UNIVERSIDADE FEDEREAL FLUMINENSE
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

RAPHAEL DOMINGUES COSTA E WELISSON REICH DE JESUS

PROTÓTIPO DE UM SISTEMA DE INFORMAÇÃO PARA SUPORTE AO CUIDADO MULTIDISCIPLINAR EM INSUFICIÊNCIA CARDÍACA

NITERÓI
2009

RAPHAEL DOMINGUES COSTA E WELISSON REICH DE JESUS

PROTÓTIPO DE UM SISTEMA DE INFORMAÇÃO PARA SUPORTE AO CUIDADO MULTIDISCIPLINAR EM INSUFICIÊNCIA CARDÍACA

Monografia apresentada ao Departamento de Ciência da Computação da Universidade Federal Fluminense, como parte das exigências para a obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. JOSÉ RAPHAEL BOKEHI

NITERÓI
2009

RAPHAEL DOMINGUES COSTA E WELISSON REICH DE JESUS

PROTÓTIPO DE UM SISTEMA DE INFORMAÇÃO PARA SUPORTE AO CUIDADO MULTIDISCIPLINAR EM INSUFICIÊNCIA CARDÍACA

Monografia apresentada ao Departamento de Ciência da Computação da Universidade Federal Fluminense, como parte das exigências para a obtenção do título de Bacharel em Ciência da Computação.

Aprovada em Dezembro de 2009.

BANCA EXAMINADORA

_____

Prof. JOSÉ RAPHAEL BOKEHI – Orientador
UFF

_____

Prof.ª TERESA CRISTINA DE AGUIAR
UFF

_____

Prof.ª LUCIANA FERRAZ THOMÉ
UFF

NITERÓI
2009

AGRADECIMENTOS

À Universidade Federal Fluminense,

Ao Professor José Raphael Bokehi,

A todos os professores do curso de Ciência de Computação,

A Evandro Tinoco Mesquita e Sabrina Bernardez ambos do Centro de Assistência e Pesquisa em Insuficiência Cardíaca do Hospital Universitário Antonio Pedro.

RESUMO

O tratamento da insuficiência cardíaca é uma das áreas da saúde que requer a integração de esforços de vários profissionais. Médicos, fisioterapeutas, enfermeiros e farmacêuticos vêm integrando um grupo multidisciplinar de acompanhamento ao paciente portador de Insuficiência Cardíaca Congestiva (ICC). Este trabalho pressupõe o compartilhamento de várias informações entre estes profissionais. Além disto, várias pesquisas vêm sendo desenvolvidas tomando por base as informações coletadas pelos diversos profissionais que acompanham o paciente com ICC. A inexistência de um sistema de informação que permita o registro adequado destas informações, bem como facilite sua recuperação e compartilhamento entre os diferentes usuários é uma das dificuldades enfrentadas por esta equipe. Neste projeto, propõe-se o desenvolvimento de um protótipo de sistema de informação voltado especificamente para o atendimento multidisciplinar em ICC, visando sua otimização e a facilitação da troca de informação entre o grupo multidisciplinar. Espera-se que, com o uso dessa ferramenta, possam-se suprir estas dificuldades.

Palavras- chave: Sistema de informação em Saúde. Java. Arquitetura de Sistemas Web. JSP. Engenharia de Software. Struts. MVC

ABSTRACT

The treatment of heart failure is one of health activities that requires the integration of many professionals' efforts. Doctors, physiotherapists, nurses and pharmacists are integrating a multidisciplinary group of up to patients with congestive heart failure (CHF). This work involves the sharing of information between these various professionals. Moreover, several studies have been developed and based on the information collected by the various professionals who deal with CHF patients. The lack of an information system that allows proper recording of this information, and facilitates their recovery and sharing among different users is one of the difficulties faced by this team. This project, proposes the development of a prototype information system designed specifically for multidisciplinary treatment on CHF, focusing its optimization and facilitating the exchange of information between the multidisciplinary group. It is expected that the use of this tool can remedy these difficulties.

Keywords: Health Information System. Java. Web Systems Architecture. JSP. Software Engineering. Struts. MVC

Sumário

1. Introdução 9
1.1. Objetivo 10
1.2. Estrutura do trabalho 10
2. Sistemas de Informação em Saúde 12
2.1. Dados, informação, conhecimento e tomada de decisão 12
2.2. Sistemas de Informação em Saúde 14
3. Fundamentos Teóricos 15
3.1. Engenharia da *Web* e a Engenharia de *Software* convencional 15
3.2. Processos de Engenharia de *Software* aplicáveis a projetos *Web* 16
3.3. Atividades Guarda-Chuvas 20
3.4. Padrão de Arquitetura MVC 20
3.5. *Frameworks* 22
4. Engenharia da *Web* 23
4.1. Por que utilizar aplicações Web? 23
4.2. Processo Adotado 24
4.2.1. Arcabouço do Processo 25
4.2.1.1. Comunicação 27
4.2.1.2. Planejamento 28
4.2.1.3. Modelagem 31
4.2.1.4. Construção 35
4.2.1.5. Implantação 37
4.3. Atividades guarda-chuvas utilizadas 37
4.3.1. Gestão de projeto 37
4.3.2. Gerência de Configuração 38
4.3.3. Gerência de Riscos 39
5. Tecnologias Utilizadas 40
5.1. PHP, ASP e JSP 40
5.2. Padrão de Arquitetura Adotado 43
5.2.1. Camada *Model* 43
5.2.2. Camada *View* 44
5.2.3. Camada *Controller* 44
5.3. Struts *Framework* 44

6. O Protótipo Desenvolvido 49
7. Conclusão 52
8. Referências Bibliográficas 53
9. Apêndice 56
9.1. Ata de Reunião 56
9.2. Documento de Requisitos 58
9.3. Documento de Visão 60
9.4. Análise de Riscos 63
9.5. Descrição do caso de uso Cadastrar Paciente 65
9.6. Plano de Testes 69

Capítulo 1

Introdução

A insuficiência cardíaca (IC) é a via final comum da maioria das doenças que acometem o coração. Representa importante problema de saúde pública, considerando-se a prevalência crescente e os índices de hospitalização associados à alta morbimortalidade, sendo um dos mais importantes desafios clínicos atuais na área da saúde (MESQUITA *et al.*, 2002; Sociedade Brasileira de Cardiologia, 2009).

No ano de 2007, as doenças cardiovasculares representaram a terceira causa de internações no SUS, com 1.156.136 hospitalizações. A IC é a causa mais freqüente de internação por doença cardiovascular. O custo socioeconômico da síndrome é elevado, envolvendo gastos com medicamentos, internações repetidas, perda de produtividade, aposentadorias precoces, eventuais cirurgias e, ocasionalmente, transplante cardíaco (MESQUITA *et al.*, 2002).

A elevada morbimortalidade encontrada em pacientes portadores de IC, requer freqüentes internações e re-internações, exigindo, portanto, abordagem multidisciplinar contínua. Essa complexidade supõe um importante desafio para o sistema de saúde uma vez que existem numerosas lacunas e ineficiências no manuseio desses pacientes quando acompanhados em clínicas não especializadas (MESQUITA *et al.*, 2002).

Segundo a Sociedade Brasileira de Cardiologia (2009), dentro desse contexto e objetivando um melhor seguimento dos pacientes portadores de IC torna-se necessário a criação de clínicas especializadas, com corpo multidisciplinar, capaz de fornecer assistência integral aos pacientes. Estas equipes são formadas geralmente por médicos especialistas em IC, enfermeiros, nutricionistas, odontologistas, farmacêuticos, fisioterapeutas e psicologistas.

As clínicas de IC, estruturas multidisciplinares com profissionais especializados na doença, têm conseguido melhorar a adesão dos pacientes não apenas à terapêutica

medicamentosa, mas à restrição hidrossalina e outras medidas não-farmacológicas. Essas mudanças de comportamento têm permitido melhoras nos índices de sobrevida e qualidade de vida dos pacientes, redução do número de hospitalizações relacionadas diretamente à doença e redução dos custos hospitalares (Sociedade Brasileira de Cardiologia, 2009; SILVA *et al*., 2006).

Estas equipes multidisciplinares coletam e analisam um grande volume de dados. São coletados dados que identificam o paciente, informações genéticas, dados de anamnese, exames físicos, comorbidades, exames laboratoriais, diagnósticos, tratamentos utilizados e a evolução da doença.

Os processos de coleta e de inferências sobre estes dados são feitos de forma manual, dificultando o trabalho de pesquisa das diversas áreas médicas envolvidas neste estudo.

1.1. Objetivo

O objetivo deste trabalho é desenvolver um protótipo de um sistema de informação capaz de automatizar as coletas de dados, a geração de consultas e auxiliar na tomada de decisão.

O protótipo que será desenvolvido neste projeto não atenderá o grupo multidisciplinar como um todo, serão desenvolvidos os módulos que darão suporte à equipe médica, ao cadastro de pacientes e ao cadastro de usuários. O sistema final irá facilitar o fluxo de informação entre as equipes do grupo multidisciplinar, concentrando os dados em uma única base de dados.

Com o desenvolvimento deste protótipo, pretende-se estruturar um modelo para os novos módulos. Esta estrutura consiste na definição das atividades e ações que deverão ser utilizadas no processo de desenvolvimento, da arquitetura e tecnologias a serem utilizadas na construção dos novos módulos.

1.2. Estrutura do trabalho

Este trabalho está organizado nos seguintes capítulos:

- Capítulo 2 - Sistemas de Informação em Saúde

Nesse capítulo, são abordados os fatores relevantes para os sistemas de informação para a área de Saúde.

- Capítulo 3 - Fundamentos Teóricos

Neste capítulo, é apresentada a base da teoria aplicada neste projeto.

- Capitulo 4 - Engenharia da Web

Neste capítulo, são apresentadas as atividades de engenharia da Web utilizada no desenvolvimento do sistema.

- Capitulo 5 - Tecnologias Utilizadas

Neste capítulo, são apresentadas as tecnologias utilizadas no desenvolvimento do sistema.

- Capítulo 6 – O Protótipo Desenvolvido

Neste capítulo, é apresentado o protótipo desenvolvido e suas funcionalidades.

- Capítulo 7 - Conclusão

Nesse capítulo, são destacadas as contribuições deste trabalho, bem como as melhorias que podem ser feitas em trabalhos futuros.

Capítulo 2

Sistemas de Informação em Saúde

Neste capítulo, são apresentadas as relações entre dados, informação e conhecimento. Apresenta-se ainda o papel dos sistemas de informação, em particular os aplicados em Saúde, como ferramenta auxiliadora no gerenciamento das informações e na tomada de decisão.

2.1. Dados, informação, conhecimento e tomada de decisão

Compreendem-se dados como uma base para gerar informação, ou seja, matéria-prima para a informação. O dado é informação bruta e sem associatividade (FERREIRA, 2001). Dado é uma estrutura fundamental sobre a qual um sistema de informação é construído. Símbolo intencionalmente destacado, para representar uma característica ou propriedade da realidade a ser tratada (POMPILHO, 1995).

Define-se informação como o produto obtido de determinada combinação e interpretação de dados. A informação é criada a partir dos dados inseridos em um contexto. Para se obter a informação um processo deve ser acionado. Este processo faz parte de um mecanismo que envolve desde a forma com que os dados estão agregados, suas restrições e até as relações estabelecidas entre eles. Portanto para se ter informação, é necessário antes conhecer no mínimo o contexto essencial para o qual servirá. Com base neste aspecto, buscam-se os dados necessários, efetuam-se as agregações e relacionamentos, de maneira a permitir que os processos possam fornecer o significado e a interpretação esperada (TONSIG, 2000).

O conhecimento é um tipo de informação mais estruturada, com alto grau de refinamento e carregada de entendimento sobre um domínio. Conhecimento pode ser definido como sendo informações que foram analisadas e avaliadas sobre a sua confiabilidade, sua relevância e sua importância. Neste caso, o conhecimento é obtido pela interpretação e integração de vários dados e informações, ou seja, os insumos provenientes das diversas fontes são analisados e combinados na síntese de um produto final, o conhecimento (DAVENPORT, 2000).

No processo de tomada de decisão é importante ter disponíveis dados, informações e conhecimentos, mas esses normalmente estão dispersos, fragmentados e armazenados na cabeça dos indivíduos e sofrem interferência de seus modelos mentais. Nesse momento, o processo de comunicação e o ambiente multidisciplinar têm papéis relevantes para resolver algumas das dificuldades essenciais no processo de tomada de decisão. Pelo processo de comunicação, pode-se buscar o consenso que permitirá prever a adequação dos planos individuais de ação. Pelo trabalho em ambiente multidisciplinar, pode-se obter o maior número de informações e perspectivas de análise distintas, sendo validada a proposta mais convincente no confronto argumentativo dos demais (ANGELONI, 2003). Na figura 2.1 são apresentados os elementos do processo de tomada de decisão.

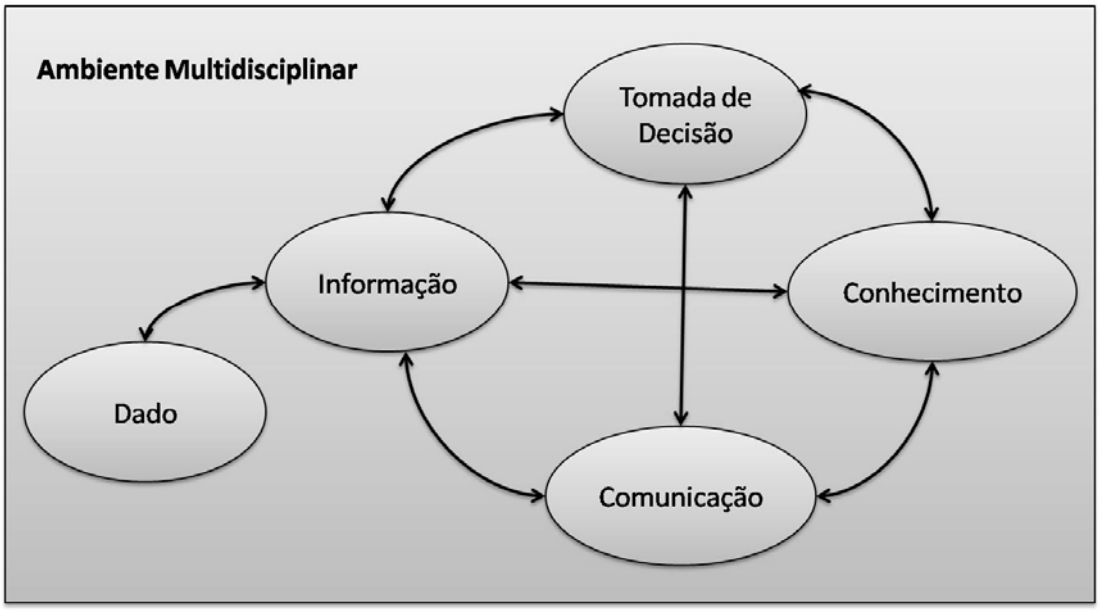


Figura 2.1: Representação gráfica dos elementos do processo de tomada de decisão (Adaptado de ANGELONI, 2003).

2.2. Sistemas de Informação em Saúde

Sistema de Informação pode ser entendido como um conjunto de procedimentos que buscam transmitir informações entre pessoas e órgãos através de qualquer meio (BENITO, 2001).

Sistema de Informação em Saúde é um mecanismo de coleta, processamento, análise e transmissão de informação necessária para se organizar e operar os serviços de saúde e, também atuam na investigação, no planejamento e na pesquisa, com vistas ao controle de doenças. O propósito do sistema de informação em saúde é selecionar os dados pertinentes a esses serviços e transformá-los na informação necessária para o processo de decisões, próprio das organizações e indivíduos que planejam, financiam, administram, provêem, medem e avaliam os serviços de saúde (ANGELONI, 2003).

A pesquisa realizada pelas equipes multidisciplinares requer fácil acesso a informação no local de uso e no momento em que é necessária para uma intervenção mais ágil e precisa.

Em um sistema várias partes trabalham juntas visando um objetivo em comum. Em um Sistema de Informação não é diferente, porém o objetivo é um fluxo mais confiável e menos burocrático das informações. Em um Sistema de Informação bem construído, suas principais vantagens são: otimização do fluxo de informação permitindo maior agilidade e organização, redução de custos operacionais e administrativos, ganho de produtividade, maior integridade, veracidade da informação, estabilidade e maior segurança de acesso à informação.

Neste projeto a equipe multidisciplinar, formada por médicos, fisioterapeutas, farmaceuticos e enfermeiros, necessita acessar informações que foram coletadas por uma determinada parte desta equipe, ou seja, existe um fluxo de informações. Por exemplo, os médicos necessitam obter informações sobre exames realizados pelos fisioterapeutas. Pelos métodos adotados hoje por esta equipe, esta troca de informação é realizada manualmente, com os resultados dos exames registrados em papel, o que dificulta o tráfego de informações. Com a implantação do sistema proposto neste projeto, o fluxo de informação ocorrerá de forma mais ágil, já que todos os dados dos pacientes serão incluídos no sistema, ou seja, não seria mais necessário esperar pela entrega do resultado de um exame.

Capítulo 3

Fundamentos Teóricos

Neste capítulo, é apresentada a base da teoria para o desenvolvimento de aplicações voltadas para *Web*. São apresentados como a Engenharia de *Software* pode ser aplicada no desenvolvimento das aplicações, as atividades guarda-chuvas e como estas atividades podem auxiliar e garantir um melhor desenvolvimento destas aplicações, o padrão de arquitetura *Model-View-Controller* (MVC) e o conceito de *framework*.

3.1. Engenharia da *Web* e a Engenharia de *Software* convencional

As expectativas e as exigências sobre as aplicações *Web* têm aumentado significativamente ao longo dos anos. Como resultado, a concepção, o desenvolvimento, a implantação e a manutenção destas aplicações, tornaram-se mais complexas e difíceis de gerir.

O processo de desenvolvimento de aplicações *Web* quebra o desenvolvimento em pedaços menores e mais gerenciáveis. Além disto, oferece técnicas para ajudar os desenvolvedores e analistas a gerir com êxito o projeto de aplicações voltadas para *Web* (DANIEL, 2008).

Alguns métodos e conceitos da engenharia de *software* convencional podem ser aplicados na engenharia da *Web*, porém alguns destes métodos podem ser modificados ou novos métodos podem ser adicionados para suprir as demandas específicas de projetos voltados para *Web*.

Os sistemas baseados na *Web* possuem algumas características especiais como a diversidade de usuários e arquitetura altamente especializada. Além disto, estes sistemas possuem atributos de qualidade como a usabilidade, navegação, facilidade de manutenção, segurança e confiabilidade (PRESSMAN, 2006).

3.2. Processos de Engenharia de *Software* aplicáveis a projetos *Web*

O processo de desenvolvimento de aplicações *Web* é um conjunto de atividades, que devem ser seguidas pelos analistas e desenvolvedores, para a construção de uma aplicação de qualidade e que atenda as demandas do cliente. Para que um processo possa ser considerado eficaz, deverá ser bem planejado, definir claramente este conjunto de atividades e garantir que manutenções corretivas e evolutivas, com base no *feedback* dos usuários, possam ser feitas sem dificuldades. Além disto, deve ser adaptável, ou seja, as atividades de engenharia de *software* a serem realizadas devem ser adequadas ao cronograma, às restrições do projeto, à equipe e ao problema a ser resolvido. Em última instância, para que um processo de desenvolvimento seja considerado bom e eficaz, deve satisfazer as necessidades do cliente (PRESSMAN, 2006; DANIEL, 2008).

Em aplicações *Web*, podem ser adotados diversos processos, onde cada um engloba um conjunto de atividades específicas, fazendo com que, dependendo da aplicação a ser construída, um processo possa ser mais indicado do que o outro. Dentre os utilizados destacam-se os processos ágeis (*Extreme Programming* - *XP* e *Scrum*), o unificado (*Rational Unified Process* - RUP) e o genérico, adaptável para vários tipos de projetos, sugerido por PRESSMAN (2006).

O XP é um processo ágil para pequenas e médias equipes e que possui as seguintes características: utiliza métodos informais, codificação simples e gera o menor número possível de produtos de trabalho. O XP possui quatro atividades de arcabouço: planejamento, projeto, codificação e testes. Na atividade de planejamento é criado um conjunto de histórias do usuário, que descrevem as características e funcionalidades de uma aplicação. Estas histórias serão utilizadas, pelos programadores, para desenvolver a aplicação. Na atividade de projeto são definidas as diretrizes de implementação das histórias definidas na etapa anterior. A atividade de codificação é iniciada com o planejamento dos testes unitários, seguido da

codificação. Após a elaboração do código, o *software* é submetido aos testes unitários. Na atividade de testes são realizados testes de integração e de validação (WELLS, 2009).

O RUP é um processo unificado que fornece uma técnica disciplinar para determinar tarefas e responsabilidades, organizando o processo de desenvolvimento. Sua meta é garantir a produção de uma aplicação de alta qualidade, que contemple os requisitos solicitados pelo usuário final, com cronogramas e orçamentos pré-definidos. O RUP possui quatro fases: concepção, elaboração, construção e transição. Na fase de concepção são identificados os riscos críticos, levantados os requisitos e os casos de uso são esboçados. Na fase de elaboração os casos de uso esboçados na fase anterior são aprimorados, são identificados os riscos significativos, já que os críticos foram mapeados na fase anterior e é definida a arquitetura do projeto. Na fase de construção os riscos levantados nas fases anteriores são monitorados e uma versão operacional do projeto é construída. Na fase de transição o sistema desenvolvido é entregue, são coletados possíveis erros e o *feedback* do usuário (VALLATI e DOMINGOS, 2003; MURTA, 2009).

Segundo PRESSMAN (2006), o arcabouço do processo genérico estabelece o alicerce para um processo de *software* completo, pela identificação de um pequeno número de atividades de arcabouço aplicáveis a todos os projetos, independentemente de seu tamanho ou complexidade. Além disso, engloba um conjunto de atividades guarda-chuvas que são aplicáveis durante todo o processo de *software*.

O processo genérico é constituído de cinco atividades: comunicação, planejamento, modelagem, construção e implantação. Cada atividade do processo é composta por ações bem definidas e que geram artefatos ou produtos de trabalho, que serão utilizados como guia para os desenvolvedores durante todo o desenvolvimento. Os artefatos gerados podem ser modelos, documentos, dados, relatórios, formulários, programas etc. Cada ação é constituída por um conjunto de tarefas de trabalho.

A atividade de arcabouço comunicação envolve alta comunicação e colaboração com os interessados e abrange o levantamento de requisitos e outras ações relacionadas. O objetivo desta atividade é identificar as necessidades e os objetivos dos interessados em relação ao sistema, através de técnicas que visam facilitar a comunicação entre os engenheiros e os outros interessados (PRESSMAN, 2006).

As seguintes técnicas podem ser utilizadas na atividade de comunicação: entrevistas, aplicação de questionários e formulários, leitura de documentos e reuniões. Estas técnicas são aplicadas na coleta de requisitos e sempre que um incremento precisa ser entregue.

A atividade planejamento estabelece um plano de trabalho, desenvolve estimativas para avaliar a possibilidade das datas de entregas desejadas, considera os riscos, define um cronograma e estabelece mecanismos de monitoramento e controle (PRESSMAN, 2006).

Mesmo no caso de desenvolvimento de pequenos sistemas baseados na *Web*, a atividade de planejamento precisa ocorrer, riscos precisam ser considerados, um cronograma deve ser estabelecido e controles precisam ser definidos como forma de evitar falhas e erros.

A atividade de arcabouço planejamento engloba um conjunto de ações que devem ser realizadas para que esta fase esteja concluída. Em suma, o escopo desta atividade consiste em: definição do escopo do sistema, da estratégia de entregas, do arcabouço de processo, elaboração do documento de Análise de Risco e estabelecimento de um cronograma. Este conjunto de ações juntos formam o plano de projeto.

A atividade modelagem precisa ser capaz de representar a informação que o *software* transforma, a arquitetura e funções que permitam que esta transformação ocorra, as características que os usuários desejam e o comportamento do sistema à medida que a transformação ocorre (PRESSMAN, 2006).

Na atividade modelagem duas classes de modelo são criadas: modelo de análise e modelo de projeto.

O modelo de análise engloba uma variedade de representações *Unified Modeling Language* (UML*)*, que atuam em determinadas questões como: que informação deve ser manipulada, que funções devem ser realizadas pelo usuário final, e quais comportamentos devem ser exibidos pelo sistema quando ele apresenta a informação e realiza funções? A modelagem de análise é o primeiro degrau na solução do problema. Permite que os profissionais entendam melhor o problema e possam utilizar a estratégia de dividir para conquistar, ou seja, um problema complexo e grande é quebrado em problemas menores e mais simples e que possam ser facilmente entendidos.

A modelagem de análise é baseada em cenários e em classes. Os cenários são representados por casos de uso. Os casos de uso descrevem a interação entre o usuário final e o sistema. Ao desenvolver os casos de uso, as classes de análise são coletadas através de um trabalho de exame dos enunciados do problema ou executando uma análise gramatical das narrativas dos casos de uso. As classes de análise são extraídas do domínio do negócio em questão e não levam em consideração detalhes inerentes à tecnologia de implementação. À medida que a aplicação é desenvolvida, as classes de análise são incrementadas com novos detalhes.

A modelagem de projeto é subdividida em outros projetos menores que englobam atividades específicas. Estes projetos são:

- Projeto de Interface, que descreve a estrutura e a organização da interface com o usuário.
- Projeto de Conteúdo, que define o conteúdo apresentado na aplicação.
- Projeto de Navegação, que define os fluxos de navegações entre as funcionalidades da aplicação.
- Projeto de Arquitetural, que trata do modo pelo qual a aplicação é estruturada para gerir a interação com o usuário, manipular tarefas de processamento interno, efetuar navegação e apresentar conteúdo.
- Projeto de Componente, que desenvolve a lógica de processamento detalhada necessária para implementar os componentes funcionais.

A atividade construção engloba duas ações: codificação e testes. Estas etapas auxiliam na construção do *software* operacional pronto para ser entregue ao usuário final. A etapa de codificação é composta basicamente por geração de código. Na etapa de testes, o objetivo é exercitar o *software* a fim de encontrar erros e corrigi-los. Um planejamento é criado para acomodar os casos de teste e os resultados das execuções dos casos de teste.

Nesta etapa, os seguintes tipos de testes podem ser aplicados: teste de conteúdo, de interface, de navegação, de componente, de configuração, de desempenho e de segurança.

Após a execução das atividades presentes no planejamento, caso sejam encontrados erros, estes deverão ser corrigidos e os testes deverão ser refeitos para que seja garantido que as correções realizadas tenham solucionado os erros e que nenhum erro novo tenha sido introduzido. Na Figura 3.1 é apresentado o processo de testes.

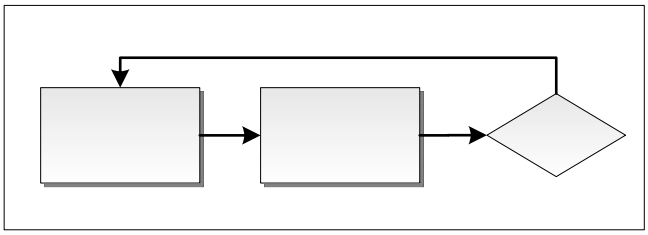


Figura 3.1: Representação gráfica do processo de testes (Adaptado de PRESSMAN, 2006).

A atividade de implantação engloba duas ações: entrega e *feedback*. Esta atividade ocorre várias vezes ao longo do desenvolvimento, já que os incrementos ficam prontos e o *software* caminha para ficar completo. Cada ciclo de entrega fornece ao cliente um incremento de *software* operacional com determinado número de funcionalidades. Após a entrega é realizado um teste de aceitação que culmina no termo de aceite do cliente. A última ação desta atividade é o recebimento do *feedback* que pode determinar mudanças nas funcionalidades e nas características da aplicação para o incremento seguinte.

O teste de aceitação é realizado tendo como guia o manual desenvolvido. Este manual determina um fluxo para a utilização das funcionalidades da aplicação de forma correta, contidas no incremento entregue.

3.3. Atividades Guarda-Chuvas

Segundo PRESSMAN (2006), algumas atividades guarda-chuvas são essenciais para o sucesso da aplicação. Estas atividades são aplicáveis durante todo o processo de engenharia e ajudam no desenvolvimento de aplicações corretas, no cumprimento dos prazos, na gerência de versões dos artefatos gerados e no controle e gerenciamento dos riscos que possam afetar o projeto. Estas atividades complementam as atividades já definidas para o arcabouço.

As atividades guarda-chuvas que complementam o arcabouço são: gestão de projeto, gerência de risco, garantia da qualidade de software, revisões técnicas formais, medição, gestão de configuração de software e gestão de reusabilidade.

3.4. Padrão de Arquitetura MVC

O padrão de arquitetura MVC é um padrão que separa de maneira independente, uma aplicação em três partes: a parte *Model*, que representa as regras de negócio, o acesso e a atualização à base de dados, a *View*, que representa a interface com o usuário ou outros sistemas e o *Controller*, que representa o controle e a coordenação de envio de requisições feitas entre a *View* e o *Controller*. Isso facilita a manutenção do sistema, pois se uma alteração for necessária, basta apenas modificar a parte específica (BAPTISTELLA, 2009).

Nas aplicações web, o padrão MVC sofre algumas alterações em relação ao padrão original. No padrão original, a camada *View* é notificada quando modificações na camada *Model* acontecem, devido a todos os recursos estarem no mesmo servidor e os clientes terem uma conexão aberta com esse servidor. Na figura 3.2 é apresentado o modelo original do padrão MVC. Pode-se notar que todas as camadas podem se comunicar entre si, o que não acontece nas aplicações web devido ao desacoplamento entre o cliente e o servidor, ou seja, os recursos estão distribuídos em diversos servidores (SOUZA, 2004; HUSTED 2004). O padrão MVC modificado para a web é apresentado na figura 3.3, ilustrando a comunicação entre as camadas.

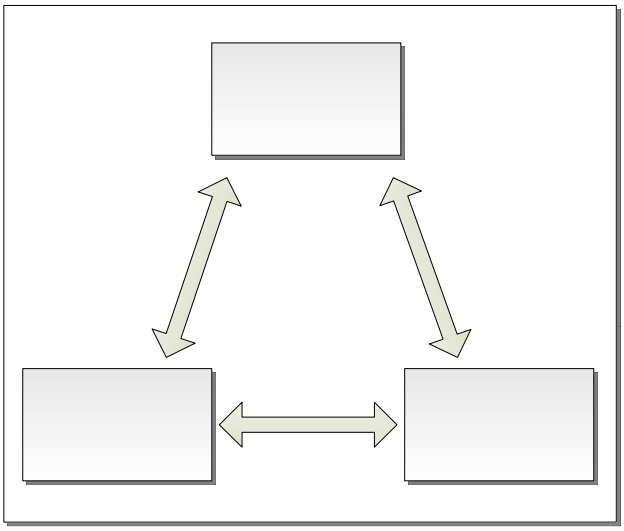


Figura 3.2: Representação gráfica do padrão MVC original (Adaptado de HUSTED, 2004).

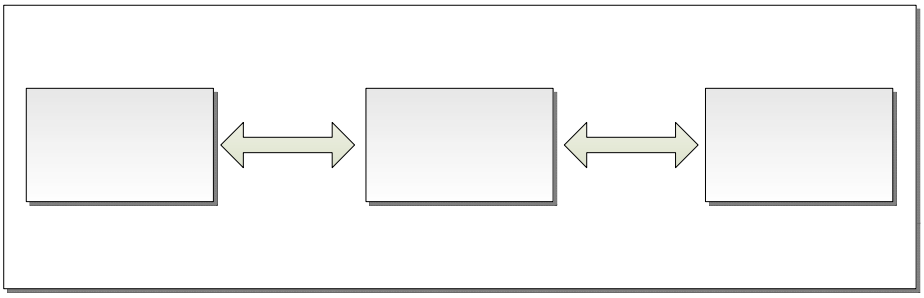


Figura 3.3: Representação gráfica do padrão MVC Web (Adaptado de HUSTED, 2004).

3.5. *Frameworks*

Um *framework* orientado a objetos é um conjunto de classes e interfaces que incorpora um projeto abstrato. Provê uma infra-estrutura genérica para construção de aplicações dentro de uma família de problemas semelhantes, de forma que esta infra-estrutura genérica deve ser adaptada para a geração de uma aplicação específica. O conjunto de classes que forma o *framework* deve ser flexível e extensível para permitir a construção de várias aplicações com pouco esforço, especificando apenas as particularidades de cada aplicação (TANI, 2009).

Atualmente no mercado, existem inúmeros *frameworks* que implementam o padrão de arquitetura MVC associados a diversas tecnologias, por exemplo: ASP Xtreme Evolution e ASP.NET (associados a tecnologia *Active Server Pages* - ASP), Apache Struts, VRaptor, Spring MVC e JSF (associados a tecnologia *Java Server Pages* - JSP), PRADO e XPT Framework (associados a tecnologia *Hypertext Processor* - PHP), entre outras.

Capítulo 4

Engenharia da *Web*

Neste capítulo são apresentadas as metodologias utilizadas neste projeto, referentes à engenharia da *Web*, bem como as justificativas para o uso das metodologias.

4.1. Por que utilizar aplicações Web?

Aplicações *Web* é um termo para designar sistemas projetados para serem utilizados através de um navegador, em um ambiente de *internet* ou *intranet*. Se um dia a *internet* era composta somente de páginas estáticas com conteúdos institucionais, hoje ela oferece uma infinidade de aplicações com conteúdos dinâmicos e personalizados.

De acordo com TEMPLE *et al.* (2009), diversas tecnologias possibilitaram essa revolução, seja para construir um simples *site* com conteúdo dinâmico ou para construir um complexo sistema de informação, é necessária a utilização de ferramentas que possibilitem consultas a bancos de dados, integração com sistemas corporativos, entre outras inúmeras funcionalidades.

O principal motivo de construir aplicações baseadas na *Web* é devido às aplicações *Web* possibilitarem o acesso de forma distribuída, ou seja, os usuários finais podem acessá-las de qualquer terminal que possua um navegador *Web*. Isto faz com que estes usuários ganhem mobilidade e não fiquem presos a um terminal específico.

As principais vantagens destas aplicações são:

- A Interface *Hyper Text Markup Language* (HTML) é reconhecida por uma grande gama de usuários já acostumados com o funcionamento dos navegadores.
- Desenvolvimento, manutenção e atualização centralizada da aplicação. Não é necessária a instalação da aplicação em diversos equipamentos diferentes, basta colocá-lo no servidor para que os usuários a acessem.
- Escalabilidade no processamento. Se houver necessidade de aumentar o poder de processamento, basta fazer isto no servidor.

As principais desvantagens destas aplicações são:

- A interface HTML pode ser um problema, pois não há uma padronização entre os diversos navegadores e a aplicação poderia ser exibida de uma maneira diferente dependendo do navegador utilizado.
- A entrada de uma grande massa de dados é prejudicada na interface HTML, pois não existe uma maneira padrão de criar máscaras de entrada de dados.
- A interface HTML não é rica em controles gráficos e peca no quesito posicionamento, fazendo com que o visual da aplicação possa não ficar tão atrativo.

4.2. Processo Adotado

O processo utilizado neste projeto foi uma adaptação do processo genérico citado na seção 3.2. Este processo baseia-se em atividades de arcabouço bem definidas, o que facilita o desenvolvimento de um *software*.

O processo ágil define um conjunto de atividades, porém não da ênfase a modelagem de análise e de projeto, que é de extrema importância neste projeto, já que este sistema será expandido com a adição de novos módulos. Para a criação destes novos módulos serão necessárias constantes consultas a toda documentação gerada nos módulos anteriores.

O processo unificado possui algumas características que inviabilizaram a utilização do mesmo neste projeto. Tais características como: requisitos não serem completamente definidos antes do projeto, o projeto não ser completamente definido antes do início da

programação, a modelagem não ser feita de forma completa e precisa, a programação não ser uma tradução mecânica do modelo para o código, as iterações não duram meses, mas sim semanas e o planejamento não ser especulativo, mas sim refinado durante o projeto (MURTA, 2009).

4.2.1. Arcabouço do Processo

O arcabouço do processo adotado possui cinco atividades básicas: comunicação, planejamento, modelagem, construção e implantação. Na figura 4.1 são apresentadas as atividades do arcabouço, as ações e os produtos de trabalho relacionados a cada atividade, bem como as atividades guarda-chuvas que complementam o arcabouço do processo.

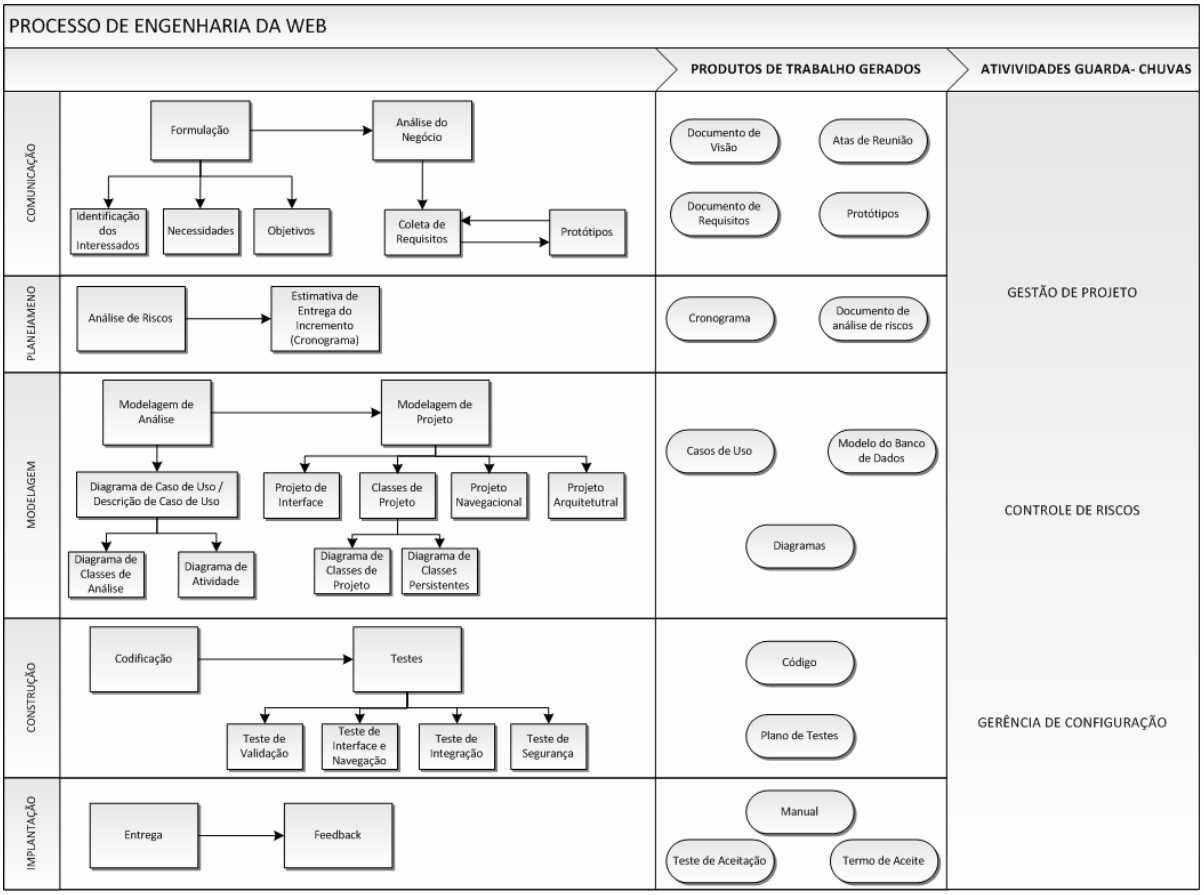


Figura 4.1: Representação gráfica do Processo utilizado.

O processo utilizado no desenvolvimento do Sistema de Informação em Saúde auxilia na atividade de comunicação (elicitação de requisitos e na comunicação entre os vários

envolvidos no processo de desenvolvimento), de planejamento, de modelagem (documentação do sistema), de construção (casos de teste) e de implantação, facilitando o processo de futuras manutenções e evoluções.

Algumas atividades como a modelagem e a construção utilizadas neste processo, foram adaptadas da engenharia de *software* convencional para atender as características especiais dos sistemas baseados na *Web*, como por exemplo, a segurança na navegação entre as funcionalidades do sistema.

O modelo de ciclo de vida proposto é o incremental, pois acomoda as incertezas do projeto, ou seja, não é muito rígido, permitindo que o escopo do projeto sofra alterações ao longo do desenvolvimento. Este processo é apresentado na figura 4.2. Neste modelo é enfatizado o desenvolvimento em fases, através da realização de mini-projetos, chamados de incrementos, que adicionam funcionalidades ao projeto. Os incrementos são liberados logo que desenvolvidos. Os primeiros incrementos contêm o núcleo do produto. Após o incremento ser liberado, este é analisado pelo cliente que fornece um *feedback*. Através deste *feedback* alterações podem ser feitas e novas funcionalidades podem ser requeridas. Estas alterações serão implementadas nos incrementos posteriores. Este processo é repetido até que o produto esteja completo. Associado ao ciclo de vida incremental, em determinados incrementos, o modelo de prototipagem foi utilizado, para diminuir o grau de incerteza dos interessados, em relação aos requisitos.

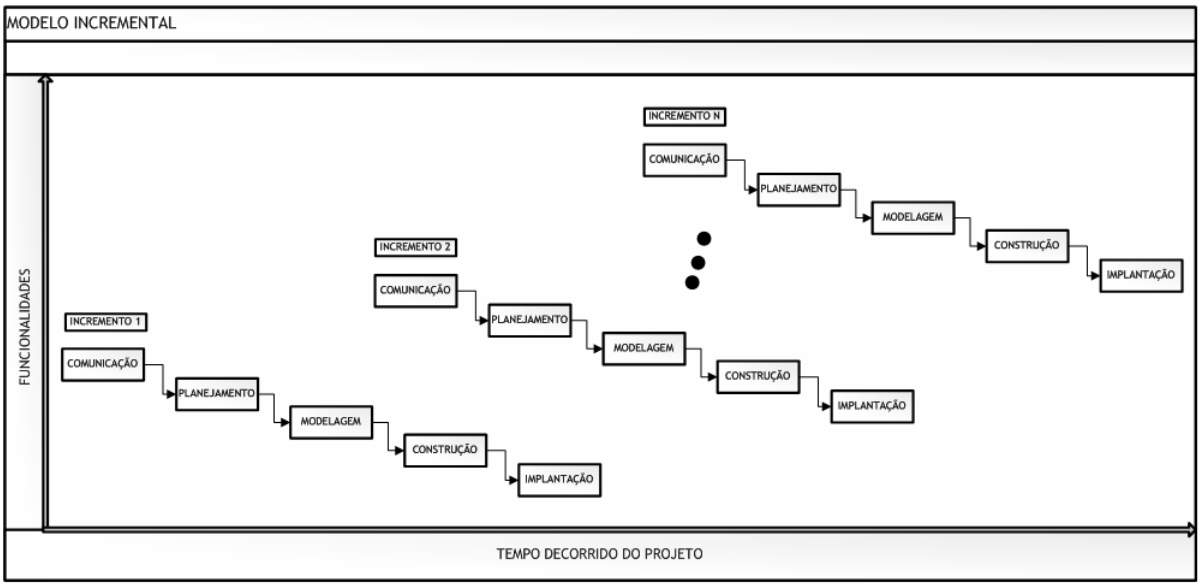


Figura 4.2: Representação gráfica do modelo incremental (Adaptado de PRESSMAN, 2006).

4.2.1.1. Comunicação

A primeira tarefa desta atividade, adotada neste processo, foi a identificação dos interessados, ou seja, a identificação das pessoas que tem interesse no resultado bem-sucedido do projeto. Após o término desta tarefa podem ser adotados caminhos diferentes, dependendo da complexidade do sistema a ser construído. Pode-se adotar um plano mais detalhado, com mais tarefas de trabalho para a coleta de requisitos, para atender sistemas mais complexos. Ou menos detalhado para atender sistemas menos complexos e menores.

Após a finalização da ação formulação, novas reuniões mais específicas para coleta de requisitos são realizadas. É então iniciada a etapa de análise de negócio e os requisitos são efetivamente coletados.

A tarefa coleta de requisitos foi iniciada com uma reunião informal com todos os interessados. Estas reuniões geralmente são confusas, necessitando de um líder para guiar a reunião, evitando que assuntos fora do escopo da reunião sejam abordados. Apesar disto, são de extrema importância para o entendimento geral do escopo do sistema, pois nela serão expostas todas as idéias, necessidades e diferentes pontos de vistas de todas as partes envolvidas. A partir desta reunião pode-se definir se as próximas reuniões serão com todos os integrantes da anterior ou somente com grupos específicos ou até mesmo individuais, dependendo da complexidade e do tamanho do sistema a ser construído. Neste segundo momento os requisitos são coletados efetivamente.

Um dos produtos de trabalho gerado na tarefa coleta de requisitos é a ata de reunião. As atas são importantes fontes de consultas dos assuntos abordados ao longo das reuniões e podem conter uma lista de pendências para as próximas reuniões. Após serem desenvolvidas, as mesmas podem ser submetidas à revisão por parte do grupo de interessados que participaram. Caso haja alguma divergência, as atas são refatoradas a fim de eliminar qualquer dúvida sobre as necessidades, objetivos e requisitos coletados. No apêndice 9.1 desta monografia é apresentado o template do produto de trabalho ata de reunião.

Após a primeira reunião, foi constatado que os interessados estavam inseguros e indecisos em relação aos requisitos. Então o processo de prototipação foi utilizado para que a partir de modelos (protótipos), os interessados pudessem avaliá-los em um esforço para identificar e solidificar requisitos de software. Na figura 4.3 é apresentado o processo de prototipação.

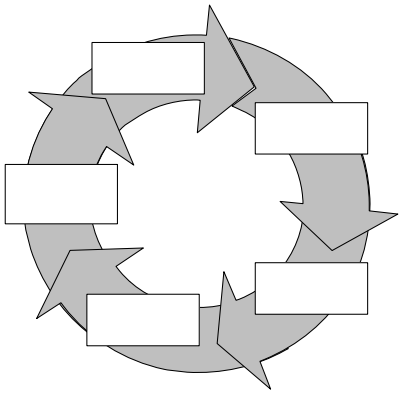


Figura 4.3: Representação gráfica do Processo de prototipação (Adaptado de PRESSMAN, 2006).

Quando a tarefa coleta de requisitos termina, é gerado o produto de trabalho documento de requisitos. No apêndice 9.2 desta monografia é apresentado o template do documento de requisitos.

Quando as ações formulação e análise de negócio não são realizadas ou são realizadas superficialmente, a probabilidade de construção de um sistema que não atenda as reais necessidades do negócio ou que não possua determinada funcionalidade requerida pelo cliente, tende a aumentar.

Após a conclusão das ações formulação e análise de negócio, o produto de trabalho documento de visão é gerado. Este documento fornece uma solução e uma visão geral do problema, engloba todos os objetivos e as funcionalidades requeridas e os requisitos de arquitetura. No apêndice 9.3 desta monografia é apresentado o template do documento de visão.

4.2.1.2. Planejamento

Nesta atividade foram utilizadas todas as ações mencionadas na seção 3.2. São elas:

- Definição do escopo do sistema que está sendo construído, através da atividade comunicação, ou seja, esta atividade é pré-requisito para a o início da atividade planejamento. Na definição do escopo são definidas as características e funcionalidades que o sistema deve ter quando estiver pronto.

- Definição da estratégia de entregas incrementais, de tal forma que estes incrementos forneçam funcionalidades para os usuários finais, através da categorização e agrupamento dos requisitos.
- Definição do arcabouço de processo de desenvolvimento, de tal forma que as atividades e suas ações devem ser selecionadas, de acordo com as características do produto a ser desenvolvido. Além do arcabouço, o modelo de ciclo de vida também é escolhido.
- Elaboração do documento de análise de risco, de tal forma que os riscos são mapeados e classificados por probabilidade de ocorrer e impacto que podem causar. Através desta classificação os riscos podem ser administrados, através de planos de atenuação de riscos. No apêndice 9.4 desta monografia é apresentado o template do produto de trabalho análise de riscos.
- Estabelecimento de um cronograma, de tal forma que as tarefas a serem realizadas em curto prazo tenham granularidade menor e granularidade maior para as tarefas dos períodos posteriores. Os cronogramas permitem avaliar o progresso do projeto e tomar ações necessárias para cumprir os prazos estipulados no mesmo, sendo utilizado como um mecanismo de monitoramento e gestão do projeto. Na figura 4.4 é apresentado o cronograma elaborado para o incremento do módulo Cadastro de Pacientes.

Este conjunto de ações juntos forma o plano de projeto, no qual mais detalhes são apresentados na seção 4.3.1, desta monografia.

	SISTEMA DE INFORMAÇAO PARA SUPORTE AO CUIDADO M	Duração	Início	Término	Predecessoras
2	**Comunicação**	**19 dias**	**Seg 01/06/09**	**Qui 25/06/09**	
3	Identificação dos Interessados	2 dias	Seg 01/06/09	Ter 02/06/09	
4	Levantamento Macro Necessidades e Objetivo	5 dias	Qua 03/06/09	Ter 09/06/09	3
5	Elicitação de Requisitos	10 dias	Qua 10/06/09	Ter 23/06/09	4
6	Etapa de Comunicação Concluída	0 dias	Qui 25/06/09	Qui 25/06/09	5
7	**Planejamento**	**13 dias**	**Sex 26/06/09**	**Seg 13/07/09**	
8	Definição das tarefas do Arcabouço	5 dias	Sex 26/06/09	Qua 01/07/09	6
9	Definição do modelo de Processo	2 dias	Qui 02/07/09	Sex 03/07/09	8
10	Análise de Riscos	3 dias	Seg 06/07/09	Qua 08/07/09	9
11	Estimativa de Entrega do Incremento	3 dias	Qui 09/07/09	Seg 13/07/09	10
12	Etapa de Planejamento Concluída	0 dias	Seg 13/07/09	Seg 13/07/09	11
13	**Modelagem**	**20 dias**	**Qui 16/07/09**	**Qua 12/08/09**	
14	Modelagem de Análise	10 dias	Qui 16/07/09	Qua 29/07/09	12
15	Modelagem de Projeto	10 dias	Qui 30/07/09	Qua 12/08/09	14
16	Etapa de Desenvolvimento Concluída	0 dias	Qua 12/08/09	Qua 12/08/09	15
17	**Construção**	**46 dias**	**Sex 14/08/09**	**Sex 16/10/09**	
18	Codificação	40 dias	Sex 14/08/09	Qua 07/10/09	16
19	Testes	5 dias	Qui 08/10/09	Qua 14/10/09	18
20	Etapa de Construção Concluída	0 dias	Sex 16/10/09	Sex 16/10/09	19
21	**Implantação**	**4 dias**	**Seg 19/10/09**	**Qui 22/10/09**	
22	Entrega	2 dias	Seg 19/10/09	Ter 20/10/09	20
23	Feedback	1 dia	Qua 21/10/09	Qua 21/10/09	22
24	Etapa de Implantação Concluída	1 dia	Qui 22/10/09	Qui 22/10/09	23

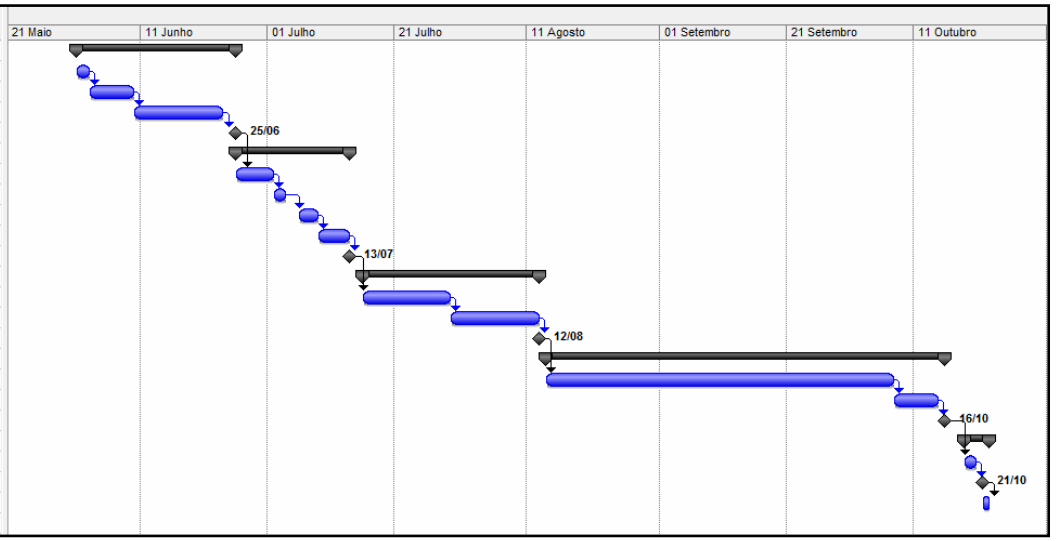


Figura 4.4: Cronograma do incremento módulo Cadastro de Pacientes.

4.2.1.3. Modelagem

A primeira ação da atividade modelagem de análise é o desenvolvimento de casos de uso. Os casos de uso descrevem a interação entre o usuário final e o sistema. O primeiro passo para descrever casos de uso é identificar o conjunto de atores que estarão envolvidos na história. Os atores representam os papéis que pessoas ou entidades externas desempenham quando o sistema está em operação. Os casos de uso são agrupados em diagramas de package, conforme a divisão do sistema em módulos. Na figura 4.5 é apresentado o diagrama de package do sistema. Na figura 4.6 é apresentado diagrama de caso de uso do package Cadastro de Paciente.

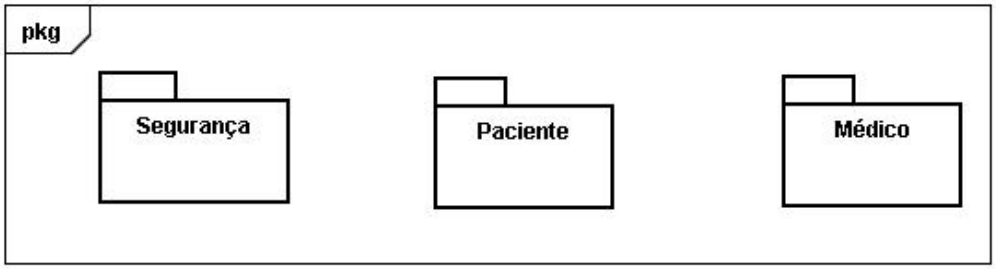


Figura 4.5: Diagrama de package do sistema.

No apêndice 9.5 desta monografia é apresentado o produto de trabalho caso de uso, referente à funcionalidade cadastrar paciente, do package Paciente.

Após o desenvolvimento dos casos de uso, pode ser necessária a criação de diagramas de atividade que complementam o caso de uso, fornecendo uma representação gráfica do fluxo de interação em um cenário específico. Os retângulos arredondados servem para identificar funções do sistema, as setas determinam o fluxo, os losangos para representar decisões e linhas sólidas horizontais para representar atividades ocorrendo em paralelo.

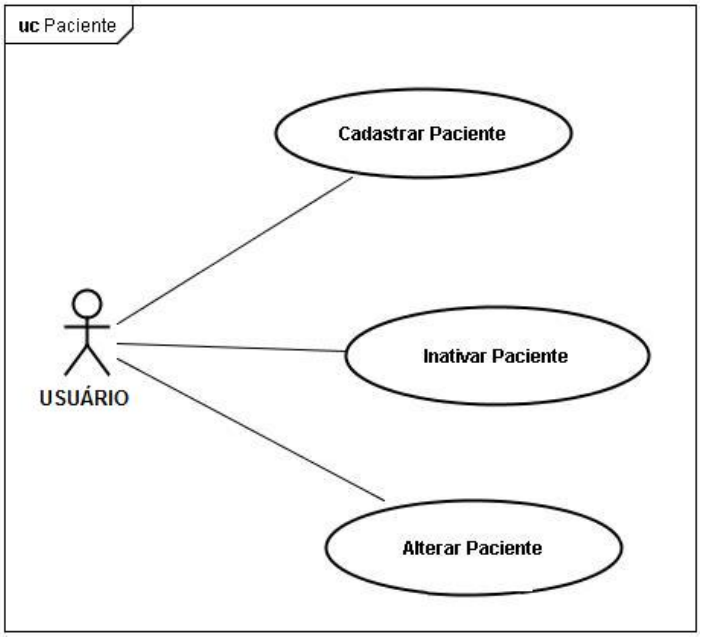


Figura 4.6: Diagrama de caso de uso do package Cadastro de Paciente

Após a criação dos diagramas de atividades, as classes de análise são identificadas. As classes são determinadas identificando substantivos. A decisão de inclusão ou não de uma classe é subjetiva e uma avaliação posterior pode causar o descarte ou a inclusão de uma classe descartada anteriormente. Nesta etapa pode-se ainda identificar atributos, possíveis operações relacionadas com as classes e relacionamento entre elas, extraídos também do caso de uso.

Na ação modelagem de projetos os seguintes subprojetos são realizados: projeto de interface, projeto navegacional e projeto arquitetural. Nesta ação também são desenvolvidos e refinados as classes de projeto.

O projeto de interface inclui uma representação do *layout* da tela, uma definição dos modos de interação e uma descrição dos mecanismos de navegação. Neste subprojeto são desenvolvidos protótipos, incluindo o *layout*, caso ainda não tenha sido desenvolvido durante a atividade comunicação. Caso o protótipo já tenha sido construído, este poderá ser revisado e refinado para conter novas funcionalidades. Ao desenvolver os protótipos são definidos os mecanismos de interação com o usuário e os mecanismos de navegação, ou seja, os links entre as interfaces. Estes mecanismos de interação podem ser:

- Menus de navegação: são organizados verticalmente ou horizontalmente e listam funcionalidades, que podem ser acessadas pelo usuário.

- Ícones gráficos: são botões que permitem ao usuário acessar determinada funcionalidade.

Os diagramas de classe do domínio do negócio ou diagrama de classes de análise, que são desenvolvidos inicialmente na etapa de modelagem de análise, neste momento são refinados para que possam conter todos os atributos e métodos necessários para implementar algum elemento do domínio do negócio do cliente. Pode surgir a necessidade de novas classes que darão suporte ao desenvolvimento. Na figura 4.7 é apresentado o diagrama de classe de projeto do módulo Cadastro de Paciente.

Além destes refinamentos os diagramas de classes persistentes ou diagrama de banco de dados são criados para dar suporte à persistência dos estados das aplicações e para recuperar estados já guardados no servidor de banco de dados.

No Projeto navegacional, são definidos os fluxos de navegações entre as funcionalidades da aplicação. Os caminhos navegacionais permitem ao usuário ter acesso ao conteúdo e aos serviços da aplicação. Na figura 4.8 é apresentado o diagrama navegacional do módulo Cadastro de Pacientes.

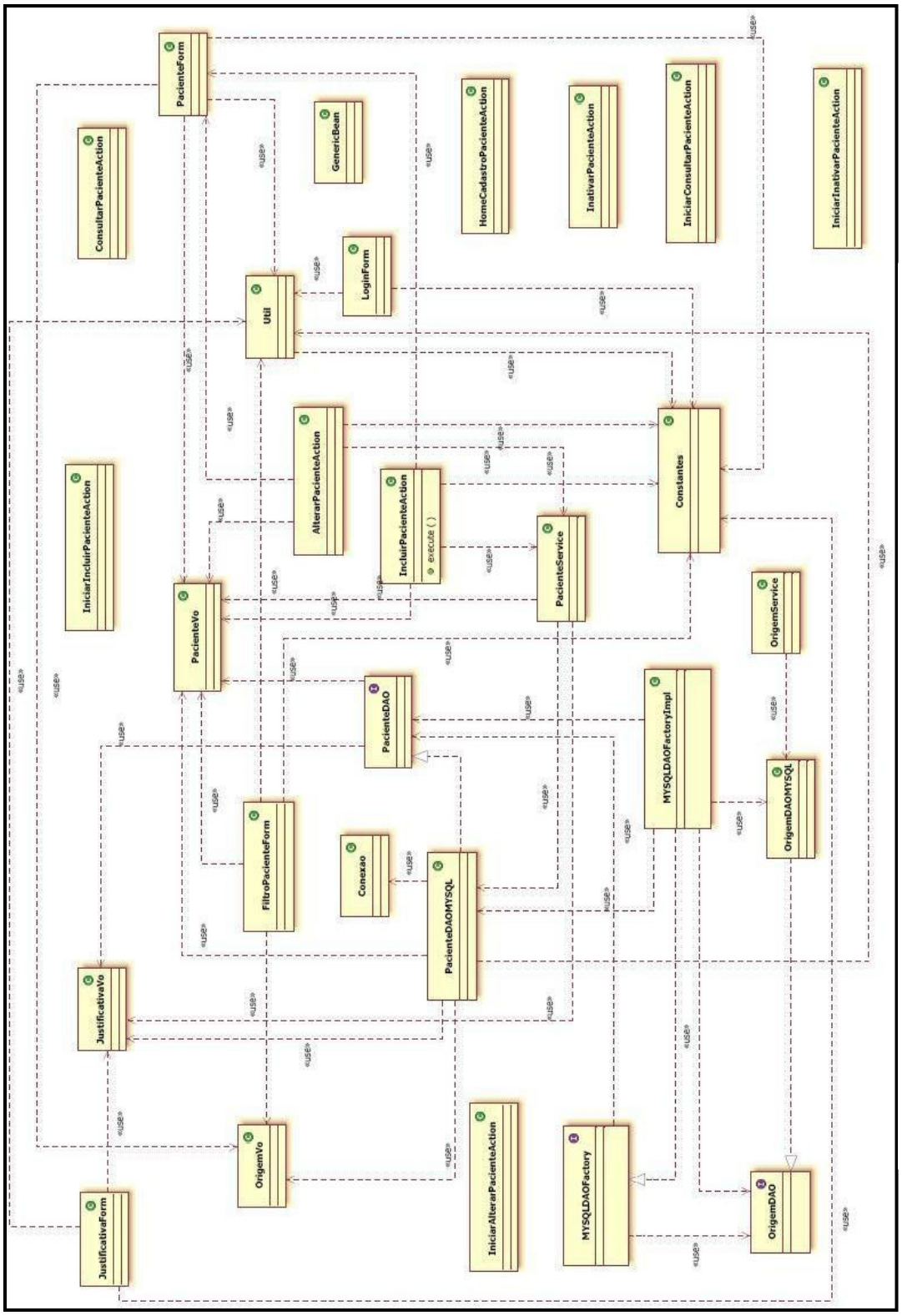


Figura 4.7: Diagrama de Classe de Projeto do módulo Cadastro de Paciente

O Projeto Arquitetural define como as aplicações devem ser construídas usando camadas nas quais diferentes preocupações são levadas em conta. Os dados devem estar separados dos conteúdos de página e esses conteúdos, por sua vez, devem ser claramente separados dos aspectos da interface, ou seja, das páginas. No capítulo 5 será detalhado o padrão de arquitetura, linguagem e framework adotados neste projeto, bem como os motivos para a utilização dos mesmos.

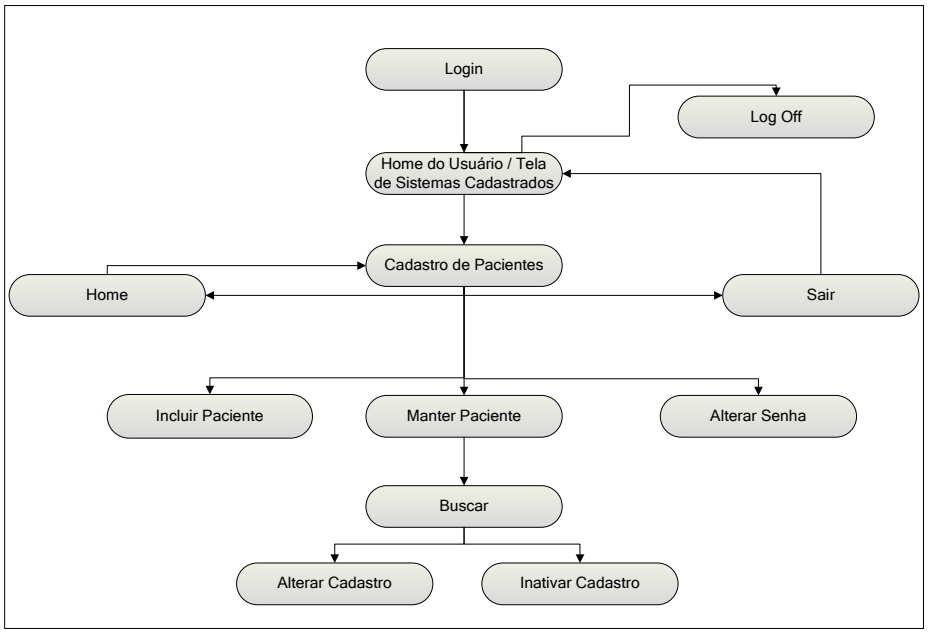


Figura 4.8: Diagrama Navegacional do incremento Cadastro de Paciente

4.2.1.4. Construção

A atividade construção compreende duas etapas: codificação e testes. A etapa de codificação foi composta basicamente por geração de código. Neste projeto são utilizados métodos de gerência de configuração e controle de versão que serão explicitados na atividade guarda-chuva Gerência de Configuração, na seção 4.3.2. Estes métodos auxiliam na gerência de modificações, tornando-as rastreáveis.

Na etapa de teste foi determinado um conjunto de testes, são eles: teste de validação, interface e navegação, integração e segurança.

O teste de validação avalia se os requisitos foram satisfeitos pelo incremento de software. Este teste é realizado através da análise dos casos de uso, verificando se existe

algum fluxo que deixou de ser tratado ou se existem erros em algum fluxo básico, alternativo ou de exceção.

No apêndice 9.6 desta monografia é apresentado o template do documento plano de testes.

O teste de interface avalia se os mecanismos de interação estão funcionando como previsto e verifica possíveis erros de grafia nas páginas do incremento a ser testado. O objetivo deste teste é descobrir erros que resultem de mecanismos de interação fracamente implementados, ambigüidades, erros relacionados a mecanismos de interface específicos como erros na execução de uma ligação de menu, no modo pelo qual os dados são inseridos nos formulários do sistema ou nos links existentes nas páginas (PRESSMAN, 2006).

O teste de integração avalia a integração entre os incrementos já testados separadamente, de forma incremental, até chegar a um módulo único composto por todos os incrementos. Logo este teste não é aplicado no primeiro incremento, somente sendo aplicado nos incrementos posteriores. Na figura 4.12 é apresentado o processo de teste de integração.

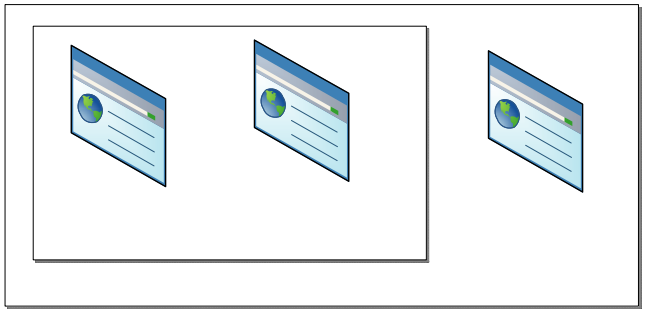


Figura 4.12: Representação gráfica do processo de teste de integração

O teste de segurança avalia brechas na aplicação, que são os pontos fracos do sistema e servem como auxílio para usuários mal- intencionados invadirem ou burlarem o sistema. A intenção deste teste é mostrar que uma quebra de segurança é possível. Os testes de segurança realizados neste projeto foram limitados a parte do cliente, ou seja, somente possíveis vulnerabilidades do lado do cliente foram exploradas.

4.2.1.5. Implantação

Na atividade de implantação, do módulo cadastro de paciente, foram desenvolvidos os manuais que darão suporte aos usuários. Estes manuais foram utilizados também como guias para o teste de aceitação do cliente. Após a realização do teste, o cliente aceitou a entrega do módulo e foi colhido o feedback sobre o módulo entregue.

4.3. Atividades guarda-chuvas utilizadas

As atividades guarda-chuvas utilizadas neste projeto foram:

4.3.1. Gestão de projeto

A gestão de projetos de aplicações para a *Web* requer uma abordagem ágil, porém é necessário que haja algum planejamento. A gestão de projetos envolve o planejamento, a monitoração e o controle de equipes, processos e suas respectivas atividades através de um plano de projeto e de mecanismos de acompanhamento deste plano. Estes mecanismos podem ser tanto reuniões de acompanhamento com as equipes que estão envolvidas no projeto quanto o acompanhamento dos marcos intermediários das atividades de arcabouço de um incremento, realizado diretamente pelo líder de projeto.

O plano de projeto define o processo e as atividades a serem executadas, o membro de uma equipe que irá executar determinada tarefa e mecanismos para avaliação de riscos e para o controle de modificações. O plano de projeto pode ser criado assim que escopo do sistema a ser desenvolvido tiver sido delimitado, na atividade de planejamento do arcabouço do processo de desenvolvimento.

A gerência de equipes é necessária para que possa garantir que cada membro de uma equipe esteja realizando as atividades, definidas no plano de projeto, efetivamente.

O líder de projeto deve definir quais atividades do arcabouço serão utilizadas, levando em consideração as características das equipes, o cliente que solicitou o produto e as características do sistema a ser desenvolvido. Os diferentes projetos de software possuem características e demandas únicas, fazendo com que determinadas atividades do arcabouço sejam realizadas intensamente e outras não sejam necessárias.

Nas reuniões de acompanhamento participam as equipes envolvidas e um líder de projeto que irá verificar se o cronograma está sendo seguido, ou seja, se um dado incremento poderá ser entregue no prazo estimado anteriormente no início do projeto. Caso o líder constate que a data de entrega de algum entregável possa desrespeitar o prazo definido no início do projeto, ele terá que tomar as ações necessárias para manter o cronograma ou, em último caso, modificar os prazos para as demais tarefas.

Outra forma de gestão de projetos é através de marcos que estão estabelecidos no final de cada atividade de arcabouço. Durante o desenvolvimento do incremento o cronograma é avaliado constantemente pelo líder de projeto, para verificar se o tempo estimado para cada atividade de arcabouço poderá ser cumprido. Caso haja necessidade de modificação nas datas, o cronograma terá que ser refatorado.

4.3.2. Gerência de Configuração

Segundo PRESSMAN (2006), a gerência de configuração é definida com a arte de identificar, organizar e controlar as modificações no software que está sendo construído por uma equipe, com o objetivo de maximizar a produtividade e minimizar os erros. Esta atividade é aplicada ao longo de todo o desenvolvimento do software.

Em projetos em que não são empregados os métodos de gerência de configuração, a probabilidade de ocorrer erros aumenta exponencialmente já que vários programadores concorrem pela utilização do código fonte (repositório) e caso não sejam criadas formas de comunicar que uma alteração tenha sido feita por um programador específico, os outros não saberão que aquele objeto do repositório foi alterado, podendo haver sobreposição de alterações. Esta situação exposta causa alguns efeitos indesejáveis como a perda de código fonte, impossibilidade de determinar o que aconteceu com um programa ou parte dele e a impossibilidade de determinar quem, porque e quando foram efetuadas as modificações nos objetos do repositório.

Neste projeto foram utilizados métodos de gerência de configuração. Nos itens de configuração como os produtos de trabalho, foram aplicados métodos de controle de versão (*releases*). No início de cada documento foram definidos campos que identificam a versão, a data, o motivo e o responsável pela modificação. Desta forma as modificações nos produtos de trabalho, tornam-se rastreáveis.

No caso dos códigos fontes foi utilizado o *Subversion* para gerenciar. O *Subversion* auxilia no controle de versões, controlando o acesso ao repositório e mantendo as alterações através de um controle de versões. A arquitetura proposta pelo *Subversion* é semelhante a um cliente-servidor, no servidor temos o repositório dos arquivos que são armazenados em um banco de dados e diversos clientes que acessa o repositório (RAPCINSKI, 2009).

Os documentos são recuperados do repositório e armazenados localmente, após os documentos serem alterados pode-se submetê-los ao servidor, alterando o número de revisão do repositório. O número de revisão é seqüencial iniciando em um e incrementado a cada alteração dos documentos. Cada documento tem armazenado no repositório sua última situação e a situação em cada uma das revisões, desta forma podemos reverter qualquer alteração e comparar o documento com uma revisão específica (RAPCINSKI, 2009).

4.3.3. Gerência de Riscos

Nesta atividade foram gerenciados os riscos levantados na atividade planejamento, através da execução do plano de atenuação, conforme o apêndice 9.4. Após a execução do plano de atenuação os riscos são continuamente monitorados a fim de verificar a reincidência do mesmo.

Capítulo 5

Tecnologias Utilizadas

Neste capítulo são apresentadas as tecnologias utilizadas neste projeto, bem como as justificativas para o uso destas tecnologias.

5.1. PHP, ASP e JSP

Dentre as diversas tecnologias disponíveis atualmente para o desenvolvimento de aplicações *Web*, destacam-se o PHP, ASP e JSP. Estas tecnologias possibilitam, de forma simplificada, o tratamento das requisições de página de um servidor *Web*, permitindo que se adicione conteúdo dinâmico à resposta enviada aos clientes (navegadores) através da utilização de scripts (aplicações).

Estas tecnologias têm em comum a implementação dos recursos mínimos esperados neste porte de solução, como controle de sessões, módulos para acesso a bancos de dados variados, tratamento de formulários e do fluxo requisição/retorno de páginas, porém, existem diferenças significativas entre elas que podem levar projetistas de sistemas a se decidirem por uma ou outra.

O PHP é uma linguagem desenvolvida a partir de um projeto pessoal de Rasmus Ledorf, inicialmente como uma pequena linguagem de *script* para adicionar alguma lógica ao processamento de formulários de seu *site*. Os seguintes fatores que colaboraram para sua disseminação foram: uma curva de aprendizado pouco íngreme, um excelente desempenho e o

acesso nativo a um dos bancos de dados mais populares em sites *Web*, o *Mysql* (PHP, 2009; NARAMORE *et al.*, 2005).

Uma desvantagem é a fraquíssima implementação da orientação a objetos, isto torna inaplicáveis boa parte dos recursos atuais de projeto e desenvolvimento, dificultando o reaproveitamento de código e a manutenção das aplicações.

Outra desvantagem, bem mais evidente, é o fato da lógica misturar-se à camada de apresentação, o que torna confusa a leitura dos módulos de um sistema. Os *scripts* PHP são executados em meio ao código HTML da apresentação, o que torna qualquer trabalho de manutenção ou reaproveitamento, tanto pelos desenvolvedores como pelos *web designers*, extremamente delicado (EVJEN, 2006).

O ASP é a solução *Microsoft* para proporcionar aos usuários de seu ambiente a geração de páginas *Web* de conteúdo dinâmico. O ASP baseia-se na linguagem *Basic* (no caso, o *Visual Basic*) para a construção dos *scripts*. O acesso a bancos de dados é, também, bastante variado, graças ao uso do *Open Data Base Connectivity* (ODBC) e do *ActiveX Data Objects* (ADO), ambas as tecnologias *Microsoft* aplicadas e suportadas globalmente. Utilizando-se outros produtos do portfólio da *Microsoft*, podem-se construir aplicações realmente complexas e integradas a partir do ASP (MICROSOFT, 2009 ; EVJEN, 2006).

Não se pode classificar o aprendizado de ASP como difícil, embora o contato com a linguagem *Basic* ajude bastante neste quesito. A abundante oferta de mão-de-obra formada neste ambiente facilita em muito a decisão pelo ASP.

Assim como o PHP, o ASP sofre com o acoplamento entre a camada de apresentação e a camada lógica, o que a princípio torna fácil o seu aprendizado e acaba por tornar-se um empecilho em projetos de maior porte, dificultando as tarefas de desenvolvedores e *web designers*. As características de orientação a objetos são, como no PHP, tímidas perto do JSP.

O JSP foi a resposta da Sun, criadora da tecnologia Java, às necessidades dos desenvolvedores de aplicações *Web*. Diferentemente do ASP, e sem chegar a concorrer diretamente com o PHP, o JSP nasce com atributos suficientes para a missão a que se propõe: orientação a objetos nativa e forte, desacoplamento entre código e apresentação, e o que se chama no meio servidor de "robustez industrial", a solidez procurada para aplicações de missão crítica, característica esta herdada do ambiente Java (SUN, 2009; BASHAM, 2008).

A curva de aprendizado do JSP não pode ser considerada suave, como a apresentada pelo PHP ou pelo ASP. Apenas os desenvolvedores bastante habituados com Java, e que tenham a orientação a objetos como um paradigma natural em suas cabeças, não sentirão o

impacto da troca de um ambiente procedural para o desenvolvimento de aplicações em camadas.

Na tabela 5.1 é apresentado um comparativo entre as linguagens que podem ser utilizadas na camada *View*.

ASPECTO	**JSP**	**PHP**	**ASP**
Curva de Aprendizado	Aguda	Suave	Suave
Código Fonte	Aberto	Aberto	Proprietário
Linguagem Nativa	Própria	Semelhança com C ANSI	Visual Basic e *javascript*
Drives para banco de Dados	JDBC e ODBC nativos	Modulares ou ODBC	ODBC/ADO
Porte típico dos projetos	Médios a Grandes	Pequenos a Médios	Pequenos a Médios
Ambiente típico	Java	Unix	Microsoft

Tabela 5.1: Comparativo entre as linguagens.

Assim, o JSP apresenta-se como uma solução profissional, e pode ser considerado o futuro das aplicações *Web* devido a desacoplar com facilidade a apresentação da lógica, permitir a compilação em tempo real das aplicações (uma vez executado o *script*, uma versão compilada se apresenta às requisições seguintes) e integrar-se à perfeição no mundo Java. O JSP possibilita acesso a diversos bancos de dados (através de JDBC), é executado em inúmeros sistemas operacionais e vem se tornando a opção de fato para o milionário mercado de servidores de aplicação, além de possuir alternativas gratuitas ou *Open Source* como o projeto *Jakarta/Tomcat* (BASHAM, 2008).

A penetração do JSP só não é maior devido ao conjunto de conhecimentos exigidos para que se tire proveito de seus melhores atributos: o entendimento do paradigma da orientação a objetos e a fantástica extensão de sua linguagem nativa, o Java. Assim, o JSP é uma aposta sem margem de erro, e não encontra concorrentes quando se pensa em aplicações críticas em ambientes corporativos.

5.2. Padrão de Arquitetura Adotado

Neste projeto o padrão de arquitetura adotado foi o padrão MVC adaptado para *Web*, como citado na seção 3.4. Este padrão é o mais indicado para aplicações *Web* por possuir as seguintes características: separação clara entre a camada *Model* e a camada *View*, manutenção facilitada, reaproveitamento de código e melhor visibilidade da camada *Model*. Apesar de ser o padrão mais indicado, este apresenta algumas desvantagens como: o seu uso pode criar uma complexidade desnecessária, em sistemas pouco complexos e exige disciplina do programador. A seguir serão apresentadas as camadas presentes neste padrão de arquitetura (HUSTED, 2004).

5.2.1. Camada *Model*

Segundo SOUZA (2004), a camada *Model* representa o estado da aplicação. É responsável por fazer a interface da aplicação com a fonte dos dados, muito freqüentemente um banco de dados. Quando existe a necessidade de se guardar o estado da aplicação, é através dessa camada que as informações manipuladas pelo sistema podem ser armazenadas na base de dados.

Nesta camada foi utilizado o padrão de projeto *Data Access Object* (DAO) que consiste na manutenção dos dados (incluir, alterar, excluir), provendo métodos de busca de dados persistidos e listando dados de várias formas.

Segundo SGUILLARO e LOPES (2009), as vantagens da utilização deste padrão são:

- Inteligência, pois descobre o melhor meio de persistir os dados.
- Tem funcionalidades mínimas para a manutenção dos dados.
- Pode adicionar funcionalidades extras sem modificar as características.
- Maior flexibilidade para o padrão MVC, pois a camada *Model* apenas cria a entidade e chama os métodos de manutenção ou as funções extras sem se preocupar com seu funcionamento.
- Divisão de responsabilidade, pois se projeta as entidades separadamente das outras camadas.

Este padrão pode trazer algumas desvantagens em sua utilização, como por exemplo, um objeto entidade se torna mais complexo, não podendo ser usado em qualquer caso.

5.2.2. Camada *View*

A camada *View* é a camada de apresentação do sistema ao usuário, isto é, a interface que proporcionará a entrada de dados e a visualização de respostas geradas. Nas aplicações voltadas para a *web*, normalmente a camada *View* é construída através de páginas HTML. A utilização de HTML somente, não permite que o conteúdo das páginas seja dinâmico, fazendo-se necessário o uso de uma linguagem que permita tal funcionalidade. Para a geração de conteúdo dinâmico, neste projeto, foi utilizada a linguagem JSP como descrito na seção 5.1, desta monografia.

5.2.3. Camada *Controller*

A camada *Controller* funciona como intermediária entre a camada *View* e a camada *Model*. Sua função é controlar e coordenar o envio de requisições feitas entre a *View* e o *Model*. O *Controller* define o comportamento da aplicação, interpreta as interações (cliques, seleções de menus, etc) feitas por usuários e com base nestes requerimentos o controlador comunica-se com o *Model* que seleciona a *View* e atualiza-a para o usuário, ou seja, o *Controller* controla e mapeia as ações (BAPTISTELLA, 2009).

5.3. Struts *Framework*

O *Struts Framework* é um projeto *open source* mantido pela *Apache Software Foundation*. É uma implementação do padrão de arquitetura MVC para aplicações Java para *Web*. Na figura 5.1 é apresentado o modo como o *Struts* implementa este padrão de arquitetura.

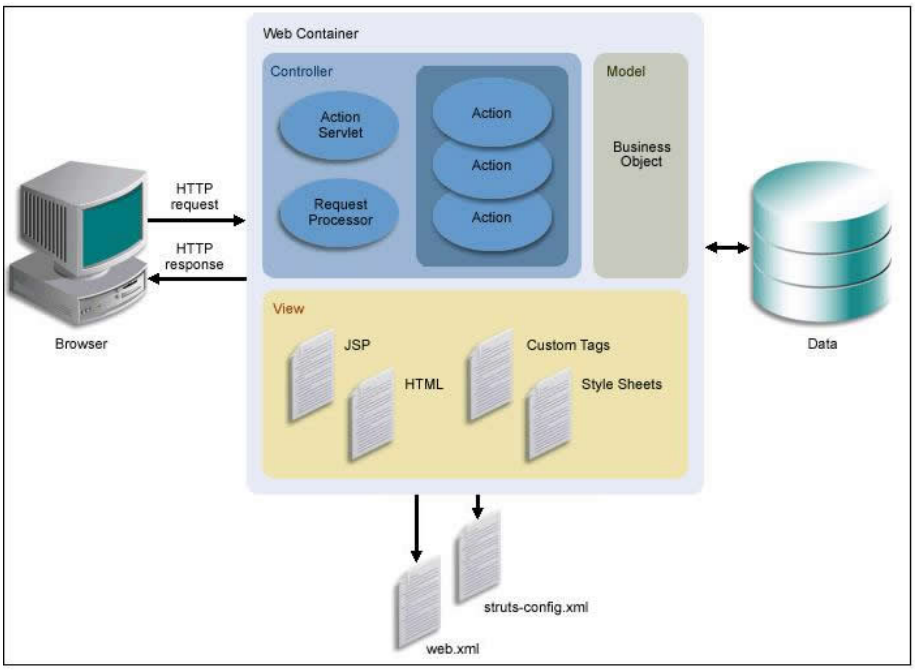


Figura 5.1: Representação gráfica da arquitetura Struts implementando MVC (Adaptado de HUSTED, 2004).

Segundo SOUZA (2009), o *Struts framework* foi desenvolvido *por Craig McClanahan* em Maio de 2000, e desde então vem sendo melhorado pela comunidade *open source*. Foi desenvolvido com o objetivo de fornecer um *framework* para facilitar o desenvolvimento de aplicações Java para *Web*. A seguir são apresentados alguns fatores motivacionais para a utilização deste *framework*:

- Tornou-se um padrão de mercado.
- Garantia de que alguém (*Apache Group*) irá manter o *framework*, isto é, suporte para correção de erros e novos *releases*.
- Integração com a maioria dos ambientes de desenvolvimento do mercado.
- Separa a camada de negócio da camada de apresentação.
- Incorpora diversos *Designs Patterns*.
- Criação de aplicações padronizadas, facilitando a manutenção.
- Criação de Aplicações Internacionalizadas.
- Possibilidade de gerar a saída de acordo com o dispositivo usado (HTML, SHTML, WML etc).
- Aumento da produtividade.

Segundo HUSTED (2004), apesar dos benefícios, o *Struts framework* possui algumas desvantagens, tais como:

- Nenhum modelo de eventos, ou seja, o *Struts* está intimamente ligado ao modelo de solicitação-resposta usado pelo HTTP.
- Apenas um *ActionServlet* pode ser usado em uma única aplicação. Esse limite pode levar a conflitos de configuração.
- Para trabalhar com o *Struts*, o desenvolvedor precisa compreender várias classes especiais e como elas interagem.
- O acesso para um modelo de dados permanente é deixado inteiramente como um exercício para o desenvolvedor.

De acordo com SOUZA (2009), o *Controller* já vem implementado no *Struts*, embora seja possível estendê-lo a fim de adicionar funcionalidades. O fluxo da aplicação é programado em um arquivo *Extensible Markup Language* (XML) através das ações dos usuários da aplicação, que serão executadas. As ações são classes nativas implementadas pelo *framework* seguindo o padrão MVC, sendo necessário estendê-las a fim de adicionar novas funcionalidades desejadas. A geração da interface é feita através de *custom tags*, também já implementadas pelo Struts, evitando assim o uso de *Scriptlets*, ou seja, inclusão de código Java na página JSP. Desta forma as páginas JSP ficam mais simples, auxiliando na manutenibilidade.

A seguir é apresentado o fluxo de navegação através do *Struts Framework*, desde a interação entre o usuário e a aplicação até a resposta desta interação. Na figura 5.2 é apresentado o fluxo de navegação.

1. O usuário faz uma solicitação através de uma *url* no navegador. Ex: http://localhost:7070/Seguranca/iniciarLogin.do. O “.do” do final da *url* será usado para invocar o *Servlet controller* do *Struts*.
2. Se for a primeira solicitação que o container recebeu para esta aplicação, ele irá invocar o método *init*() da *ActionServlet* (*controller* do Struts) e irá carregar as configurações do arquivo struts-config.xml em estruturas de dados na memória. Este fluxo só será executado uma única vez, pois nas solicitações subseqüentes, o *Servlet* consulta estas estruturas em memória para decidir o caminho a ser seguido.

3. Baseado no fluxo definido no arquivo struts-config.xml, e que neste momento já se encontra carregado em estruturas na memória, o *ActionServlet* identificará qual o *ActionForm* (classe para a validação dos dados) irá invocar. A classe *ActionForm* através do método *Validate* irá verificar a integridade dos dados que foram recebidos na solicitação originária do navegador.
4. O controle da aplicação é retomado pelo *ActionServlet*, que verifica o resultado do *ActionForm*.
    a. Se faltar alguma coisa (campo não preenchido, valor inválido etc), o usuário recebe um formulário HTML (geralmente o mesmo que fez a solicitação), informando o motivo do não atendimento da solicitação, para que o usuário possa preencher corretamente os dados para fazer uma nova solicitação.
    b. Se não faltou nenhuma informação, ou seja, todos os dados foram enviados corretamente, o *controller* passa para o próximo passo (*Action*).
5. O *ActionServlet*, baseado no fluxo da aplicação (estruturas já carregadas em memória) invoca uma classe *Action*. Esta classe passará pelo método *execute* que irá delegar a requisição para a camada de negócio (*Model*).
6. A camada de negócio irá executar algum processo (geralmente popular um *bean*, ou uma coleção). O resultado da execução deste processo (objetos já populados) será usado na camada de apresentação para exibir os dados.
7. Quando o controle do fluxo da aplicação voltar ao *Action* que invocou o processo da camada de negócio, será analisado o resultado, e definido qual o mapa adotado para o fluxo da aplicação. Neste ponto, os objetos que foram populados na camada de negócio serão anexados como atributos na seção do usuário.
8. Baseado no mapeamento feito pela *Action*, o *Controller* faz um *forward* para o JSP para apresentar os dados.
9. Na camada de apresentação (*View*), os objetos que foram anexados como atributos da sessão do usuário serão consultados para montar o HTML para o navegador.
10. Chega o HTML da resposta requisitada pelo usuário.

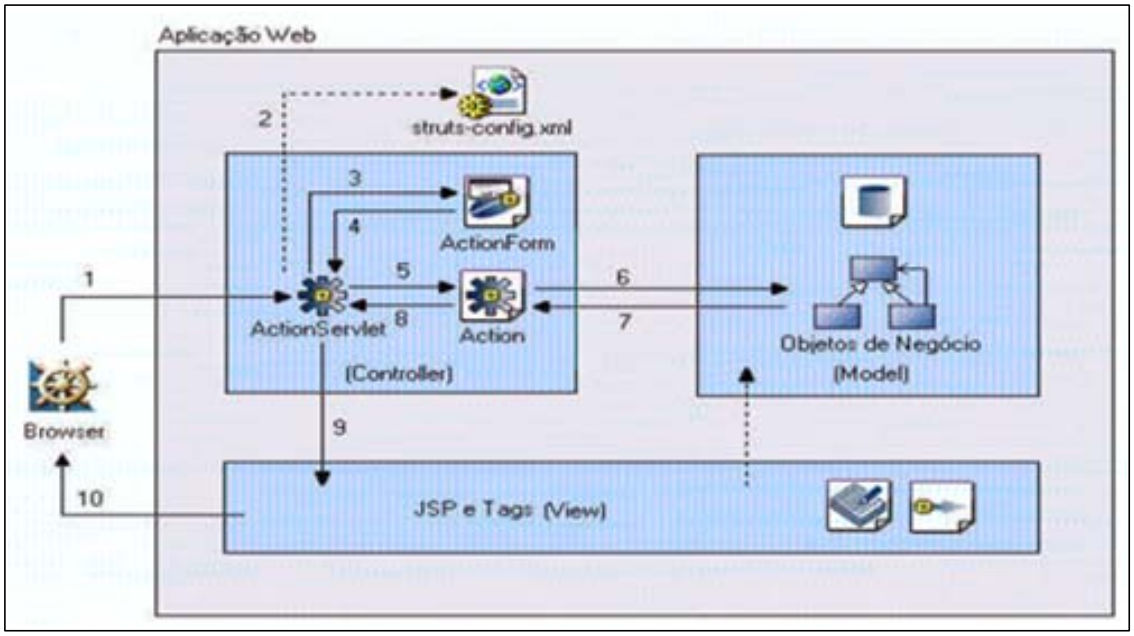


Figura 5.2: Representação gráfica do fluxo de navegação do Struts *Framework* (Adaptado de SOUZA, 2009).

Capítulo 6

O Protótipo Desenvolvido

O protótipo desenvolvido é constituído pelo núcleo e pelo módulo médico. O núcleo é composto de dois módulos: o módulo de cadastro de paciente e o módulo de cadastro de usuários. Estes módulos foram denominados de núcleo devido a serem os módulos compartilhados por todos os outros módulos. Na figura 6.1 é apresentado o esquema que representa todos os módulos que darão suporte a equipe.

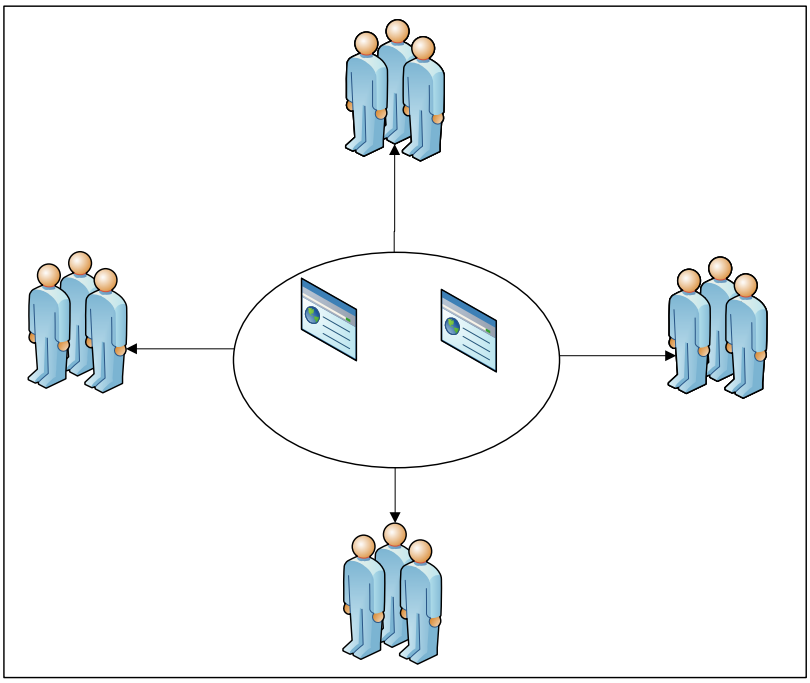


Figura 6.1: Esquema representativo de todos os módulos do sistema.

Na figura 6.2 é apresentada a tela principal do sistema, contendo os módulos desenvolvidos.

O módulo de cadastro de pacientes dará suporte à manutenção do cadastro de paciente, ou seja, inclusão, alteração e inativação de pacientes. Para algumas funcionalidades, como a inativação do paciente, será necessário informar uma justificativa para realizar uma determinada ação. Desta forma, estas ações poderão ser rastreáveis, ou seja, a qualquer

momento as informações relativas a esta ação, como o usuário responsável pela ação, data e justificativa, poderão ser visualizadas.

Figura 6.2: Tela do sistema contendo os módulos.

O módulo de segurança implementa a manutenção dos usuários do sistema, controlando a inclusão, alteração e inativação. Além disto, este módulo é capaz de definir perfis de acessos a determinados usuários. Estes perfis poderão ser criados, conforme a necessidade dos usuários. Na figura 6.3 é apresentado o módulo de segurança.

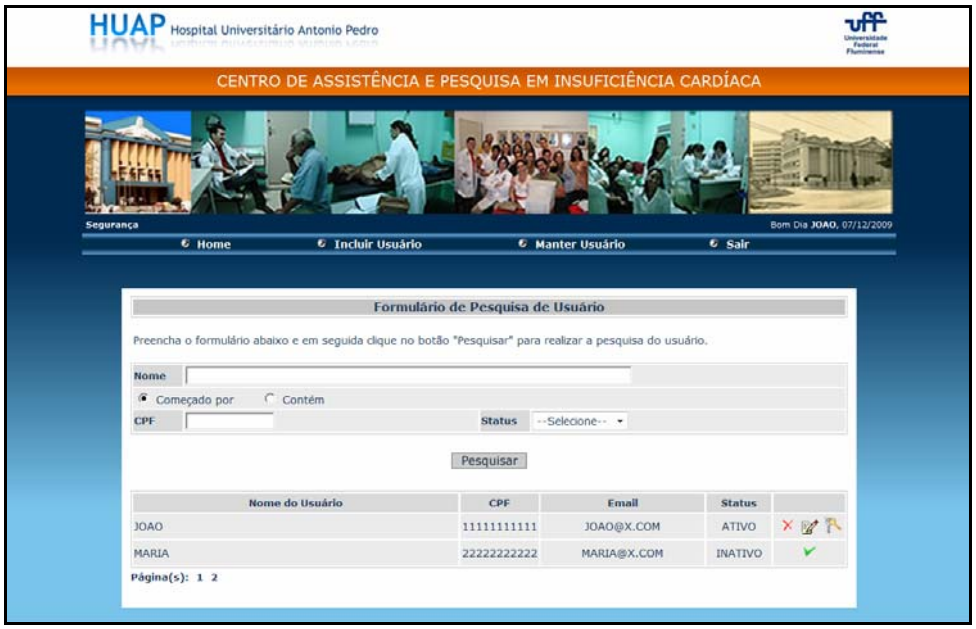


Figura 6.3: Tela do módulo de segurança.

O módulo médico dará suporte ao preenchimento dos formulários relacionados aos atendimentos dos pacientes. Este sistema possibilita a visualização do histórico de um determinado paciente, assim como a geração de relatórios destas informações, permitindo que os avanços dos quadros dos pacientes possam ser acompanhados pela equipe médica.

Todas as funcionalidades do sistema estão protegidas pelo módulo de segurança que impedirá o acesso de usuários sem permissão a determinadas funcionalidades.

Como a pesquisa realizada pela equipe multidisciplinar já possuía uma base de dados em Excel, foi necessário a geração de scripts para importar os dados para a base de dados do sistema. Esta importação automática garante que os dados legados poderão ser enviados para o sistema de forma íntegra, evitando os possíveis erros do processo manual, de forma mais ágil e menos trabalhosa.

Capítulo 7

Conclusão

Conforme apresentado na introdução deste projeto final de curso, o produto principal deste trabalho é um sistema que atende a equipe multidisciplinar do hospital universitário Antonio Pedro, auxiliando na troca de informações entre os membros desta equipe.

Tivemos dificuldades no levantamento de requisitos devido à equipe não ter, em um primeiro momento, as reais necessidades para este sistema. Foi necessário mapear o processo interno de atendimento aos pacientes, para que as necessidades e funcionalidades pudessem ser definidas.

Neste projeto foi desenvolvido o núcleo do sistema, formado pelos módulos de cadastro de paciente e segurança. Além disso, foi iniciado o desenvolvimento do módulo médico que dará suporte à equipe médica. Existe a necessidade de ampliar o sistema para atender as demandas das demais áreas não contempladas neste primeiro estágio. Estas áreas são: Fisioterapia, Farmácia e Enfermagem.

Após o desenvolvimento destes módulos o sistema passará a funcionar de forma completa, atendendo a toda a equipe multidisciplinar, gerando relatórios, gráficos, exportando os dados para Excel.

Para a construção destes novos módulos, toda a arquitetura definida na construção do núcleo e do módulo médico, poderá ser utilizada, facilitando e agilizando a construção destes novos módulos.

Capítulo 8

Referências Bibliográficas

1. ANGELONI, Maria T.: Elementos intervenientes na tomada de decisão, 2003.

2. BASHAM, Bryan; SIERRA, Kathy; BATES, Bert: Head First Servlets and JSP, Segunda Edição, 2008.

3. BAPTISTELLA, Adriano: Abordando a Arquitetura MVC e Design Patterns: Observer, Composite, Strategy. Disponível em: http://www.webartigos.com/articles/20878/1/abordando-a-arquitetura-mvc-e-design-patterns. Acesso em: 15 de Nov de 2009.

4. BENITO, Gladys A. V.: Concepção de um sistema de informação de apoio à supervisão da assistência em enfermagem hospitalar: uma abordagem da ergonomia cognitiva, Florianópolis (SC): Universidade Federal de Santa Catarina, Programa de Pós- Graduação em Engenharia de Produção; 2001.

5. DANIEL, M. Brandon.: Software engineering for modern Web applications: methodologies and technologies, 2008.

6. DAVENPORT, Thomas H.: Ecologia da Informação, Editora Futura, São Paulo, 2000.

7. EVJEN, Bill; HANSELMAN, Scott; MUHAMMAD, Farhan; SIVAKUMAR, Srinivasa; RADER, Devin: Professional ASP.NET 2.0, 2006

8. FERREIRA, S. M. G.: Sistema de Informação em saúde. Ministério da Saúde Gestão Municipal de Saúde: textos básicos, Rio de Janeiro, 2001.

9. HUSTED, Ted.: Struts em ação, 1ª edição, 2004.

10. MESQUITA, Evandro. T.; BOCCHI, Edimar A.; VILAS-BOAS, Fábio; BATLOUNI, Michel: Revisão das II Diretrizes da Sociedade Brasileira de Cardiologia para o Diagnóstico e Tratamento da Insuficiência Cardíaca, Sociedade Brasileira de Cardiologia, 2002.

11. Microsoft: Active Server Pages. Disponível em: http://msdn.microsoft.com/en-us/library/aa286483.aspx. Acesso em: 5 de Nov de 2009.

12. MURTA, Leonardo G. P.: Processo Unificado. Disponível em: http://www.ic.uff.br/~leomurta/courses/2009.1/es1/aula3.pdf. Acesso em 15 de Nov de 2009.

13. NARAMORE, Elizabeth; GERNER, Jason; SCOUANERC, Yann L.; STOLZ, Jeremy; GLASS, Michael K.: Beginning PHP5, Apache, and MySQL Web Development, 2005.

14. PHP: Linguagem de programação. Disponível em: http://php.net/index.php. Acesso em : 10 de Nov de 2009.

15. POMPILHO, S.: Análise Essencial – Guia prático de Análise de Sistemas, Infobook, Rio de Janeiro, 1995.

16. PRESSMAN, Roger S.: Engenharia de Software, 6ª edição, 2006.

17. RAPCINSKI, Heitor.: Utilizando subversion como controle de versão. Disponível em: http://www.guj.com.br. Acesso em: 10 de Nov de 2009.

18. SGUILLARO, Daniel; LOPES, Paulo V.: Padrão de projeto de software. Disponível em: http://www.ime.usp.br/~kon/MAC5715/PLoP/2006/refact. Acesso: 05 de Nov de 2009.

19. SILVA, Christiano P.; BACAL, Fernando; PIRES, Philippe V.; MANGINI, Sandrigo; ISSA, Victor S.; MOREIRA, Silvia F. A.; CHIZZOLA, Paulo R.; SOUZA, Germano E. C.; GUIMARÃES, Guilherme V.; BOCCHI, Edimar A.: Perfil do Tratamento da Insuficiência Cardíaca na Era dos Betabloqueadores, Sociedade Brasileira de Cardiologia, 2006.

20. Sociedade Brasileira de Cardiologia: III Diretriz Brasileira de Insuficiência Cardíaca Crônica, 2009.

21. SOUZA, Marcos V. B.: Estudo Comparativo entre Frameworks Java para Construção de Aplicações Web, 2004.

22. SOUZA, Welington B.: Struts Framework. Disponível em: http://www.j2eebrasil.com.br. Acesso em: 20 de Nov de 2009.

23. SUN: JSP – JavaServer Pages Technology. Disponível em: http://java.sun.com/products/jsp. Acesso em: 10 de Nov de 2009.

24. TANI, Fábio S.: A busca de generalidade, flexibilidade e extensibilidade no processo de desenvolvimento de frameworks orientados a objetos. Disponível em: http://projetos.inf.ufsc.br/arquivos_projetos/projeto_11/Resumo_II.pdf. Acesso em: 15 de Nov de 2009.

25. TEMPLE, André; MELO, Rodrigo F.; CALEGARI, Danival T.; SCHIEZARO, Maurício: JSP, Servlets e J2EE. Disponível em: http://www.inf.ufsc.br/~bosco/downloads/livro-jsp-servlets-j2ee.pdf. Acesso em: 20 de Nov de 2009.

26. TONSIG, Sergio L.: Dos elementos de dados ao sistema de informação, 2000.

27. VALLATI, Renato V.; DOMINGOS, Talitta S. A. F.: Desenvolvimento de aplicações para Web baseado no RUP, 2003

28. WELLS, Don: The rules of extreme programming. Disponível em: http://www.extremeprogramming.org/rules.html. Acesso em: 20 de Nov de 2009

Capítulo 9

Apêndice

9.1. Ata de Reunião

UFF- Universidade Federal Fluminense
Departamento de Ciência da Computação

<Nome do Sistema>

<Nome do Módulo>

Ata Reunião

Controle de Revisão			
Release	**Data**	**Motivo**	**Responsável**
<x>	<dd/mm/aaaa>	<Descrição do motivo>	<Nome do responsável pela alteração do documento>

UFF- Universidade Federal Fluminense Departamento de Ciência da Computação	Ata de Reunião Release <x>

1. Participantes

<Listagem contendo os participantes e suas respectivas da reunião.>

2. Assuntos Tratados

<Listagem contendo os assuntos tratados na reunião.>

3. Pendências

<Listagem contendo as pendências para as próximas reuniões, separadas pelas áreas dos participantes.>

Elaborado por: <Nome do Responsável>	Impresso em: <Data e hora>	Pág: 2/2
<Caminho de armazenamento do documento>		

9.2. Documento de Requisitos

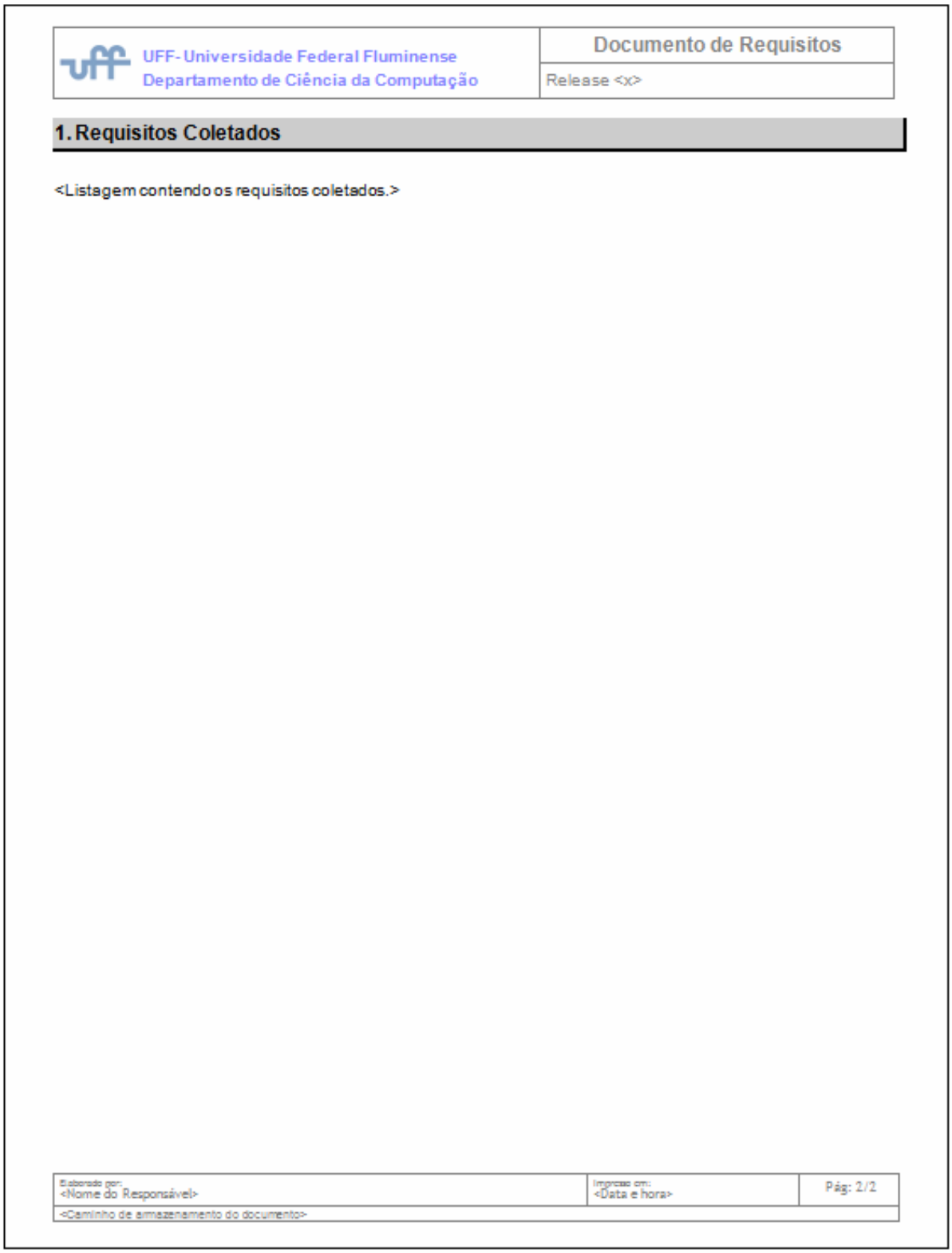
UFF- Universidade Federal Fluminense
Departamento de Ciência da Computação

<Nome do Sistema>

<Nome do Módulo>

Documento de Requisitos

Controle de Revisão			
Release	**Data**	**Motivo**	**Responsável**
<x>	<dd/mm/aaaa>	<Descrição do motivo>	<Nome do responsável pela alteração do documento>



UFF- Universidade Federal Fluminense
Departamento de Ciência da Computação

Documento de Requisitos
Release <x>

1. Requisitos Coletados

<Listagem contendo os requisitos coletados.>

Elaborado por:
<Nome do Responsável>

Impresso em:
<Data e hora>

Pág: 2/2

<Caminho de armazenamento do documento>

9.3. Documento de Visão

<Identificação do Projeto>

Documento de Visão

Controle de Revisão			
Release	**Data**	**Motivo**	**Responsável**
<x>	<dd/mm/aaaa>	<Descrição do motivo: Criação ou alteração>	<Nome do responsável pela criação ou alteração do documento>

UFF- Universidade Federal Fluminense Departamento de Ciência da Computação	Documento de Visão Release <x>

1. Introdução

<Descrição geral do problema>

2. Areas envolvidas

Area	Descrição
<Nome da Area>	<Descrição da Area>

3. Visão geral do Problema

ID	Problema	Area Envolvida	Impacto
<x>	<Descrição do Problema>	<Nome da Area>	<Descrição do Impacto>

3. Solução geral

<Descrição da solução geral>

4. Necessidades

ID	Necessidades
<x>	<Descrição da Necessidade>

4. Requisitos de Arquitetura

<Descrição dos Requisitos de Arquitetura para a solução>

9.4. Análise de Riscos

UFF- Universidade Federal Fluminense
Departamento de Ciência da Computação

<Identificação do Projeto>

Análise de Riscos

Controle de Revisão			
Release	**Data**	**Motivo**	**Responsável**
<x>	<dd/mm/aaaa>	<Descrição do motivo: Criação ou alteração>	<Nome do responsável pela criação ou alteração do documento>

UFF- Universidade Federal Fluminense Departamento de Ciência da Computação	Análise de Riscos
	Release <x>

1. Mapeamento dos Riscos

RISCO	PROBABILIDADE DE OCORRER	IMPACTO	PLANO DE ATENUAÇÃO
1. < Descrição do Risco>	<Percentual>	<Valor do Impacto (1..10)>	<x>

2. Plano de Atenuação

2.1. Plano de Atenuação associado ao risco <x>

<Descrição do plano de atenuação do risco>

9.5. Descrição do caso de uso Cadastrar Paciente

UFF- Universidade Federal Fluminense
Departamento de Ciência da Computação

Desenvolvimento de um Sistema de Informação para suporte ao cuidado multidisciplinar em Insuficiência Cardíaca

Especificação de Caso de Uso – Cadastro de Paciente

UC001 – Cadastrar Paciente

Controle de Revisão			
Release	**Data**	**Motivo**	**Responsável**
1	18/07/2009	Criação do documento	Raphael Domingues Costa / Welisson Reich de Jesus

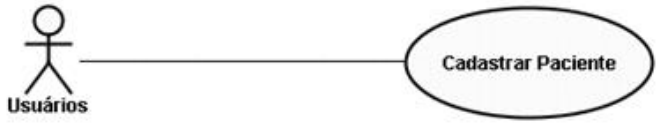
UFF- Universidade Federal Fluminense Departamento de Ciência da Computação	**Caso de Uso**
	Release 1

1. Responsável pela Informação
Sabrina Bernardez.

2. Descrição
Permitir o cadastramento de um paciente.

3. Ator
Usuários.

4. Diagrama de Caso de Uso



5. Outros Diagramas
Não se aplica.

6. Pré-Condições
- O ator deverá estar autenticado no sistema.
- O ator deverá ter acesso ao cadastro de paciente.

7. Pós-Condições
Paciente cadastrado com sucesso.

8. Nível de complexidade:
Baixa.

9. Fluxo Básico
1. O ator seleciona a opção para cadastrar paciente.
2. O sistema apresenta:
   - Formulário conforme regra de interface **RI001 - Cadastro de Paciente**.
   - Opção para cadastrar o paciente.
3. **{Cadastrar Paciente}**
   O ator preenche as informações sobre o paciente. [**FA2**].
4. O sistema valida os dados preenchidos conforme: [**FE1**]
   - Regra de Interface **RI001 - Cadastro de Paciente**.
   - Regra de Negócio **RN001 - Cadastro do Paciente**.
5. O sistema efetua o cadastro do paciente conforme informações fornecidas, limpa os campos do formulário de cadastro e emite confirmação de cadastro.

6. **{Encerramento do Caso de Uso}**
   O Caso de Uso se encerra. [**FA1**]

10. Fluxos Alternativos

FA1- Novo Cadastro
Ocorre quando o ator decide fazer um novo cadastro de paciente.

1. O fluxo continua com novo preenchimento do formulário, a partir de **{Cadastrar Paciente}.**

FA2- Desistência
Ocorre quando o ator desiste de cadastrar um paciente.

1. O fluxo se encerra em **{Encerramento Caso de Uso}**

11. Fluxos de Exceção

FE1- Erro no preenchimento do cadastro
Ocorre quando a validação das informações preenchidas detecta algum problema que impede o prosseguimento normal.

1. O sistema detecta algum erro no preenchimento das informações.
2. O sistema apresenta de uma única vez, as mensagens aplicáveis de acordo com o erro, conforme abaixo:
   a. É obrigatório informar o nome do paciente.
   b. É obrigatório informar a data de nascimento do paciente.
   c. É obrigatório informar a origem do paciente.
   d. É obrigatório informar o número do prontuário.
3. O ator confirma visualização da mensagem.
4. O fluxo continua em **{Cadastrar Paciente}**.

12. Regras de Negócio

RN001 - Cadastro do Paciente
Para cadastrar um paciente é necessário atender às seguintes regras:

- É obrigatório informar o Nome do Paciente.
- É obrigatório informar a Data de Nascimento do Paciente.
- É obrigatório selecionar a Origem do Paciente.
- É obrigatório informar a Data de Cadastro do Paciente.
- É obrigatório informar o Número do Prontuário.

13. Regras de Interface

RI001 - Cadastro de Paciente.

Nome	Tipo	Formato	Editável	Obrigatório	Domínio/Valor	OBS
Nome	Texto		S	S		
Origem	Seleção		S	S	Opções: Unidades participantes da Pesquisa.	
Sexo	Check Box		S	N		
Data de Nascimento	Número	dd/MM/yyyy	S	S		
Idade	Número		N	N		Calculado automaticamente.
Profissão	Texto		S	N		
Naturalidade	Seleção		S	N	Opções: Sigla dos Estados do Brasil.	
Endereço	Texto		S	N		
Bairro	Texto		S	N		
CEP	Número	xx.xxx-xxx	S	N		
Cidade	Texto		S	N		
Estado	Seleção		S	N	Opções: Estados do Brasil.	
Telefone	Número	xx xxxx-xxxx	S	N		Campo contendo DDD e Telefone.
Prontuário	Número		S	S		
Cor	Check Box		S	N		

9.6. Plano de Testes

UFF- Universidade Federal Fluminense
Departamento de Ciência da Computação

<Identificação do Projeto>

Plano de Testes
<Módulo>

Controle de Revisão			
Release	**Data**	**Motivo**	**Responsável**
<x>	<dd/mm/aaaa>	<Descrição do motivo: Criação ou alteração>	<Nome do responsável pela criação ou alteração do documento>

1. Testes

1.1. Teste de Validação

ID	DESCRIÇAO	CASO DE USO	RESULTADO
TE<x>	<Descrição do teste>	<Referência ao caso de Uso>	<Presença ou Ausência de erro. No caso da presença indicar o erro na parte Erros>

1.2. Teste de Interface e Navegação

ID	DESCRIÇAO	CASO DE USO	REGRA DE INTERFACE	RESULTADO
TE<x>	<Descrição do teste>	<Indicar o caso de uso que está sendo avaliado>	<Indicar a regra de interface correspondente ao caso de uso que está sendo utilizado, caso exista>	<Presença ou Ausência de erro. No caso da presença indicar o erro na parte Erros>

1.3. Teste de Integração

ID	DESCRIÇAO	MODULOS ENVOLVIDOS	RESULTADO
TE<x>	<Descrição do teste>	<Lista de módulos>	<Presença ou Ausência de erro. No caso da presença indicar o erro na parte Erros>

2. Erros

2.1. Erros dos testes de validação

ER	ID	ERRO
ER<x>	TE<x>	<Descrição do erro>

2.2. Erros dos testes de interface e navegação

ER	ID	ERRO
ER<x>	TE<x>	<Descrição do erro>

2.3. Erros dos testes de integração

ER	ID	ERRO
ER<x>	TE<x>	<Descrição do erro>