabela TRABALHOS
SENHA	VARCHAR(10)	NULL	Senha do orienador da tabela ORIENTADORES
SENHA	VARCHAR(10)	NULL	Senha do aluno da tabela ALUNOS
TELEFONE	CHAR(10)	NULL	Telefone do orientador da tabela ORIENTADORES
TELEFONE	CHAR(10)	NOT NULL	Telefone do aluno na tabela ALUNOS
TELEFONE	CHAR(10)	NOT NULL	Telefone do coorientador da tabela COORIENTADORES
TITULO	VARCHAR(250)	NULL	Titulo do tarbalho da tabela TRABALHOS
URL	VARCHAR(50)	NOT NULL	Endereço de página web na tabela ALUNOS
URL	VARCHAR(50)	NOT NULL	Endereço web do coorientador da tabela COORIENTADORES

APÊNDICE C

Neste apêndice lista-se as especificações principais dos Casos de Usos do SOA *Web*.

Nome do Caso de Uso: **Acesso ao Aplicativo**

Categoria: Login

Objetivo: usuário poder ter acesso à ferramenta.

Fluxo principal

1 - usuário entra com o login e senha cadastrada.

2 - o login e a senha são case sensitive e o usuário terá que digitá-los corretamente.

4 - o usuário tem acesso à ferramenta

Figura 21 Acesso ao Módulo Web

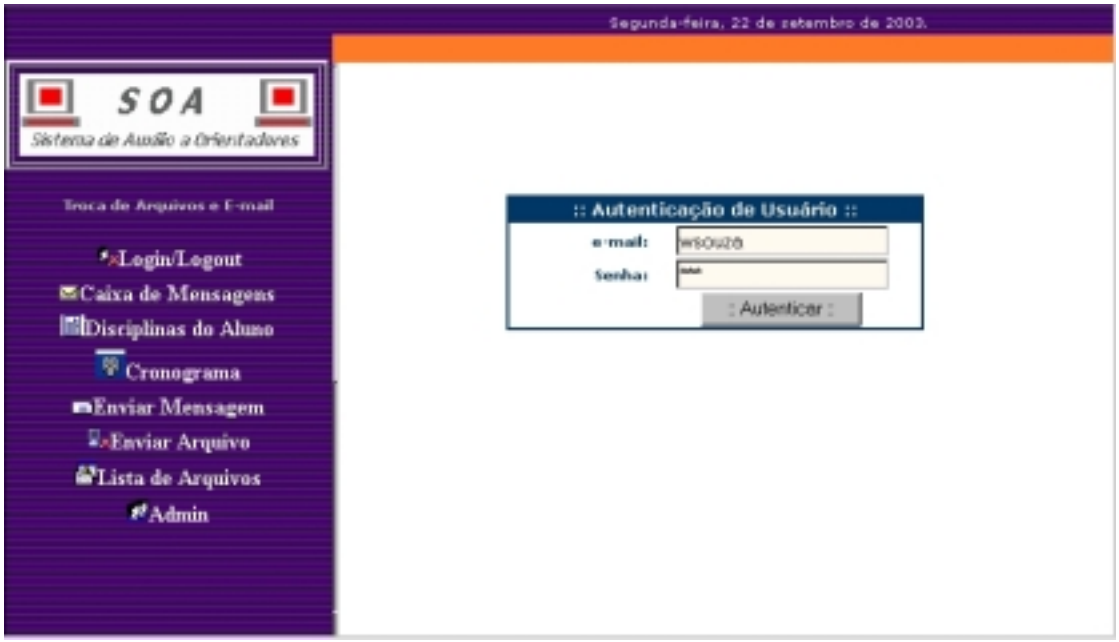


Figura 22 – Usuário Logado no Sistema

Nome do Caso de Uso: **Caixa de Mensagens**

Categoria: Mensagens

Objetivo: usuário pode ler as mensagens recebidas.

Fluxo principal

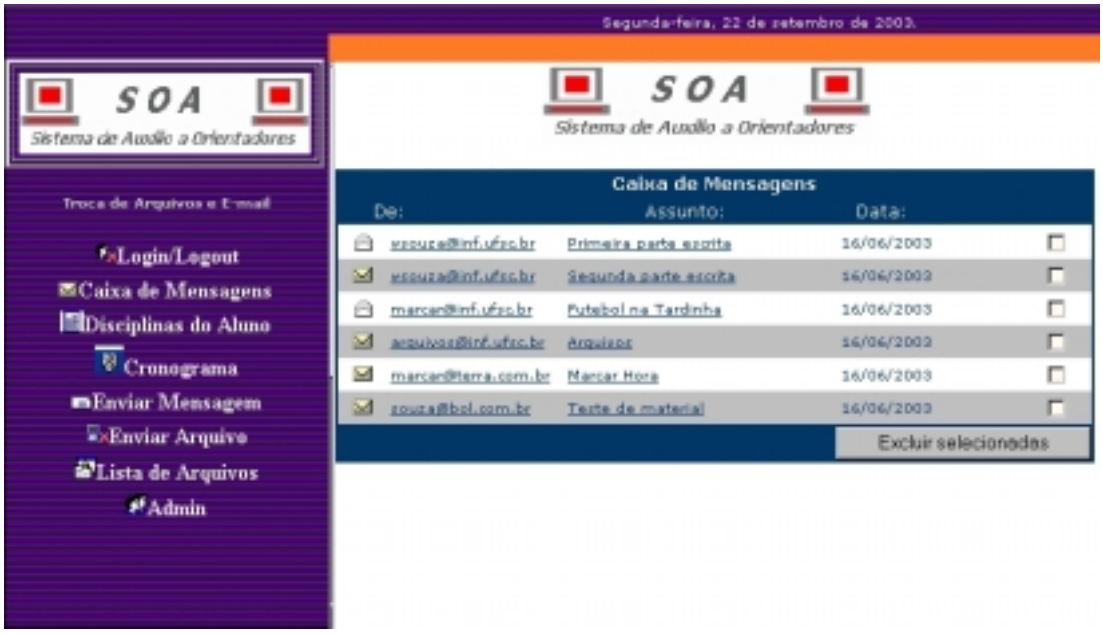
1 - usuário acessa a caixa de mensagens.

2 - usuário lê suas mensagens.

Fluxo Alternativo 1:

1 - usuário pode excluir as mensagens selecionadas.

Figura 23 – Caixa de Mensagens do Usuário



Nome do Caso de Uso: **Disciplinas do Aluno**

Categoria: Disciplinas

Objetivo: usuário pode ver e incluir suas disciplinas.

Fluxo principal

1 - usuário acessa as disciplinas cursadas.

2 - usuário inclui disciplinas.

Fluxo Alternativo 1:

1 - usuário pode excluir as disciplinas selecionadas.

Figura 24 – Disciplinas do Aluno

Nome do Caso de Uso: **Cronogramas do Aluno**

Categoria: Cronograma

Objetivo: usuário pode visualizar seus cronogramas.

Fluxo principal

1 - usuário acessa seus cronogramas.

Fluxo Alternativo 1:

1 - usuário aluno não pode incluir nem alterar o cronograma.

Figura 25 – Cronogramas do Aluno

Nome do Caso de Uso: **Envio de Mensagens**

Categoria: Mensagens

Objetivo: usuário pode enviar mensagens.

Fluxo principal

1 - usuário acessa a caixa de mensagens.

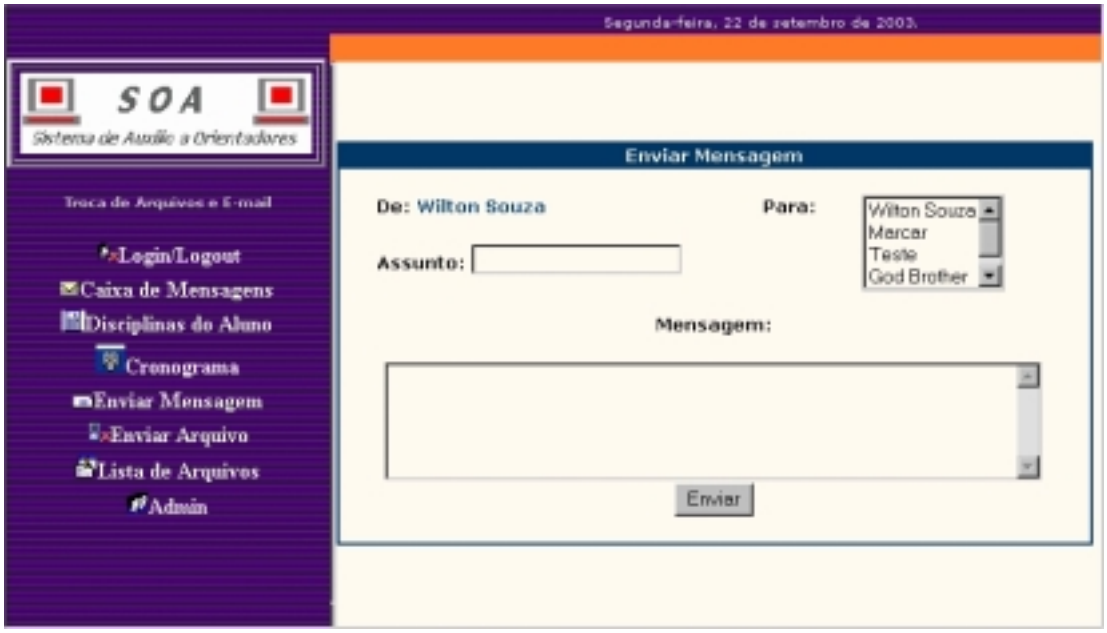
2 - usuário lê suas mensagens.

Fluxo Alternativo 1:

1 - usuário pode excluir as mensagens selecionadas.

2 - usuário só pode enviar mensagens ao seu orientador

Figura 26 – Envio de Mensagens



Nome do Caso de Uso: **Envio de Arquivos**

Categoria: Arquivos

Objetivo: usuário pode enviar arquivos.

Fluxo principal

1 - usuário acessa o módulo envio de arquivos.

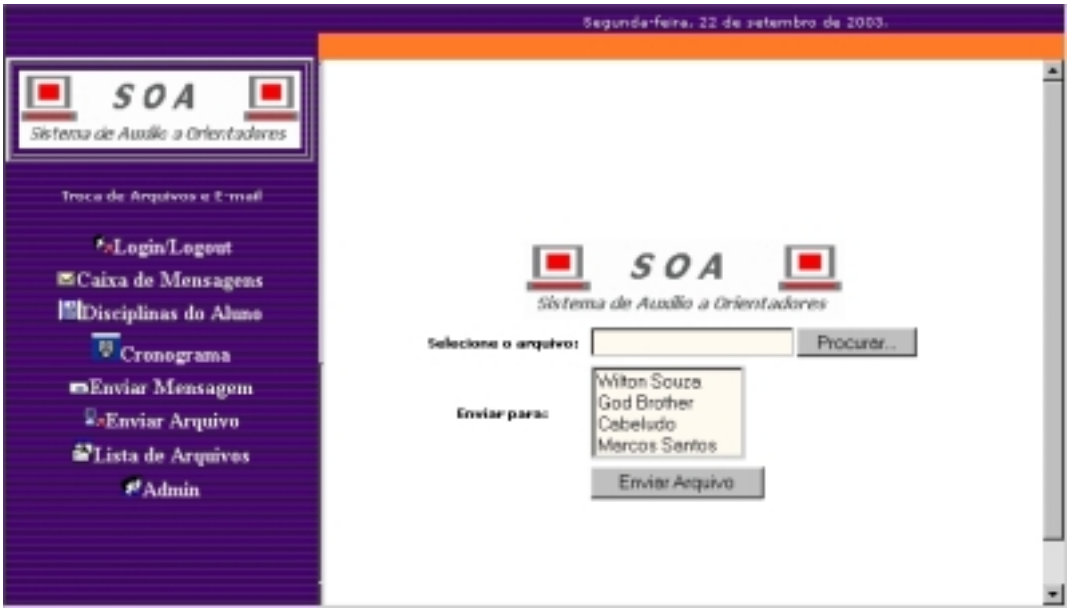
2 - usuário seleciona o arquivo que deseja enviar.

3- os arquivos têm que estar no formato .pdf ou .zip

Fluxo Alternativo 1:

1 – usuário só pode enviar arquivos ao seu orientador.

Figura 27 – Envio de Arquivos



Nome do Caso de Uso: **Lista de Arquivos**

Categoria: Arquivos

Objetivo: usuário pode visualizar os arquivos recebidos.

Fluxo principal

1 - usuário acessa o módulo arquivos.

2 - usuário pode ler os arquivos.

Fluxo Alternativo 1:

1 – usuário poderá excluir seus arquivos.

Figura 28 – Lista de Arquivos

Nome do Caso de Uso: **Administrador**

Categoria: Admin

Objetivo: cadastrar usuários no sistema.

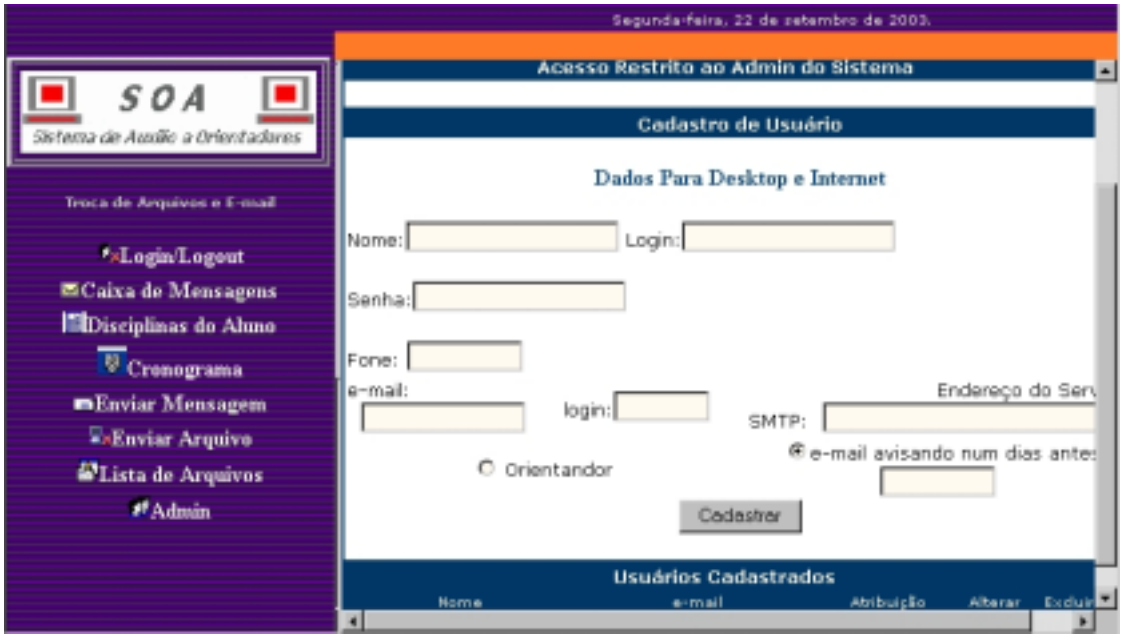
Fluxo principal

1 - usuário cadastra e define as permissões de acesso aos usuários.

Fluxo Alternativo 1:

1 - usuário pode excluir pessoas selecionadas.

Figura 29 – Admin



APÊNDICE D

Neste apêndice lista-se as especificações principais dos Casos de Usos do SOA *Desktop*.

Nome do Caso de Uso: **Acesso ao Aplicativo**

Categoria: Login

Objetivo: usuário terá acesso à ferramenta.

Fluxo principal

1 - usuário entra com o login e senha cadastrada.

2 - usuário pode alterar a sua senha na tela de login.

3 - o login e a senha são case sensitive e o usuário terá que digitá-los corretamente.

4 - o usuário tem acesso à ferramenta

Fluxo Alternativo 1:

1 - Não é obrigatório o usuário efetuar o evento 2, o usuário poderá faze-lo depois.

Tela 01 – Login de usuário

Nome do Caso de Uso: **Cadastro de Auxiliares**

Categoria: Cadastro

Objetivo: usuário poderá cadastrar andamento, nível, cursos e disciplinas.

Fluxo principal

1 - usuário entra com o andamento clicando em incluir andamento.

2 - usuário entra com o nível clicando em incluir nível.

3 - usuário entra com nome do curso clicando em incluir curso.

4 - o usuário entra com o nome da disciplina que deseja cadastrar.

Fluxo Alternativo 1:

1 - Não é obrigatório o usuário efetuar o evento 4 neste momento, o usuário poderá fazê-lo quando for cadastrar as disciplinas no caso de uso cadastro do aluno.

Tela 02 Cadastro de Auxiliares

Nome do Caso de Uso: **Cadastro de Alunos**

Categoria: Cadastro

Objetivo: usuário poderá cadastrar dados dos alunos.

Fluxo principal

1 - usuário entra com os dados pessoais do aluno clicando em incluir.

2 - usuário entra com as disciplinas do aluno clicando em incluir.

3 - usuário pode cadastrar disciplinas clicando em cadastrar disciplina.

4 - usuário poderá imprimir as disciplinas cursadas pelo aluno

Fluxo Alternativo 1:

1 - usuário poderá visualizar os arquivos recebidos de seus alunos clicando em visualizar arquivo.

Fluxo Alternativo 2:

1 - usuário poderá visualizar e alterar os trabalhos de seus alunos

2 - usuário poderá alterar os dados do trabalho.

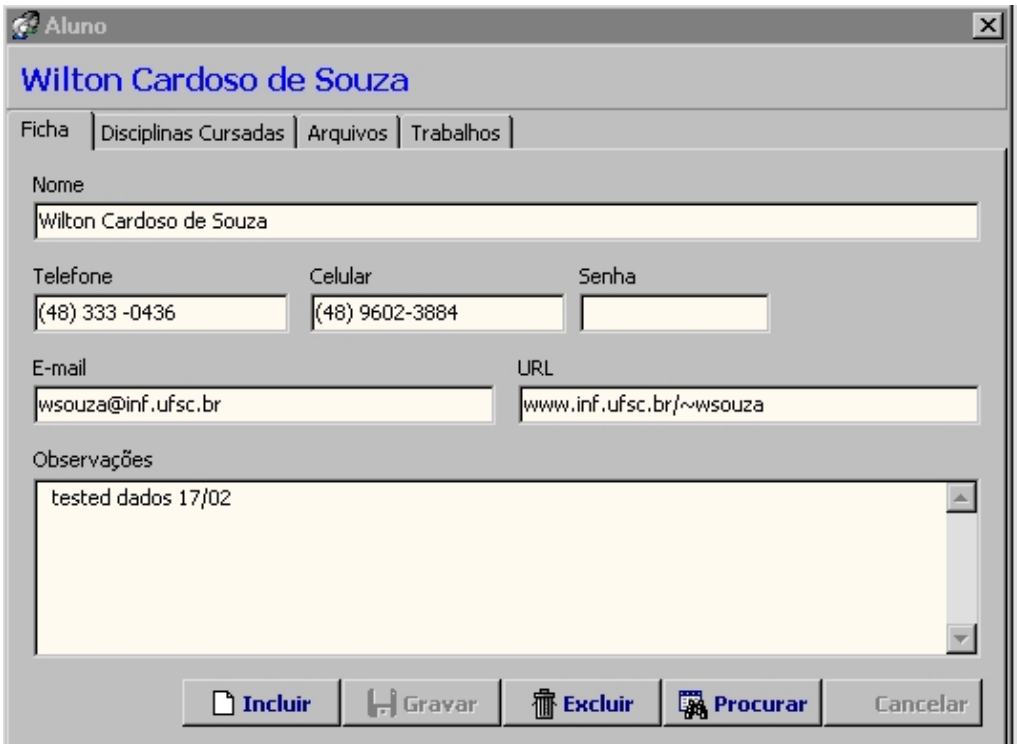
3 - usuário poderá incluir trabalho.

4 - usuário poderá imprimir trabalho.

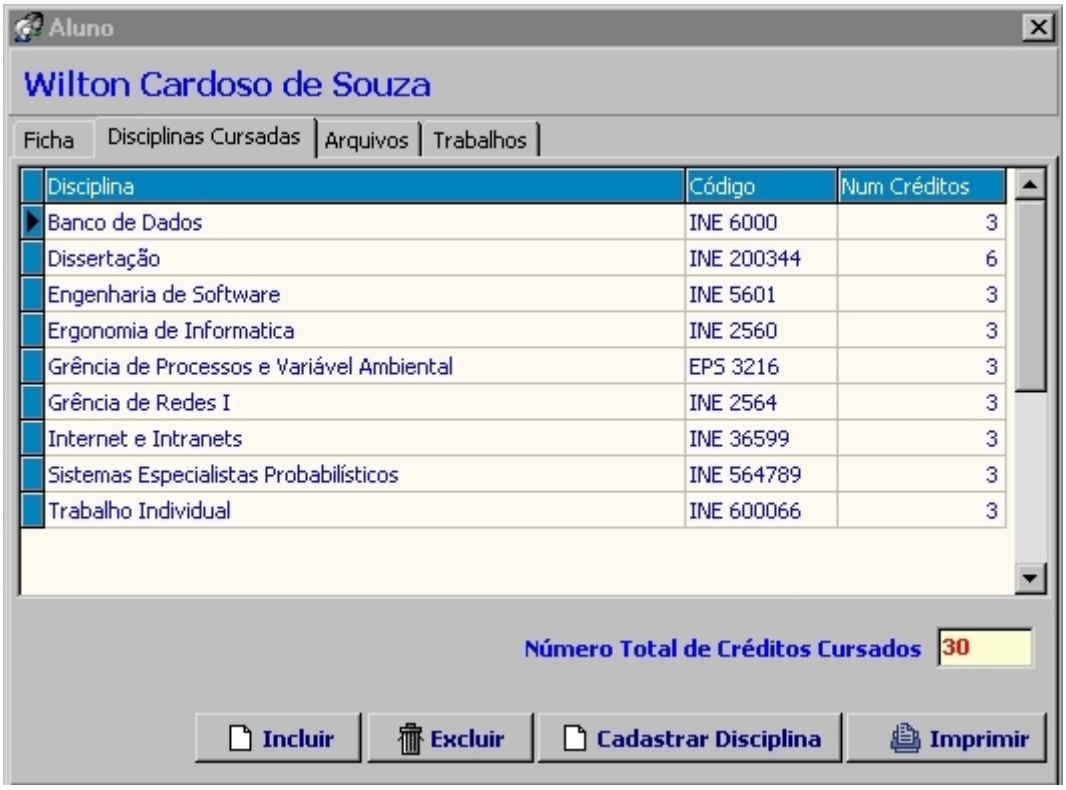
Fluxo Alternativo 3:

1 - usuário pode importar os dados pessoais do aluno clicando em importar alunos.

Tela 03 – Cadastro de Alunos

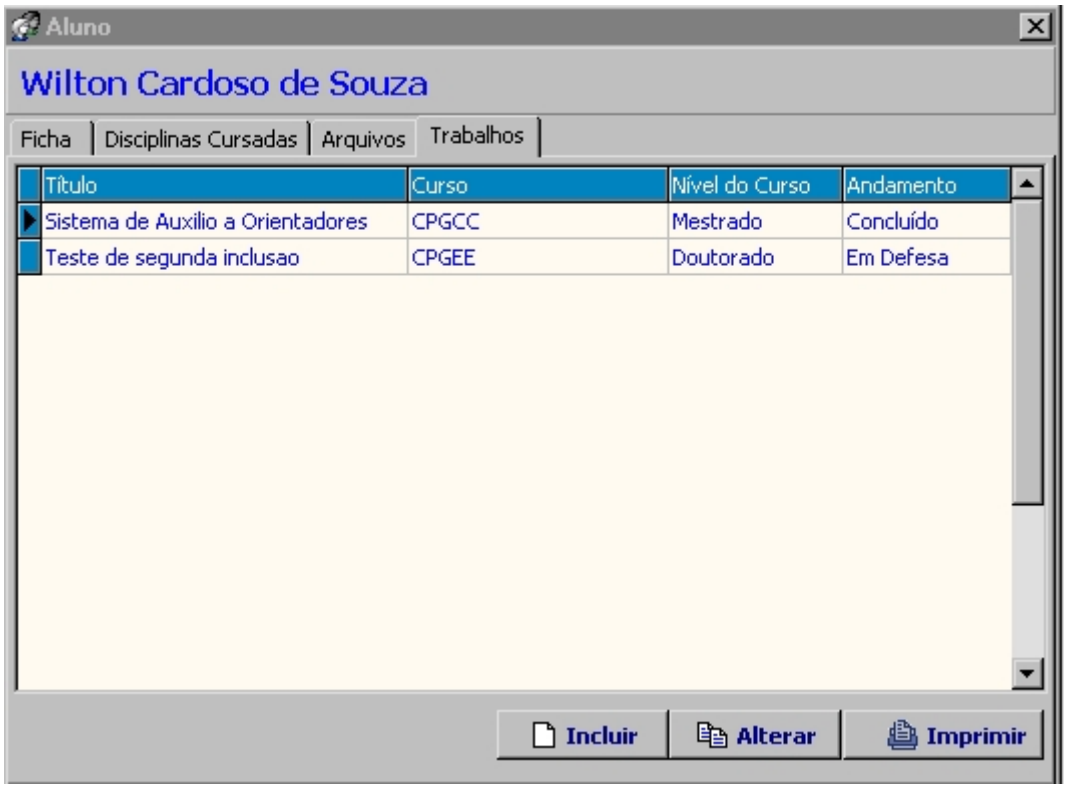


Tela 04 – Disciplinas



Tela 05 – Arquivos

Tela 06 – Trabalhos



Nome do Caso de Uso: **Cadastro de Coorientadores**

Categoria: Cadastro

Objetivo: usuário poderá cadastrar os coorientadores.

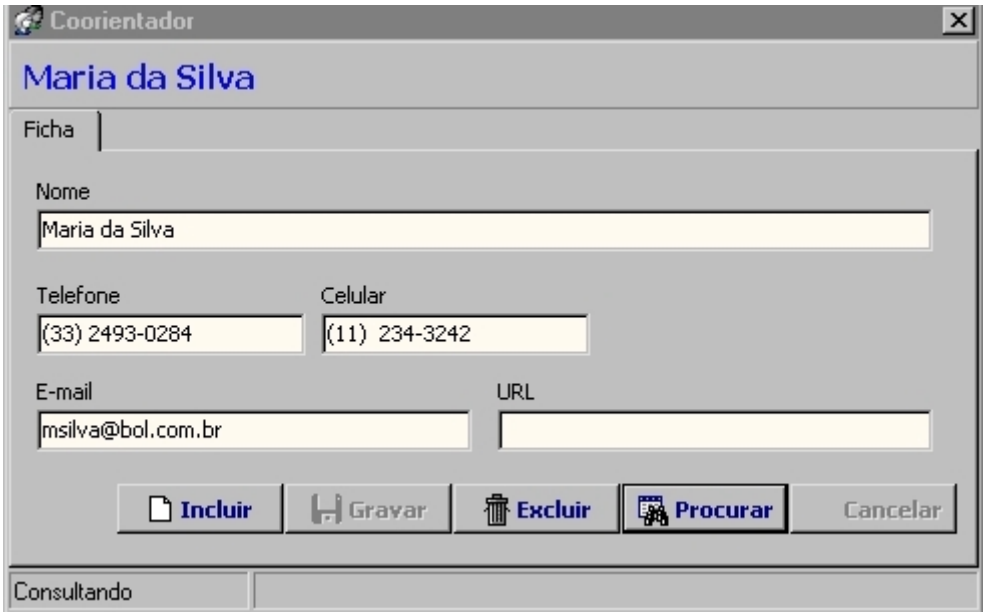
Fluxo principal

1 - usuário entra com os dados pessoais do coorientador clicando em incluir.

Fluxo Alternativo 1:

1 - usuário poderá visualizar os coorientadores cadastrados clicando em procurar.

Tela 07 – Coorientadores



Nome do Caso de Uso: **Cadastro de Trabalhos**

Categoria: Cadastro

Objetivo: usuário poderá cadastrar os trabalhos dos alunos.

Fluxo principal

1 - usuário entra com os dados do trabalho do aluno clicando em incluir.

2 - usuário entra com o resumo do trabalho do aluno.

3 - agenda armazena os mail e avisos enviados ao aluno.

4 - usuário poderá imprimir as disciplinas cursadas pelo aluno

5 - usuário anexa arquivos enviados pelo aluno.

6 - usuário cadastra cronogramas do aluno.

Fluxo Alternativo 1:

1 - usuário poderá visualizar as mensagens enviadas a seus alunos clicando na mensagem.

2 - usuário pode enviar mensagens ao seu aluno

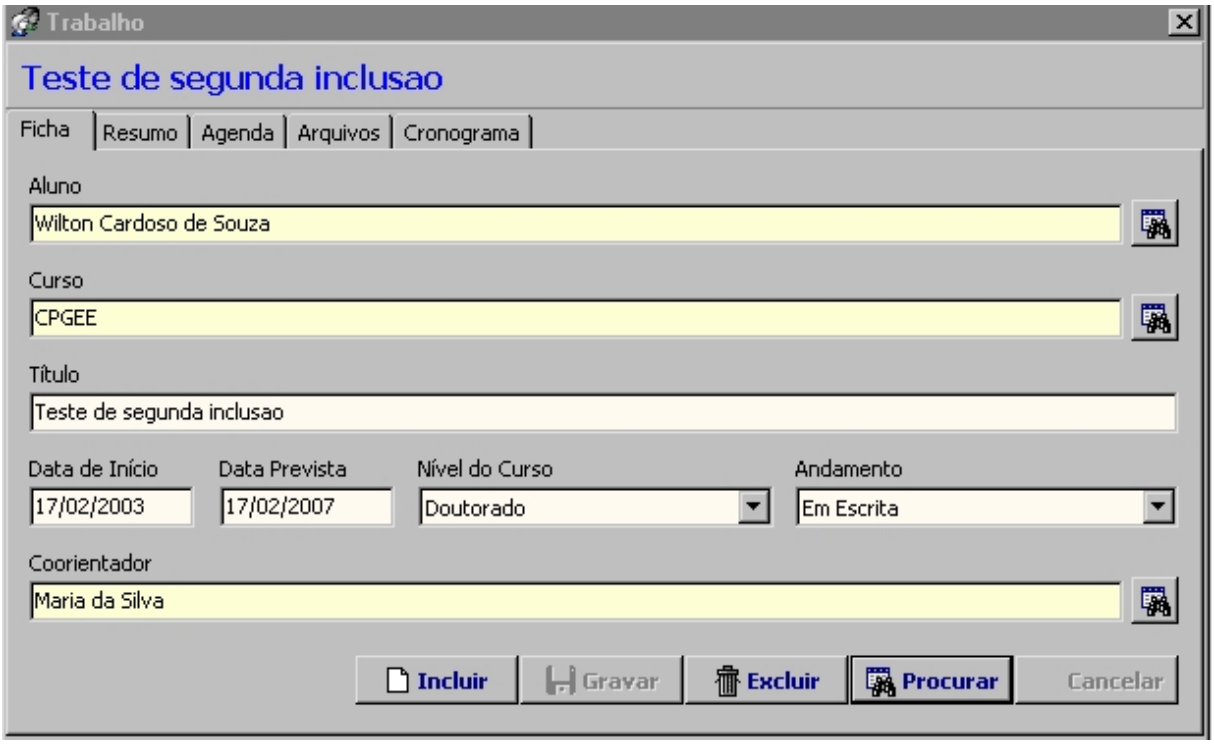
Fluxo Alternativo 2:

1 - usuário poderá incluir, visualizar, comentar e alterar os arquivos recebidos de seus alunos.

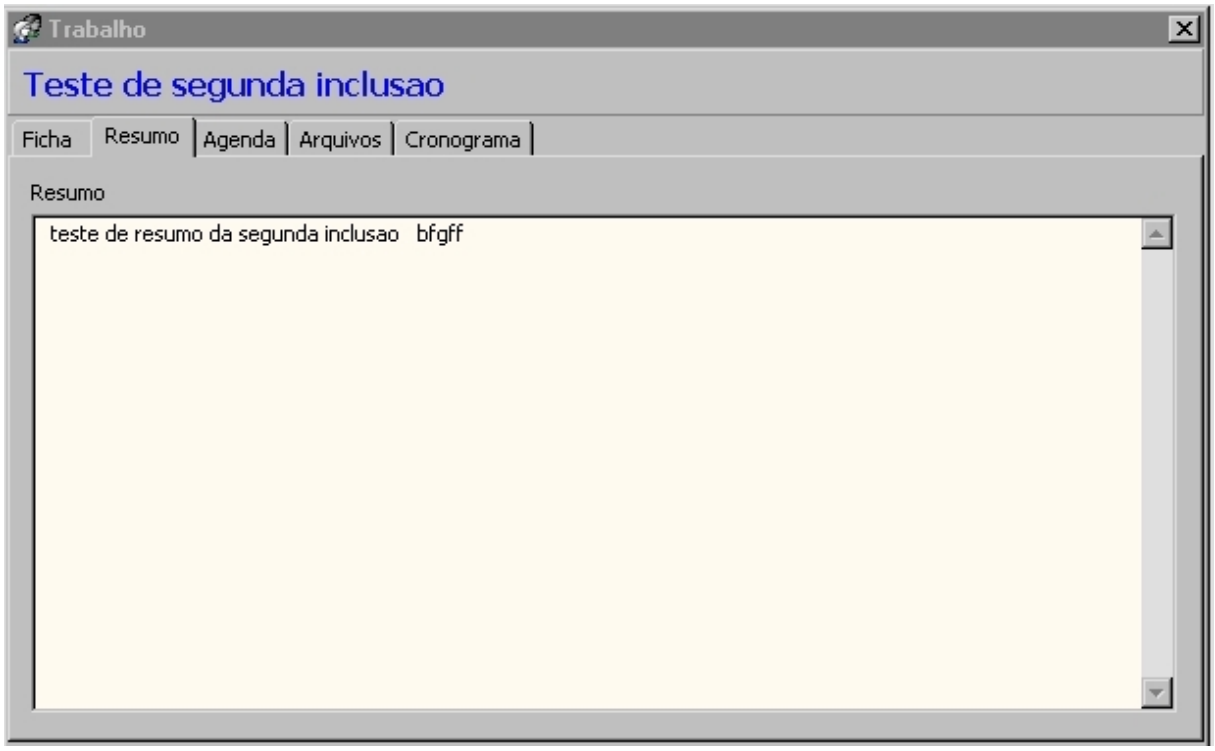
Fluxo Alternativo 3:

1 - usuário poderá avisar o aluno do cronograma a hora que desejar.

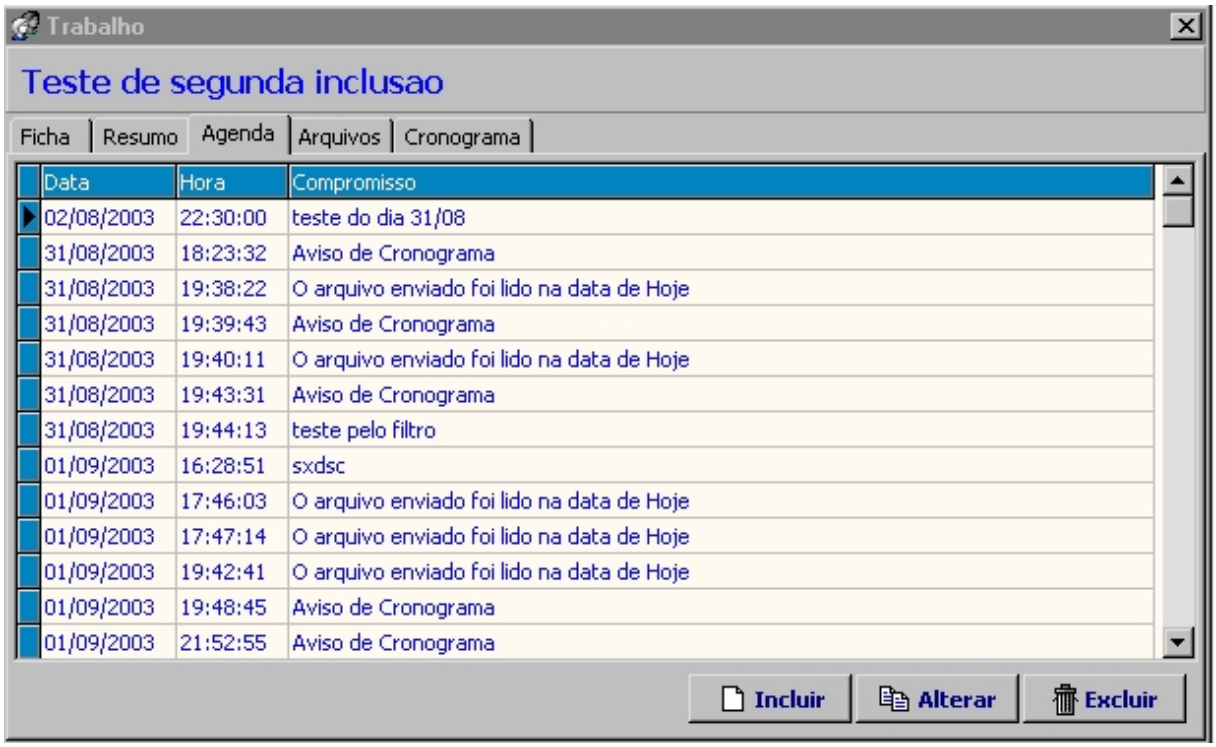
Tela 08 – Trabalho Aluno



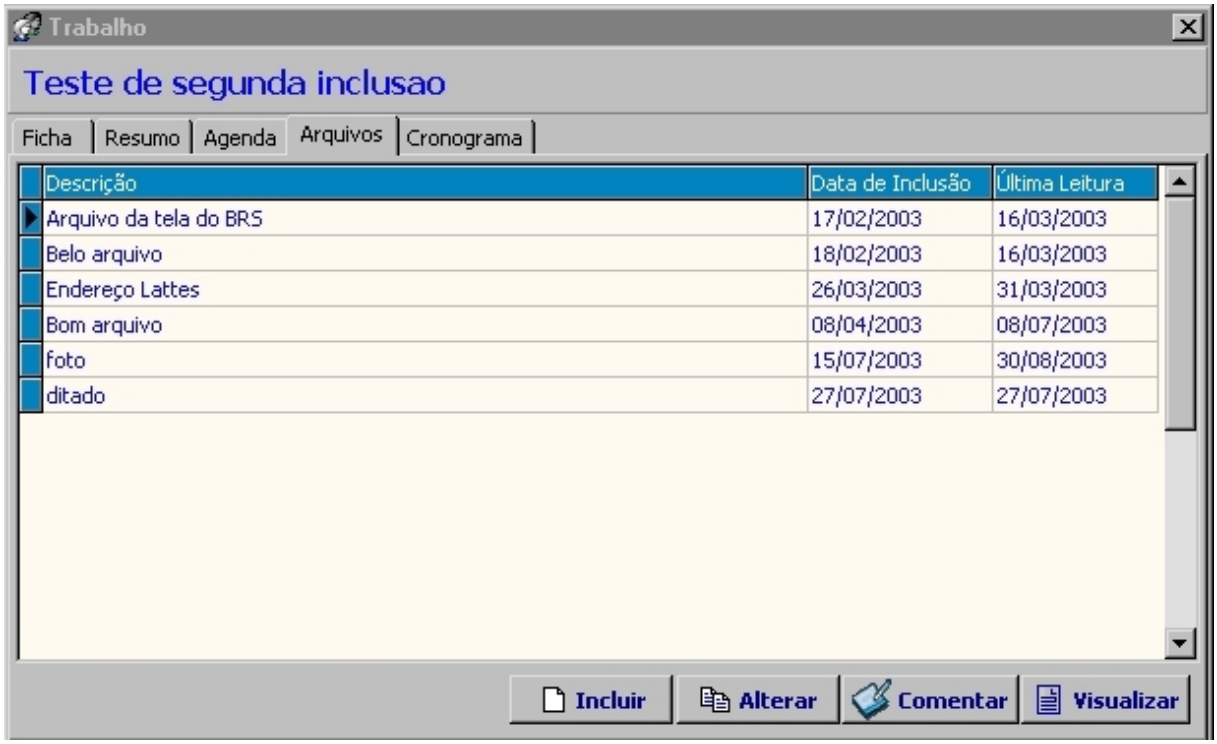
Tela 09 – Resumo Trabalho



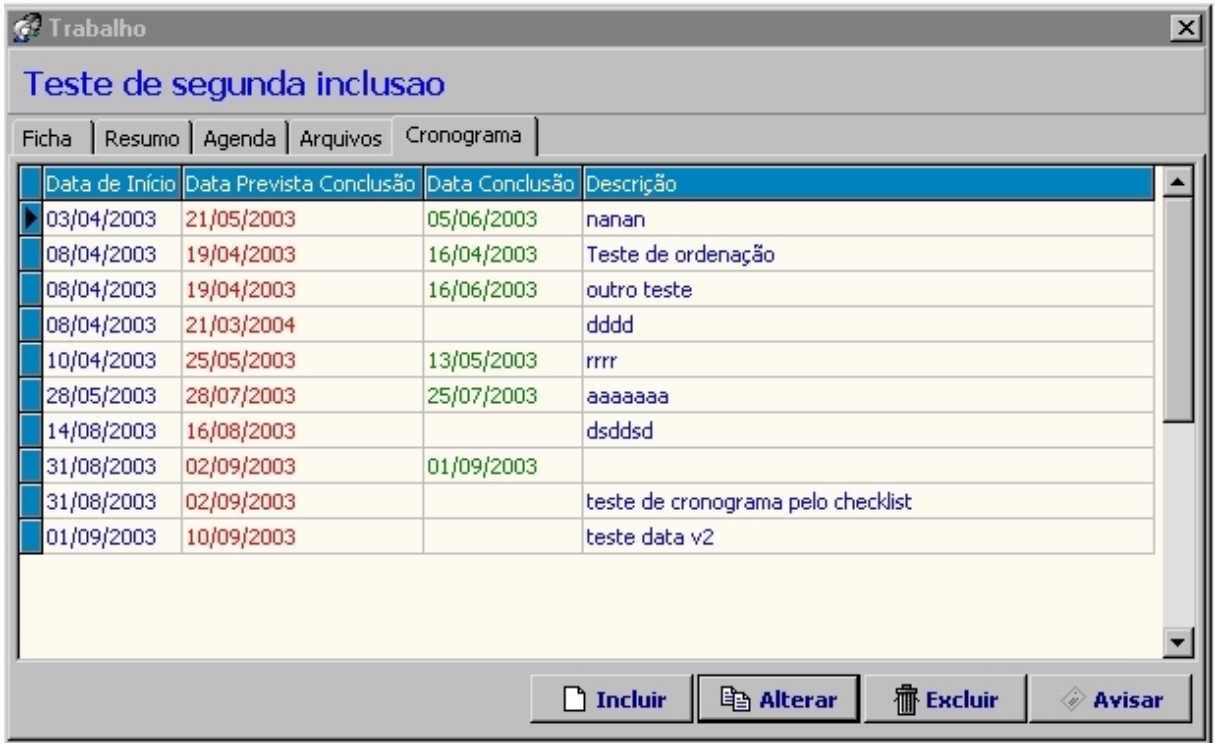
Tela 10 – Agenda Aluno



Tela 11 – Arquivos Aluno



Tela 12 Cronograma Aluno



Nome do Caso de Uso: **Alterar Senha**

Categoria: Processos

Objetivo: usuário poderá alterar sua senha pessoal.

Fluxo principal

1 - usuário informa seu login e sua nova senha.

Tela 13 – Altera Senha

Nome do Caso de Uso: **Backup e Restoure**

Categoria: Processos

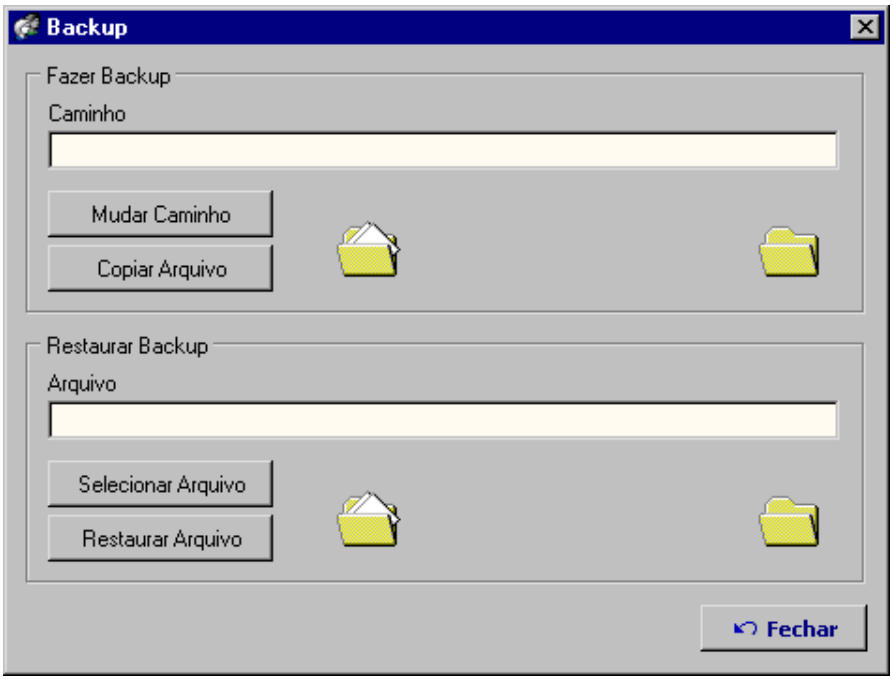
Objetivo: usuário poderá efetuar backup e restaurar os dados cadastrados no banco de dados.

Fluxo principal

1 - usuário informa onde quer copiar os dados.

2 - usuário informa de onde deseja copiar os dados.

Tela 14 Backup e Restoure



Nome do Caso de Uso: **Enviar E-mail**

Categoria: Processos

Objetivo: usuário poderá enviar e-mail aos seus alunos, podendo usar o filtro programado.

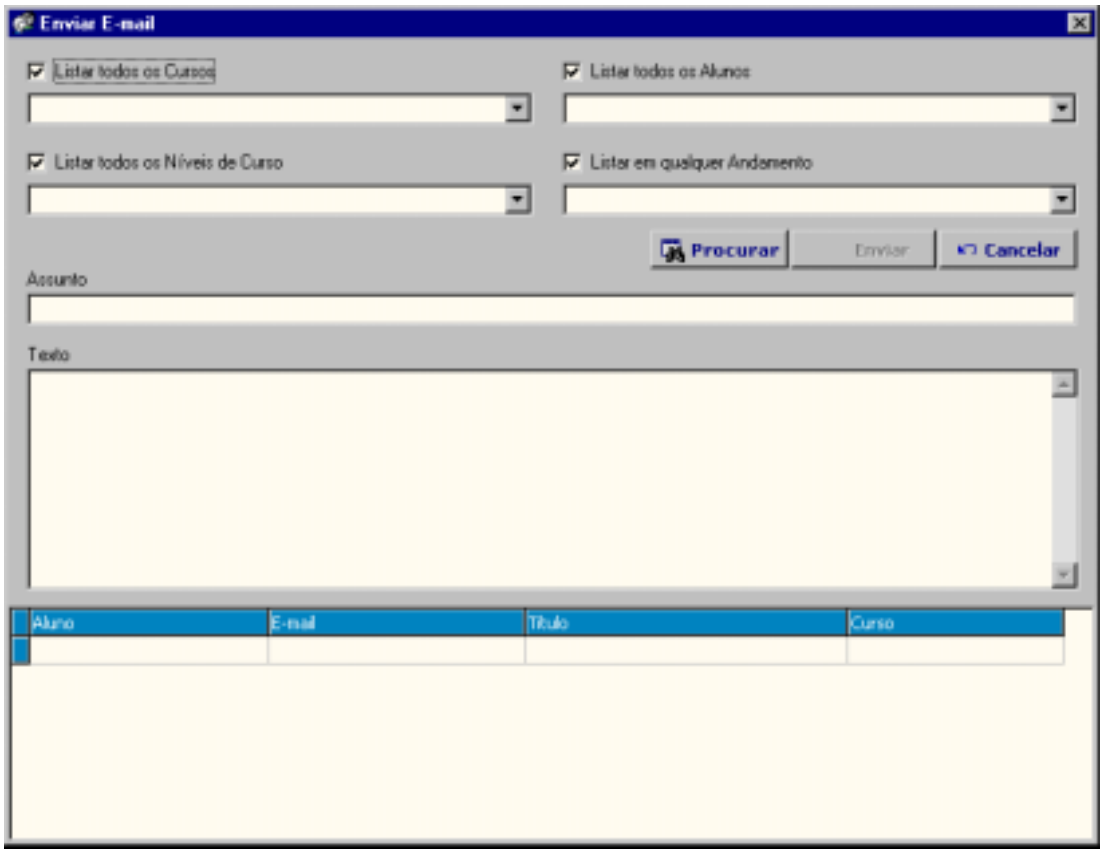
Fluxo principal

1 - usuário escolhe para quem deseja enviar o mail

2 - usuário define o assunto.

3 - usuário define a mensagem.

Tela 15 – Filtro de E-mail



Nome do Caso de Uso: **Procura Compromisso**

Categoria: Processos

Objetivo: procurar compromissos anotados na agenda do aluno.

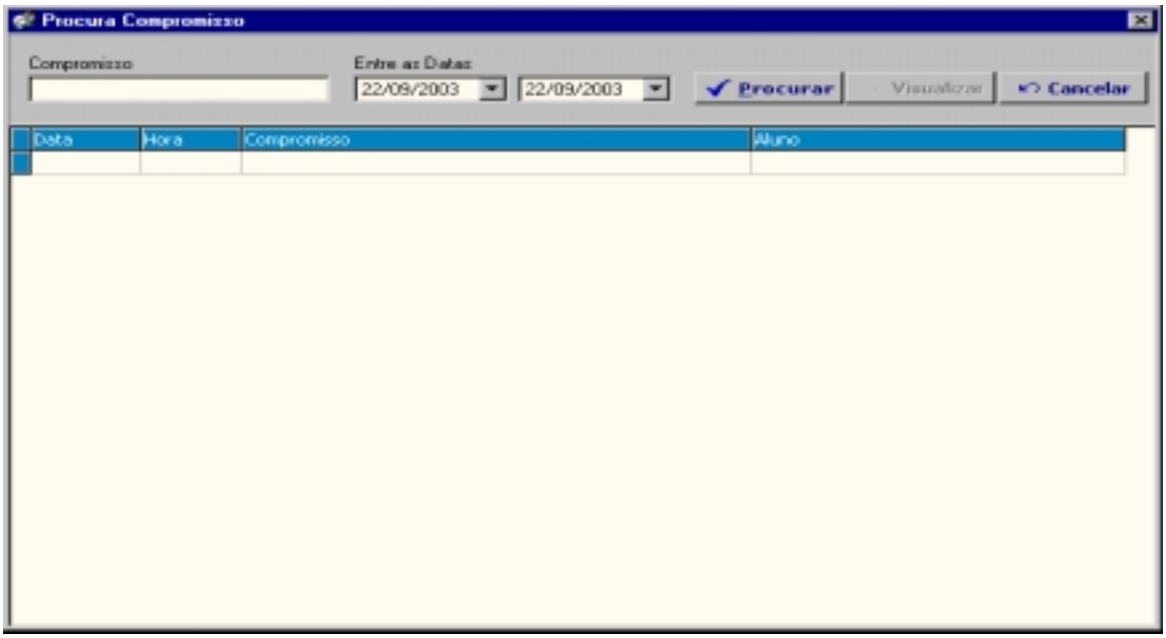
Fluxo principal

1 - usuário escolhe o nome do compromisso que deseja procurar.

2 - usuário define o período que deseja procurar.

3 - usuário pode visualizar o compromisso clicando em visualizar.

Tela 16 – Procura Compromissos



Nome do Caso de Uso: **Relatório Alunos por Curso**

Categoria: Relatórios

Objetivo: usuário poderá efetuar consulta de seus alunos.

Fluxo principal

1 - usuário informa o curso que deseja obter os dados

2 - usuário pode obter os dados de todos os cursos.

Tela 17 – Relatório Alunos por Curso

Nome do Caso de Uso: **Relatório Trabalho por Curso**

Categoria: Relatórios

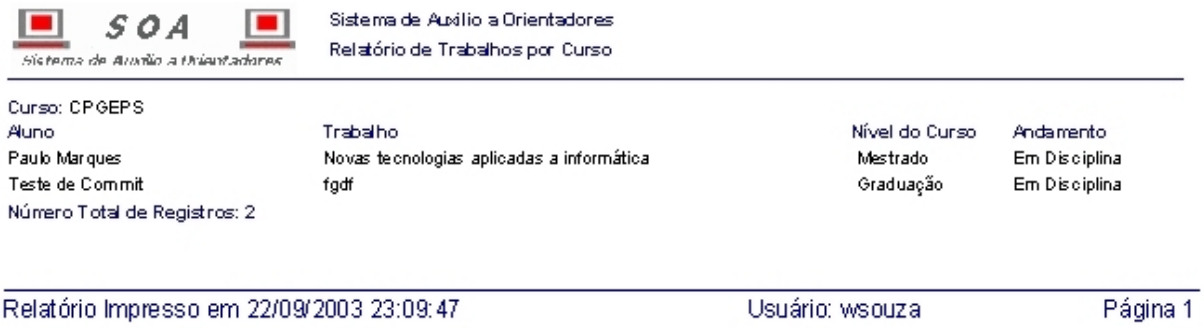
Objetivo: usuário poderá efetuar consulta dos trabalhos de seus alunos.

Fluxo principal

1 - usuário informa o curso que deseja obter os dados

2 - usuário pode obter os dados de todos os cursos.

Tela 18 – Relatório de Trabalhos



Nome do Caso de Uso: **Relatório Trabalho por Coorientador**

Categoria: Relatórios

Objetivo: usuário poderá efetuar consulta dos trabalhos de seus alunos para o coorientador.

Fluxo principal

1 - usuário informa o coorientador que deseja obter os alunos

Tela 18 – Relatório para Coorientador

Nome do Caso de Uso: **Relatório Avançado**

Categoria: Relatórios

Objetivo: usuário poderá efetuar consulta concatenando curso, trabalho, aluno, andamento e data de conclusão do curso e a ordem que deseja visualizar o relatório.

Fluxo principal

1 - usuário escolhe os campos que deseja marcar para o relatório

2 - usuário pode imprimir o relatório

Tela 19 – Relatório Avançado

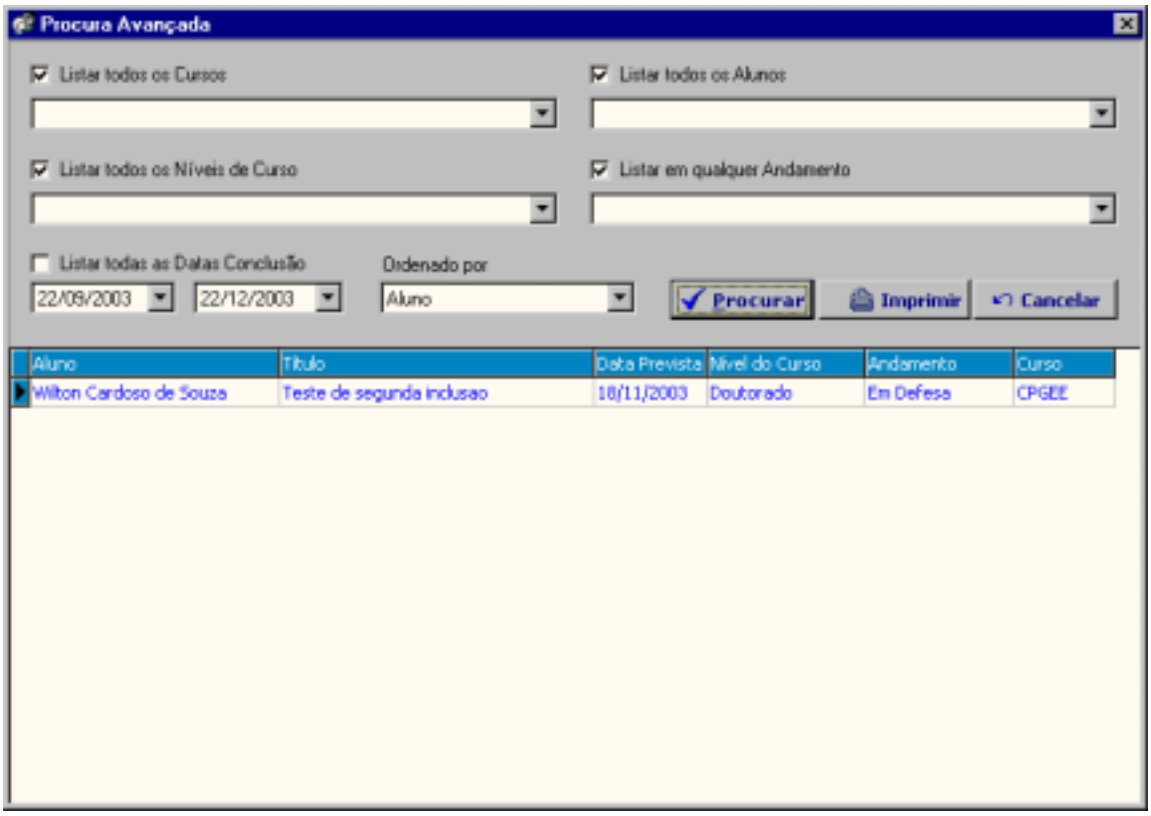


Figura 20 - Tela Principal da Ferramenta de Auxílio a Orientadores