Universidade Federal de Santa Catarina
Departamento de Informática e Estatística
Curso de Sistemas de Informação

Estudo do uso de Metodologias Ágeis no Desenvolvimento de uma Aplicação de Governo Eletrônico

DIOGO LUDVIG
JONATAS DAVSON REINERT

Florianópolis – SC
Ano 2007 / 2

DIOGO LUDVIG
JONATAS DAVSON REINERT

Estudo do uso de Metodologias Ágeis no Desenvolvimento de uma Aplicação de Governo Eletrônico

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Sistemas de Informação

Orientador: Prof. José Eduardo De Lucca
Co-orientador (a): Profa. Maria Marta Leite

Banca Examinadora
Rafael Savi
Prof. José Eduardo De Lucca
Profa. Maria Marta Leite

Dedico este trabalho ao meu Vovô

Jonatas

AGRADECIMENTOS

Agradeço principalmente aos meus pais Marcos e Madalena pelos exemplos e ensinamentos.

Aos meus irmãos, familiares e amigos pelo companheirismo durante a vida. A minha namorada Gabriele, sempre ao meu lado nos momentos difíceis me apoiando e incentivando.

Agradeço também ao Luiz pela ajuda no Java e aos meus orientadores Marta e De Lucca pelo auxílio para que este trabalho fosse concretizado.

Diogo

Meus sinceros agradecimentos aos meus avôs, Bruno Reinert não mais presente e Catarina Reinert por todo incentivo durante esta longa jornada do meu curso.

A minha inseparável noiva, Elaine, que esteve sempre presente me ajudando e incentivando na conclusão deste trabalho.

Ao meu grande e inseparável amigo Alexandre por sempre me ouvir nas minhas horas de desespero.

E por fim, ao meu orientador, Prof. José Eduardo De Lucca e co-orientadora, Maria Marta Leite por ter acreditado na conclusão deste trabalho.

Muito obrigado!

Jonatas

Resumo

Com o crescimento da indústria de *software*, devido a uma demanda da sociedade, os problemas relacionados ao seu desenvolvimento, como falta de gerenciamento e manutenção, complexidade e alto custo de implementação, se tornaram mais visíveis. Dentre estes, o principal ponto de falha em um projeto de *software* é o seu gerenciamento. Essa falha desperta o interesse da comunidade de desenvolvimento de *software* em melhorar os processos relacionados ao gerenciamento, entre os quais se destacam os processos ágeis, por ocasionarem uma menor burocracia e resultados mais diretos. Tendo em vista a melhoria desses processos ágeis, esse trabalho tem como objetivo apresentar uma análise comparativa entre as metodologias ágeis *easYProcess* e *Scrum* bem como as diferenças entre a abordagem ágil e tradicional de desenvolvimento de *software*. Para isto, realiza um estudo e aplicação das metodologias citadas na construção parcial de um componente de *software* para prefeituras. Diante disto, percebemos que ambas as metodologias cumprem o seu papel. O *YP* é mais detalhado em documentos e artefatos para o projeto como todo, já a metodologia *Scrum* se destaca nos aspectos de gerência e organização da equipe durante o processo de desenvolvimento.

Palavras-chave: Metodologias Tradicionais, Metodologias Ágeis, Análise Comparativa, *easYProcess*, *Scrum*.

Abstract

Due to a fast growth of the software industry, a society demand, problems related to software development such as lack of management and maintenance, complexity and high costs have become more visible lately. Among them, management seems to be the main problem related to a software project. This problem has called the attention of the software development community which has made efforts to improve processes related to management such as agile processes to decrease bureaucracy, obtaining, this way, direct results. Aiming at contributing to the improvement of the performance of these agile processes, the objective of this study is to make a comparative analysis between easYProcess and Scrum agile methodologies of software development as well as to show differences between agile and traditional approaches. In order to achieve its objective, this study applies the methodologies above mentioned to a partial construction of a software component for City Halls. Results suggest that both methodologies play their role well: during the process of software development, the easYProcess seems to be efficient when applied to documents and artifacts while the Scrum methodology has a good performance when applied to management and team organization.

Keywords: Methodologies, Agile Methodologies, Comparative Analysis, easYProcess and Scrum.

Sumário

Índice de Figuras

Índice de Gráficos e Tabelas

Lista de Abreviaturas e Siglas

YP – easYProcess

UFCG – Universidade Federal de Campina Grande

TI – Tecnologia da Informação

XP – Extreme Programming

RUP – Rational Unified Process

AM – Agile Modeling

UML – Unified Modeling Language

PET – Programa de Educação Tutorial

TAA – Tabela de Alocação de Atividades

CVS – Current Version System

SVN – Subversion

ROI – Return On Investment

CAPSs – Centro de Atendimento Psico-Socia

SUS – Sistema Único de Saúde

JSF – Java Server Faces

JPA – Java Persistence API

J2EE – Java 2 Enterprise Edition

EJB – Enterprise JavaBeans

SGBD – Sistema Gerenciador de Bando de Dados

DAO – Data Access Object

MVC – Model-view-controller

HTML – HyperText Markup Language

1 INTRODUÇÃO

Hoje a sociedade está cada vez mais dependente da indústria do *software* e, com isso, problemas relacionados ao processo de desenvolvimento de *software* ficam mais evidentes. Estes problemas incluem alto custo de implementação, alta complexidade, falta de manutenção, sem contar com a desigualdade entre as necessidades do usuário final e o produto desenvolvido.

Conforme o procedimento adotado pela empresa, as tomadas de decisão ao longo do projeto têm-se sucesso ou fracasso. Fatores como código de baixa qualidade, desligamento de membros com alto grau de conhecimento e mudança de requisitos são fatores que podem causar o fracasso do projeto.

Com isso, o interesse em melhorar os processos de desenvolvimento de *software* vem aumentando significativamente na Engenharia de Software. Melhorar o processo significa compreendê-lo melhor a fim de melhorar a qualidade do produto e reduzir custo, risco e prazo de desenvolvimento. Sommerville (2003, p.147) demonstra que a melhoria no processo de desenvolvimento de software aperfeiçoa a qualidade do produto final; e como destaca Côrtes (2001, p.20) a “preocupação com a qualidade deixou de ser um diferencial competitivo e passou a ser um pré-requisito básico para participação no mercado”.

Portanto, adotar processos mais simplificados, como as metodologias ágeis que procuram desenvolver software de maneira mais direta e menos burocrática (CAMARA, 2005), tem despertado um grande interesse entre as comunidades de desenvolvimento de software. Enquanto que as metodologias tradicionais têm seus processos baseados na produção de grande quantidade de documentação e de modelos para guiar o processo de desenvolvimento (FOWLER, 2001).

Um exemplo entre as metodologias ágeis pode-se citar YP – *easYProcess (easYProcess, 2006)*, que é uma metodologia desenvolvida através de um projeto da Universidade Federal de Campina Grande (UFCG). Por ser um método recente, com pouca pesquisa realizada bem como sua comprovação acadêmica, surgiu a

idéia da realização de um estudo mais profundo a respeito da metodologia ágil *easYProcess*.

Por outro lado, outra metodologia que será abordada neste trabalho é a *Scrum* (*Scrum Alliance, 2007)*, onde pesquisas mostram sua grande aceitação pela comunidade de desenvolvimento de *software* internacional, como mostra a Figura 1 abaixo:

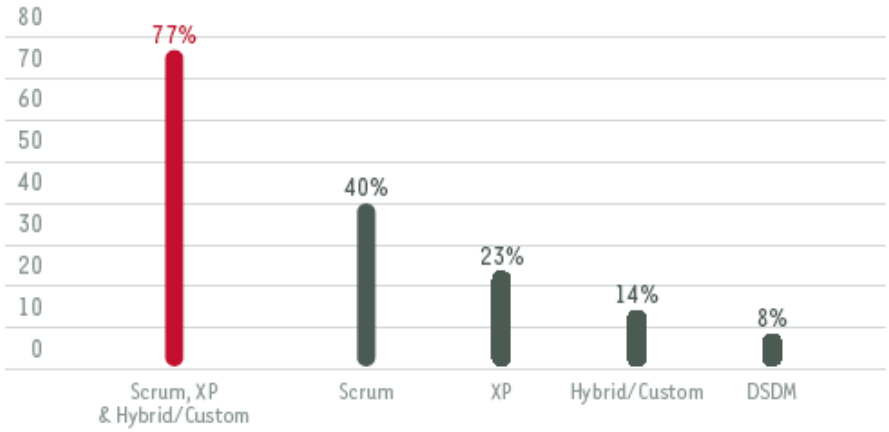


Figura 1 – Utilização de metodologias ágeis. Fonte: VERSIONONE, State of Agile Development.

1.1 FORMULAÇÃO DO PROBLEMA

Atrasos de cronograma e falhas em projeto de desenvolvimento de *software* são características indesejadas que, se não sofrerem algum tipo de controle pode levar ao fracasso de um projeto de *software*. Um dos primeiros obstáculos de grande parte das equipes de desenvolvimento que inicia um projeto de *software* é acabar enfrentando algum problema relacionado ao tempo para o desenvolvimento.

Outro fator é que a metodologia de desenvolvimento pode acabar prejudicando a evolução do projeto, pois seus integrantes terão que adequar-se à metodologia, a equipe pode apresentar resistência à mudança e pode haver muita burocratização desnecessária.

A evolução das técnicas para desenvolvimento de *software*, buscando evitar este quadro, fez surgir diversas metodologias e diversas práticas. Mais recentemente surgiu uma nova linha de metodologias denominadas metodologias ágeis responsáveis por ofertar uma significativa melhora nos processos e evitar os problemas comumente encontrados como documentação excessiva e gerenciamento dos requisitos como destaca Zanatta (2004).

Problemas como atraso dos projetos de *software*, orçamentos superfaturados, produtos com qualidade final insatisfatória e colaboradores descontentes, levam ao surgimento de vários estudos na parte de gerenciamentos de projeto de *software* e metodologias de desenvolvimento de *software* capazes de auxiliar nestes problemas encontrados por diversas equipes.

Projetos de software onde equipes se encontram no mesmo espaço físico já apresentavam muitos problemas na sua evolução, portanto em projetos onde diversas pessoas de diferentes localizações atuam em conjunto para a confecção de um componente de software se torna muito mais complicado. Com o objetivo de organizar projetos de software livre, onde pessoas de diferentes lugares podem colaborar com o desenvolvimento de projetos de software para prefeituras de todo o Brasil, foi criado o projeto Via Digital.

O Via Digital é um projeto que visa à implantação de um repositório digital de Software Livre permitindo o acesso ao desenvolvimento compartilhado de soluções com foco na realidade de pequenas prefeituras. Para isto, este projeto tem o apoio de diversas empresas e instituições acadêmicas por todo Brasil, visando à integração da comunidade para compor o seu repositório (Via Digital, 2007).

1.2 JUSTIFICATIVAS

Considerando o cenário que está se moldando em torno do setor de TI (Tecnologia da Informação[1]), não há mais espaço para projetos de *software* mal planejados. Projetos sempre estão sujeitos a riscos como: atrasos no cronograma,

sistemas obsoletos, mudança de requisitos, alta taxa de defeitos e saída de importantes membros da equipe de desenvolvimento que possuem conhecimento exclusivo e outros. As equipes de desenvolvimento, por outro lado, sempre querem que estes riscos passem longe do seu dia-a-dia.

Além disso, um dos grandes problemas que os alunos enfrentam durante o curso de Sistemas de Informação é a falta de apresentação de novas práticas de desenvolvimento de software, em particular as metodologias ágeis. Com isso, faz-se necessário, diante de um mercado com alto grau de competitividade, que os alunos busquem este conhecimento de modo que posso garantir alguma vantagem competitiva.

As metodologias ágeis de desenvolvimento têm-se mostrado como uma escolha viável para atender aos requisitos surgidos na nova realidade de mercado. Entre elas, é possível citar a *Scrum* que tem se destacado em projetos atuais na área de desenvolvimento de *software*. No cenário acadêmico temos uma nova metodologia que nasceu na Universidade Federal de Campina Grande, a *easYProcess*, uma metodologia que tem como objetivo auxiliar na prática o processo de desenvolvimento de *software* em projetos acadêmicos e que também pode ser usado em pequenos e médios projetos pelas empresas (*easYProcess*, 2006).

1.3 OBJETIVOS

Este trabalho tem alguns objetivos a serem alcançados. A seguir eles serão explorados com maiores detalhes.

1.3.1 OBJETIVO GERAL

Realizar na prática um estudo comparado entre as metodologias YP e *Scrum*, através da modelagem e implementação (parcial) de um componente de *software* baseado nestas metodologias.

---

[1] O termo TI (Tecnologia da Informação) serve para designar o conjunto de recursos tecnológicos e computacionais para geração e uso da informação. (WIKIPEDIA)

1.3.2 OBJETIVOS ESPECÍFICOS

I. Estudar as características gerais das metodologias ágeis e tradicionais de desenvolvimento de *software*;

II. Estudar mais detalhadamente as metodologias ágeis *easYProcess* e *Scrum*;

III. Modelar, com as duas metodologias, um componente de *software* que servirá como experiência para avaliação;

IV. Avaliar a utilização de ambas as metodologias;

V. Iniciar o desenvolvimento do componente baseando-se na modelagem mais apropriada.

1.4 DELIMITAÇÃO DO ESCOPO

Neste trabalho será realizada a análise de um componente de software, que poderá ser utilizado por pequenas prefeituras do Brasil, utilizando duas metodologias ágeis de desenvolvimento, a *easYProcess* e a *Scrum*.

Após as modelagens utilizando o *YP* e o *Scrum*, será feito uma análise comparativa entre as duas metodologias, e conforme o resultado desta análise será iniciado o desenvolvimento de um componente de software com metodologia mais apropriada.

1.5 ORGANIZAÇÃO DO TRABALHO

O presente trabalho está estruturado da forma descrita a seguir. O capítulo um apresenta esta introdução, visando o posicionamento do trabalho e a definição do seu escopo. O capítulo dois apresenta um estudo das metodologias de desenvolvimento. O capítulo três trata mais especificamente sobre a metodologia YP, da mesma forma que o capítulo quatro se aprofunda no estudo da metodologia *Scrum*. No capítulo cinco é apresentado um estudo de caso comparando as metodologias *YP* e *Scrum*, através do desenvolvimento de um pequeno componente

de *software*. Finalmente o capítulo seis apresenta os resultados alcançados na utilização de metodologias ágeis e na comparação entre as duas metodologias abordadas bem como as conclusões deste trabalho.

2 ESTUDO DE METODOLOGIAS DE DESENVOLVIMENTO

Este capítulo tem por objetivo apresentar uma visão geral sobre as metodologias tradicionais e ágeis de desenvolvimento de *software*. Em seguida é apresentada uma comparação entre as duas abordagens, mostrando algumas vantagens e desvantagens do uso de cada uma. Também são apresentadas as metodologias ágeis *easYProcess* e *Scrum* que serão utilizadas na modelagem do componente de *software*, bem como as práticas do *Extreme Programming* (XP) e *Rational Unified Process* (RUP). Para a confecção deste capítulo foi utilizado como base o utilizado no Trabalho de Conclusão de Curso da aluna Cecília Machado de Felippe (FELIPPE, 2007).

2.1 METODOLOGIAS DE DESENVOLVIMENTO DE SOFTWARE

Segundo Pressman (2001), processo de *software* é um conjunto de tarefas requeridas para a produção de um *software* com alta qualidade. O resultado do processo é um produto (*software*) que reflete a forma como o processo foi realizado. Embora existam vários processos de desenvolvimento de *software*, existem atividades genéricas comuns a todos eles (SOMMERVILLE, 2004):

- Especificação do *software*: definição do que o sistema deve fazer; definição dos requisitos e das restrições do *software*;
- Desenvolvimento do *software*: projeto e implementação do *software*, o sistema é desenvolvido conforme sua especificação;
- Validação do *software*: o *software* é validado para garantir que as funcionalidades implementadas estejam de acordo com o que especificado;
- Evolução do *software*: evolução do *software* conforme a necessidade de cliente.

Visando a qualidade final do produto gerado são propostos métodos que buscam padronizar as atividades e o ciclo de vida de um processo de *software*. Esta

padronização implica em criação de documentos, artefatos e marcos capazes de representar o contexto do *software* (NETO, 2004).

Portanto, uma metodologia de desenvolvimento de *software* é a representação simplificada do processo (SOMMERVILLE, 2004) que combina descrição do trabalho, procedimentos e convenções utilizadas por seus membros (COCKBURN, 2000). Ou ainda, uma metodologia de desenvolvimento de *software* é um conjunto de práticas recomendadas para o desenvolvimento de *software*, sendo que essas práticas geralmente passam por fases ou passos, que são subdivisões do processo, para ordená-lo e melhor gerenciá-lo (SOMMERVILLE, 2004).

O desafio mais comum dentro do desenvolvimento de *software* é entregar um produto que atenda as reais necessidades do cliente, dentro do prazo e orçamento previstos (FAGUNDES, 2005). Como suporte a esta atividade, a Engenharia de *Software* oferece metodologias que auxiliam os desenvolvedores durante o processo de desenvolvimento, entre elas, as metodologias tradicionais e as metodologias ágeis.

2.2 METODOLOGIAS TRADICIONAIS

Na década de 70 a atividade de "desenvolvimento de *software*" era executada de forma desorganizada, desestruturada e sem planejamento, gerando um produto final de má qualidade, que não correspondia com as reais necessidades do cliente (PRESSMAN, 2001). A partir deste cenário surgiu a necessidade de tornar o desenvolvimento de *software* como um processo estruturado, planejado e padronizado (NETO, 2004).

De acordo com Neto (2004), no início da padronização do processo foram mantidos conceitos típicos da Engenharia Civil, que ajudaram a sistematizar o processo de desenvolvimento de *software*. O desenvolvimento de *software* era descrito como um processo descritivo (FAGUNDES, 2005), baseado em um conjunto de atividades predefinidas e detalhadamente planejadas (LARMAN, 2002).

As metodologias consideradas tradicionais também são conhecidas como "pesadas" ou orientadas a documentação. Têm como característica marcante dividir

o processo de desenvolvimento em etapas e/ou fases bem definidas. Segundo SOARES (2004),

> essas metodologias surgiram em um contexto de desenvolvimento de software muito diferente do atual, baseado apenas em um *mainframe*[2] e terminais burros. Na época, o custo de fazer alterações era muito alto, uma vez que o acesso aos computadores era limitado e não existiam modernas ferramentas de apoio ao desenvolvimento do software, como depuradores e analisadores de código. Por isso, o software era todo planejado e documentado antes de ser implementado.

Muitas metodologias pesadas são desenvolvidas em cima do Modelo em Cascata (SOMMERVILLE, 2004). É um modelo em que as fases definidas são seguidas de maneira linear (LARMAN, 2002). Cada fase tem associado ao seu término uma documentação padrão que deve ser aprovada para que se inicie a etapa imediatamente posterior. Cada fase concluída gera um marco, que geralmente é algum documento, protótipo do software ou mesmo uma versão do sistema (NETO, 2004). De uma forma geral fazem deste modelo as etapas de análise de requisitos, projeto do software, geração de código e testes, como mostra a Figura 2.

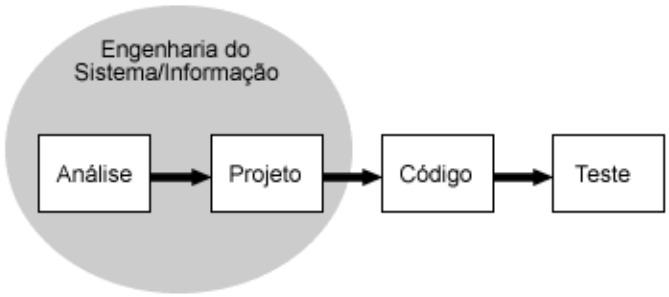


Figura 2 – Modelo em Cascata (Adaptado de PRESSMAN, 2001)

Também chamado de Modelo Clássico ou Seqüencial, o Modelo em Cascata sugere um desenvolvimento de software sistemático que engloba as seguintes atividades (PRESSMAN, 2001):

- Engenharia do Sistema/Informação e Modelagem: o trabalho começa com o levantamento de requisitos para todos os elementos do sistema e que serão pertinentes no desenvolvimento do

software. Esta visão geral é essencial quando o sistema deve interagir com outros elementos como *hardware*, pessoas e banco de dados;

- Análise de Requisitos: o processo de levantamento de requisitos é intensificado e focado especificamente para o desenvolvimento do *software*. Nesta fase os requisitos são documentados e revisados junto com o cliente;
- Projeto: o projeto do *software* é um processo que transforma os requisitos em uma representação cuja informação pode ser acessada com segurança e qualidade antes da geração de código. Assim como a análise de requisitos, o projeto é documentado e torna-se parte da configuração do *software*;
- Geração de Código: o projeto deve ser traduzido pela etapa de geração de código. Se o projeto for realizado de uma maneira detalhada, a geração de código pode ser executada mecanicamente;
- Testes: uma vez que o código foi gerado, inicia-se o processo de testes. Os testes devem ser conduzidos para descobrir erros e garantir que entradas definidas produzam resultados esperados;
- Manutenção: o *software* irá indubitavelmente sofrer mudanças após ser entregue ao cliente. Mudanças vão ocorrer porque erros serão encontrados, porque o *software* deve adaptar-se a um novo sistema operacional, ou porque o cliente quer que o software tenha novas funcionalidades. A manutenção do *software* reaplica cada uma das fases anteriores em um sistema já existente muito mais que em um novo sistema.

O problema do modelo Clássico é a divisão do projeto em fases distintas, o que dificulta possíveis alterações dos requisitos por parte do cliente. Portanto, este modelo é apropriado apenas quando os requisitos forem bem compreendidos (SOMMERVILLE, 2004). Embora seja o modelo mais usado em todo o mercado não

---

[2] Computador de grande porte, dedicado normalmente ao processamento de um volume grande de

é o mais eficaz, pois raros projetos de *software* seguem este fluxo linear, além das mudanças de requisitos que ocorrerem não serem de fácil adaptação, pois alteram toda a documentação já desenvolvida (NETO, 2004).

Processos tradicionais são baseados na produção de uma grande quantidade de documentação e de modelos para guiar a programação e que demandam muito tempo para serem gerados (FAGUNDES, 2005). Isso dificulta o controle do projeto, pois a cada mudança de requisitos será necessário uma volta ao início do mesmo para alteração de documentação (PRESSMAN, 2001), implicando em re-trabalho.

2.2.1 RUP

O RUP além de uma metodologia é visto como um produto de *software*, e como tal ele é projetado, desenvolvido, entregue e mantido como qualquer ferramenta de *software*. Alguns exemplos de indústrias que se utilizam desta metodologia são: *Alcatel*, *Xerox*, *Volvo*, *Intel*, *Visa*, *Oracle* e *Ernst & Young* entre outras (KRUCHTEN, 2003). Seu objetivo é garantir a criação de *softwares* de alta qualidade, que atenda às necessidades expressas pelo cliente e pelos usuários, e às restrições de prazo e custo (MARTINS, 2006).

2.2.1.1 As melhores práticas de desenvolvimento no RUP

Gerenciamento informal dos requisitos, não atendimento das necessidades dos usuários, incapacidade de lidar com as mudanças de requisitos, complexidade crescente e excessiva, qualidade ruim, testes insuficientes e baixo desempenho, são muitas das causas de fracasso em projetos de desenvolvimento de *software*. Para evitar estes problemas o RUP propõe práticas essenciais no desenvolvimento de *software*, e a seguir citam-se as principais.

---

informações (WIKIPEDIA).

Desenvolvimento Iterativo

Na abordagem iterativa proposta pelo RUP, o objetivo é conduzir o projeto em ondas, ou seja, iterações. Cada iteração é tratada de forma tradicional, alguns requisitos e riscos mais críticos são abordados, há um pouco de análise, implementação, testes e implantação (MARTINS, 2006). Depois há outra iteração, e assim por diante. A cada iteração tem-se uma versão testável do sistema, o que faz com que erros e mudanças de requisitos não sejam identificados somente no final do projeto.

Gerenciamento de Requisitos

Gerenciamento de requisitos é uma abordagem sistemática para concluir, organizar, comunicar e gerenciar os requisitos alterados no desenvolvimento de uma aplicação (KRUCHTEN, 2003). Um requisito é uma característica ou capacidade que o sistema precisa apresentar. O principal desafio de se gerenciar requisitos está no fato de que requisitos são dinâmicos e mudam durante a vida do projeto. Dentre os principais motivos dessas mudanças podem-se citar: usuário muda de idéia; o problema muda; mudanças técnicas e mudanças de mercado (MARTINS, 2006).

Arquitetura e Uso de Componentes

Componente pode ser definido como um pedaço do *software*, um módulo, um pacote ou um subsistema, que executa uma função específica. Esses componentes juntos formam a arquitetura do sistema. As vantagens de se desenvolver um sistema baseado em componentes são inúmeras. Dentre elas podemos citar a identificação, isolamento e correção de um erro com maior facilidade. A reutilização também é um ponto muito importante, podemos usar um mesmo componente em várias partes do sistema, tendo assim uma melhor organização e recuperação de erros mais acelerada.

Modelagem

Um modelo é uma simplificação da realidade que descreve completamente o sistema a partir de certo ponto de vista. O RUP trabalha com os modelos providos da UML[3], que são: modelo de análise; modelo de banco de dados, modelo de casos de uso; modelo de implantação; modelo de implementação; modelo de negócio; modelo de design e modelo de teste (MARTINS, 2006).

O uso destes modelos ajuda a compreender sistemas complexos como um todo, auxiliando a equipe na construção do mesmo.

Qualidade de Processo e Produto

No RUP a responsabilidade pela qualidade não é acrescentada ao produto por algumas pessoas, pelo contrário, a qualidade é de responsabilidade de todos os membros da organização de desenvolvimento (KRUCHTEN, 2003). Visto que a correção de problemas é de 100 a 1000 vezes mais cara de realizar após a implantação do sistema do que no início do projeto, a verificação constante da qualidade é importante em todo o ciclo de vida do projeto (MARTINS, 2006).

Controle de Mudanças

Controlar as mudanças é essencial quando a equipe é muito grande. Para isso o RUP prevê um controle de mudanças onde o objetivo é controlar as versões dos artefatos criados e modificados durante o projeto. E o desafio é ainda maior quando se tem uma equipe dispersa em muitos locais.

2.3 METODOLOGIAS ÁGEIS

A partir da década de 90 começaram a surgir novos métodos sugerindo uma abordagem de desenvolvimento ágil, em que os processos adotados tentam se

---

[3] UML (Unified Modeling Language) é uma linguagem para especificação, documentação, visualização e desenvolvimento de sistemas orientados a objetos. (WIKIPEDIA)

adaptar às mudanças, apoiando a equipe de desenvolvimento em seu trabalho (FAGUNDES, 2005). As metodologias ágeis surgiram como uma reação às metodologias pesadas (FOWLER, 2005) e tiveram como principal motivação criar alternativas para o Modelo em Cascata (HILMAN, 2004).

O termo “Metodologia Ágil” tornou-se popular em fevereiro de 2001, quando um grupo de 17 especialistas (referências mundiais em desenvolvimento de software) criou a Aliança Ágil e estabeleceram o Manifesto Ágil para o desenvolvimento de *software* (HIGHSMITH, 2002). Conforme Highsmith (2002), os valores do Manifesto Ágil são:

- **Indivíduos e interações valem mais que processos e ferramentas**: Equipes constroem sistemas de *software* e, para fazê-lo, elas precisam trabalhar junto com programadores, testadores, gerentes de projeto, modeladores e clientes. O ponto é que os fatores mais importantes a considerar são as pessoas e como elas trabalham juntas, porque se você não entender isso bem, de nada adiantarão as melhores ferramentas e processos. (AMBLER, 2004).
- **Um *software* funcionando vale mais que documentação extensa**: Em várias ocasiões é mais útil um protótipo simples que mostre o funcionamento de uma arquitetura do que um grande e complexo diagrama de classes. Também é mais simples para o cliente analisar o funcionamento do sistema através de um protótipo, mesmo que não seja real, do que através de um conjunto de diagramas de casos de usos e projetos de interface. Não se trata de abandonar o processo de documentação, mas apenas de utilizar a ferramenta certa para transmitir a informação desejada no momento. (SANTOS; MARTINS; LEAL, 2003).
- **A colaboração do cliente vale mais que a negociação de contrato**: Mesmo sendo difícil o cliente acertar na primeira vez, quem deve definir o que o sistema deve ou não fazer é o cliente. Pode-se dizer que o cliente não tem conhecimento suficiente para

especificar o sistema, que o cliente vive mudando de idéia sobre o que o sistema deve fazer que o cliente vive adicionando requisitos, etc., porém essa é a realidade do processo de desenvolvimento (SANTOS; MARTINS; LEAL, 2003). Ter um contrato com o cliente é importante, e entender os direitos e as responsabilidades de cada um pode se constituir nos alicerces do contrato, este contrato não é substituto para a comunicação. Os desenvolvedores de sucesso trabalham próximos aos clientes, se esforçam para descobrir o que os clientes necessitam e os educam durantes este processo (AMBLER, 2004).

- **Responder a mudanças vale mais que seguir um plano**: À medida que o sistema vai ganhando corpo os clientes começam a mudar a compreensão sobre o domínio do problema e sobre o que a empresa está construindo. A mudança é uma realidade no desenvolvimento de *software*, então embora seja de muita importância ter um plano de projeto traçado, este plano deve ser maleável às mudanças que aparecerão no decorrer do projeto.

De acordo com Fagundes (2005), para auxiliar as pessoas a compreender o enfoque do desenvolvimento ágil, os membros da Aliança Ágil refinaram as filosofias contidas em seu manifesto em uma coleção de doze princípios, aos quais os métodos ágeis de desenvolvimento de *software* devem se adequar. Estes princípios são (COCKBURN, 2000):

1. A prioridade é satisfazer ao cliente através de entregas de *software* de valor contínuas e freqüentes;
2. Entregar *softwares* em funcionamento com freqüência de algumas semanas ou meses, sempre na menor escala de tempo;
3. Ter o *software* funcionando é a melhor medida de progresso;
4. Receber bem as mudanças de requisitos, mesmo em uma fase avançada, dando aos clientes vantagens competitivas;
5. As equipes de negócio e de desenvolvimento devem trabalhar juntas diariamente durante todo o projeto;

6. Manter uma equipe motivada fornecendo ambiente, apoio e confiança necessário para a realização do trabalho;
7. A maneira mais eficiente da informação circular dentro da equipe é através de uma conversa face a face;
8. As melhores arquiteturas, requisitos e projetos provêm de equipes organizadas;
9. Atenção contínua a excelência técnica e um bom projeto aumentam a agilidade;
10. Processos ágeis promovem o desenvolvimento sustentável. Todos envolvidos devem ser capazes de manter um ritmo de desenvolvimento constante;
11. Simplicidade é essencial;
12. Em intervalos regulares, a equipe deve refletir sobre como se tornarem mais eficazes e então se ajustar e adaptar seu comportamento.

O Manifesto Ágil não é contra os modelos adotados pela abordagem tradicional. Inclui técnicas utilizadas na Engenharia de *Software*[4], porém não segue o padrão proposto pelas metodologias tradicionais (HIGHSMITH, 2002). Portanto, não rejeita os processos e ferramentas, a documentação, a negociação de contratos ou o planejamento, apenas mostra que eles têm importância secundária comparado com os indivíduos e iterações, como o *software* estar executável, com a colaboração e as respostas rápidas a mudanças e alterações.

Cockburn (2000) define desenvolvimento ágil de *software* como uma abordagem de desenvolvimento que trata os problemas das mudanças rápidas. Um processo ágil implica em um processo adaptativo e leve, que facilmente responde a mudanças (LARMAN, 2002). As metodologias ágeis adaptam-se a novos fatores decorrentes do desenvolvimento (FOWLER, 2005) ao invés de procurar analisar previamente tudo o que pode acontecer no andamento do projeto.

---

[4] Sommerville (2004) afirma que os métodos ágeis são fundamentados no Desenvolvimento Incremental.

Os métodos ágeis podem ser considerados como um meio termo entre a ausência de processo e o processo exagerado (ZANATTA, 2004). Optam por uma documentação apropriada evitando redundâncias e excessos, para que auxilie efetivamente o desenvolvimento do *software* (FILHO et alii, 2005). Aconselha a criação de documentos que têm valor (HILMAN, 2004), em que somente a documentação necessária é gerada.

Segundo Filho et alii (2005), outra característica importante das metodologias ágeis é que enfatizam os aspectos humanos de desenvolvimento de *software* ao invés dos aspectos da Engenharia. Logo, as pessoas são os principais condutores do sucesso do projeto (HIGHSMITH, 2002), não seguir um processo na tentativa de garantir a qualidade.

Outro aspecto interessante é apresentado por Camara (2005), de que as propostas ágeis baseiam-se na compreensão de que um projeto de *software* segue orientação em um serviço único, exclusivo. Um projeto nunca é igual ao outro, por mais que existam padrões e aplique-se “militarmente” uma metodologia (CAMARA, 2005). Desta forma, espera-se atender melhor às necessidades e aos requisitos do cliente, que muitas vezes são mutáveis.

É importante entender que as metodologias ágeis são uma atitude, não um processo descritivo. É uma forma efetiva de se trabalhar em conjunto para atingir as necessidades das partes interessadas no projeto (HILMAN, 2004). Reúnem boas práticas de programação e de gerenciamento de projetos a fim de produzir *software* com qualidade. Propõem uma alternativa que busca a melhoria do processo, tornando-o mais ágil e menos burocrático.

Os métodos classificados como ágeis possuem em comum o fato de ser aplicado em projetos não muito complexos, utilizando ciclos iterativos curtos, planejamento guiado por funcionalidades, retro alimentação constante, tolerância a mudanças, proximidade da equipe, intimidade com o cliente e um foco no ambiente geral de trabalho do time (HIGHSMITH, 2002).

2.3.1 YP

Assim como na construção de uma casa ou edifício, um software para estar pronto de acordo com as necessidades do cliente, envolve vários aspectos teóricos e práticos, a utilização de um processo de desenvolvimento bem como embasamentos da engenharia de *software*.

Conforme Hazzan e Dubinsky (2003) “processos de Engenharia de *Software* são difíceis de serem implantados, até mesmo na academia”. E, segundo comprovação feita pela Universidade Federal de Campina Grande (UFCG) pela disciplina de Engenharia de *Software* do curso de Ciência da Computação, perceberam que a utilização de processos como *Rational Unified Process* (RUP) e *eXtreme Programming* (XP) não renderam projetos a níveis aceitáveis, ou seja, projetos mal sucedidos e desenvolvedores mal capacitados.

Diante destas dificuldades, o Departamento de Sistemas e Computação da UFCG apresenta desde Maio de 2003, o *easYProcess*, um processo de software mais simplificado que se apóia em práticas do XP, RUP e *Agile Modeling* (AM). Sendo assim, o *YP* tem como objetivo auxiliar tanto a gerência do desenvolvimento de aplicações em disciplinas de engenharia de software quanto à aprendizagem dos conceitos desta disciplina na graduação, podendo também ser utilizado em projetos de pequeno e médio porte (GARCIA; LIMA; FERREIRA; JÚNIOR; ROCHA; MENDES; PONTES; ROCHA; DANTAS, 2007).

2.3.2 XP

A *Extreme Programming* é uma metodologia ágil de desenvolvimento de *software*. Alguns praticantes definem XP como a prática e a perseguição da mais clara simplicidade, aplicado ao desenvolvimento de *software*. XP é uma metodologia voltada para projetos cujos requisitos mudam com freqüência, que utilizem desenvolvimento orientado a objetos, equipes pequenas e desenvolvimento incremental (KUHN; PAMPLONA, 2007).

2.3.2.1 Valores

Os valores são as diretrizes da metodologia XP, eles definem as atitudes que as equipes devem ter e as principais prioridades da metodologia. Para uma empresa estar realmente utilizando o XP ela deve obedecer a todos esses valores, que são:

- **Comunicação:** O primeiro valor da XP é a comunicação. Ela está presente em todas as etapas de desenvolvimento, pois a comunicação se da através da solicitação do cliente e entre a equipe de desenvolvimento. Esta comunicação permite que todos os detalhes do projeto sejam tratados com a atenção e agilidade que são necessárias. Esta comunicação deve ser da forma mais direta e eficaz possível (GONÇALVES, 2007).
- **Coragem:** Por ser considerado um processo de desenvolvimento novo e baseado em diversas premissas que contrariam o desenvolvimento tradicional, o XP exige que os desenvolvedores tenham coragem, como por exemplo, para: Manter o sistema simples; Permitir que o cliente defina prioridades, Colocar desenvolvedores e clientes frente a frente, entre outros.
- ***Feedback*:** Quanto mais cedo a equipe de desenvolvimento liberar versões do sistema para o cliente, e receber deste a sua avaliação sobre o *software*, mais barato custará para o projeto a adequação do sistema as verdadeiras necessidades do cliente.
- **Simplicidade:** O XP contraria o modelo tradicional de desenvolvimento de *software*, e prega que deve ser desenvolvido apenas o que será necessário para o hoje, não se preocupando com futuras necessidades que nem se sabe que serão mesmo necessárias. E o que for desenvolvido, deve ser feito da maneira mais simples possível, codificar somente o necessário para atender as necessidades do cliente

2.3.2.2 Práticas

As práticas do XP são um conjunto de atividades que devem ser seguidas pelas equipes que desejam utilizar a *Extreme Programming*. Onde a fraqueza de umas é compensando com os pontos fortes de outras, havendo assim uma sinergia muito grande entre elas. Tais práticas são:

- **Cliente Presente:** O XP trabalha com uma visão diferente do modelo tradicional em relação ao cliente. O XP sugere que o cliente esteja no dia-a-dia do projeto, acompanhando os passos dos desenvolvedores, onde a sua ausência representa sérios riscos ao projeto (KUHN; PAMPLONA, 2007).
- **Jogo de Planejamento:** Trata-se de uma reunião onde o cliente avalia as funcionalidades que devem ser implementadas e prioriza aquelas que farão parte do próximo *release*[5] ou da próxima iteração[6].
- ***Stand Up Meeting*:** São reuniões realizadas a cada início de dia, de pequena duração, em torno de 20 minutos, onde seus integrantes permanecem de pé. Tendo como propósito uma reunião que vai direto ao assunto, rápida, ágil e simples. Objetiva atualizar a equipe sobre o que foi feito no dia anterior e trocar experiências das dificuldades encontradas, tendo assim uma redução do tempo de *feedback* dentro da equipe evitando o prolongamento dos problemas.
- **Programação em Par:** O XP exibe que todo e qualquer código implementado no projeto seja efetuado em dupla, chamada de programação em par. É uma técnica onde dois programadores trabalham juntos no mesmo problema, ao mesmo tempo e no mesmo computador. Um deles é responsável pela digitação

[5] Releases são módulos do sistema que geram um valor bem definido para o cliente.
[6] Iteração são períodos de tempo de poucas semanas na qual a equipe programa um conjunto de funcionalidades acordado com o cliente (GONÇALVES, 2007).

(condutor) e outro acompanha o trabalho do parceiro (navegador) (KUHN; PAMPLONA, 2007).

- ***Refactoring*:** Tem como objetivo melhorar alguma qualidade não-funcional do sistema, como: simplicidade, flexibilidade, clareza e desempenho. Destaca-se como exemplo de r*efactoring*: mudança de nome de variáveis, mudanças nas interfaces dos objetos, pequenas mudanças arquiteturais, encapsular código repetido em um novo método e generalização de métodos.
- **Código Coletivo:** O XP adota que o código não deve ser segmentado em partes, e dividido entre os desenvolvedores, pois quando esta parte do projeto necessitar de alterações somente uma pessoa estará capacitada para isto. Os desenvolvedores devem ter acesso a todas as partes do código, e alterá-lo quando necessário, pois o código é coletivo.
- **Padrões de Desenvolvimento:** Um dos valores do XP é a comunicação, e o código também é uma forma da equipe se comunicar. Uma forma clara de se comunicar através do código, é manter um padrão no projeto para que qualquer um possa rapidamente entender o código lido (KUHN; PAMPLONA, 2007).
- ***Design* Simples:** Como o XP foca que o maior valor possível seja gerado para o cliente, ele prega também um *design* do sistema o mais simples possível, a ponto que atenda a necessidade do cliente de forma clara.
- **Metáfora:** A criação de comparações a analogias com o assunto em questão faz com que as pessoas entendam de uma forma mais rápida o problema e possivelmente não esqueçam mais. Facilita a comunicação e fixação dos assuntos entre os envolvidos.
- **Ritmo Sustentável:** Esta prática defende que os desenvolvedores não devem ficar além do seu horário para que os prazos sejam cumpridos. Pois no começo esta prática até que gera resultado,

porém, depois de poucos dias o rendimento cai drasticamente devido ao cansaço físico do desenvolvedor.

- ***Releases* Curtos:** Ao invés de gerar uma versão para o cliente somente perto do prazo de entrega do projeto, no XP tem se a prática da liberação de *releases* em um curto espaço de tempo, onde um conjunto reduzido de funcionalidades é produzido e lançado para que o cliente já possa rapidamente utilizar o *software* e dar o *feedback* necessário.

2.3.3 SCRUM

Scrum é uma abordagem simples aplicada ao gerenciamento de tarefas complexas (CRUZ, 2006). Trata-se mais especificamente de uma metodologia ágil de desenvolvimento de *software*, que tem como principais características ser um processo empírico, iterativo e incremental.

Em suas premissas temos que o desenvolvimento de *software* é muito complexo e imprevisível para ser planejado totalmente no início do projeto. Ao invés disso, o processo deve ser controlado empiricamente para garantir a visibilidade, inspeção e adaptação (MARÇAL; FREITAS; SOARES; MACIEL; BELCHIOR, 2007).

Foi desenvolvido por Ken Schwaber e Mike Beedle na década de 90, possui papéis e fases bem definidas, que veremos com maiores detalhes a seguir.

2.3.3.1 Conceitos e Artefatos

A seguir veremos alguns dos mais importantes conceitos e artefatos utilizados dentro do processo de desenvolvimento com *Scrum*.

- ***Product Backlog:*** Temos no *product backlog* uma lista priorizada dos requisitos, tanto funcional como não-funcional. Em cada item desta lista temos um valor de negócio associado, onde podemos medir o retorno do projeto e a priorização dos itens.

- ***Sprint:*** Cada iteração do processo de desenvolvimento é denominada de *Sprint*. A duração de cada *Sprint* é recomendando que fique entre dois e quatro semanas.
- ***Sprint Backlog:*** É uma lista de tarefas, onde temos o trabalho da equipe em cada *Sprint* do processo. A lista nasce durante o planejamento do *Sprint*. As tarefas do *Sprint Backlog* são o que a equipe definiu como sendo necessário para a fluência da realização dos itens do *Product Backlog* nas funcionalidades do sistema. Cada tarefa é identificada pelo seu responsável e a sua quantidade estimada de trabalho restante (SCRUM FOR TEAM SYSTEM, 2007).

2.3.3.2 Papéis e Responsabilidades

A seguir veremos três dos principais papéis implementados pelo *Scrum*.

- ***Product Owner*****:** É o "dono do produto", identificando o interesse de todos no projeto. Além de priorizar os requisitos do projeto é o responsável pelo seu *ROI* (*return of investment*). Trocando em miúdos: é o dono do produto que sabe o que vai gerar retorno ao projeto (CRUZ, 2006).
- ***ScrumMaster*****:** É a pessoa responsável por fazer o *Scrum* funcionar. Deve ensinar a metodologia a todos os envolvidos no processo, assim como assegurar que todos sigam suas regras e práticas. Trabalha juntamente com o *Product Owner* na organização dos requisitos.
- ***Team Members*****:** Desenvolvem as funcionalidades do produto, são responsáveis coletivamente pelo sucesso da iteração e conseqüentemente pelo projeto como um todo (MARÇAL; FREITAS; SOARES; MACIEL; BELCHIOR, 2007).

2.3.3.3 Fluxo de Desenvolvimento

No *Scrum*, um projeto se inicia com a visão do produto que será desenvolvido. Em seu desenvolvimento, são realizados vários *Sprints*. A cada *Sprint* são estabelecidos novos itens a serem desenvolvidos, e durante sua realização temos diariamente reuniões de aproximadamente 15 minutos com os membros da equipe.

No final de cada *Sprint*, é feita uma retrospectiva para avaliar como esta o andamento do processo, posteriormente são levantados os próximos itens a serem desenvolvidos para a realização de um novo *Sprint*.

2.4 COMPARAÇÃO ENTRE METODOLOGIAS

A maioria das metodologias ágeis não possui nada de novo (COCKBURN, 2000). Segundo Highsmith (2002), o que existe de novo nos métodos ágeis não são as práticas que eles usam, mas os enfoques e os valores.

Basicamente os métodos ágeis se diferem dos tradicionais em dois aspectos (FOWLER, 2005):

- são adaptativos ao invés de prescritivos;
- são orientados às pessoas ao invés dos processos.

De acordo com Soares (2006), para uma metodologia ser efetivamente ágil ela deve aceitar mudanças ao invés de tentar prever o futuro. O problema não é a mudança em si, uma vez que mudanças é um aspecto intrínseco na vida do *software* (HILMAN, 2004). O problema é como receber, avaliar e responder às mudanças.

> Os modelos de processo convencionais adotam a estratégia de previsibilidade. Eles utilizam técnicas para levantar todos os requisitos e compreender o domínio do problema antes de iniciar o desenvolvimento. Depois de levantados os requisitos, é feito um planejamento para que as mudanças possam ser controladas no decorrer do processo de desenvolvimento do *software*. Os métodos ágeis optam pela adaptabilidade. Os requisitos são levantados aos poucos e o planejamento é contínuo, para que a adaptação às mudanças possa ocorrer (FILHO et alii, 2005).

Portanto, a principal diferença está na forma como as mudanças são tratadas durante o desenvolvimento do *software*. Os métodos tradicionais planejam previamente o projeto buscando controlar as mudanças (FILHO, 2006), enquanto os métodos ágeis incentivam a mudança nos requisitos (SOARES, 2006), planejando o projeto continuamente.

Para ilustrar como alterações no projeto são tratadas a seguir é apresentado o gráfico do custo do esforço de modificações conforme a fase do projeto considerando a abordagem tradicional e ágil (Figura 3). O eixo horizontal apresenta as fases do ciclo do projeto e o eixo vertical mostra o custo de investimento percentual da alteração comparado ao que já foi feito.

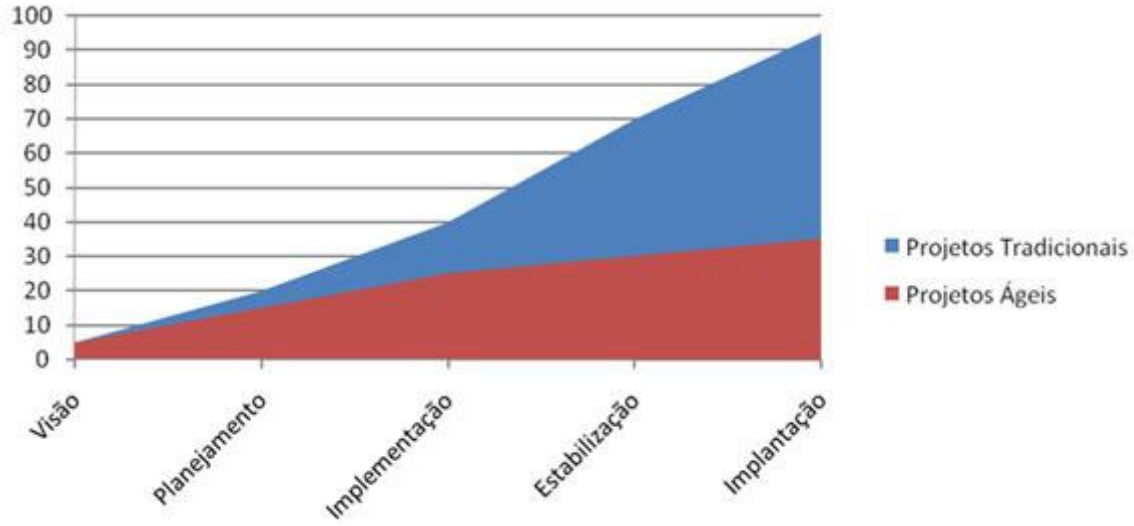


Figura 3 – Gráfico do preço de uma alteração no decorrer do projeto Fonte: (CAMARA, 2005)

Analisando o gráfico observa-se que uma mudança de escopo de uma funcionalidade na fase de Planejamento custa um esforço de re-trabalho em torno de 25% para ambas as abordagens, porém nas fases mais adiantes a diferença de esforço e custo é estrondosa. A alteração de requisitos é muitas vezes crítica nas metodologias tradicionais, pois estas não apresentam meios de se adaptar às mudanças.

Os métodos tradicionais têm o foco voltado para a documentação, necessitam de requerimentos completos e fixos, tornando-os uma metodologia pesada e não flexível (SEGUNDO e MOURA, 2004). A análise prévia dos requisitos

apresenta alto custo e caso se queira fazer alguma mudança no planejamento, a qualidade do *software* pode ser comprometida (SOARES, 2004). Por este motivo, as metodologias pesadas devem ser utilizadas somente quando os requisitos do sistema são estáveis e situações futuras são previsíveis. Já as metodologias ágeis devem ser utilizadas quando não se sabe exatamente o que se quer, pois permite que o *software* evolua no decorrer do desenvolvimento do projeto.

Assumindo que mudanças de especificação sempre vão ocorrer em todos os projetos, é possível afirmar que melhor será o projeto de *software* que mais se adaptar a estas mudanças (NETO, 2004). No entanto, segundo Zanatta (2004), as metodologias ágeis são ideais para aplicação em pequenas organizações, mas em organizações maiores sua aplicação é questionável. Enquanto o foco das metodologias tradicionais são os grandes projetos, em que necessitam de especificações detalhadas do mesmo, porém se tornando lentos para mudanças, as metodologias ágeis têm seu foco nos pequenos projetos, totalmente adaptável a mudanças, entretanto fraco na parte de documentos (NETO, 2004).

Considerando as características apresentadas, alguns aspectos significativos entre a abordagem tradicional e a abordagem ágil de desenvolvimento de *software* são apresentados na Tabela 1 (MAGALHÃES, 2004).

Abordagem Tradicional	Abordagem Ágil
Preditivo: detalhar o que ainda não é bem conhecido	**Adaptativo:** conhecer o problema e resolver o crítico primeiro
Rígido: seguir especificação predefinida, a qualquer custo	**Flexível:** adaptar-se a requisitos atuais, que podem mudar
Burocrático: controlar sempre, para alcançar objetivo planejado	**Simplista:** fazer algo simples de imediato e alterar, se necessário
Orientado a processos: segui-los possibilita garantir a qualidade	**Orientado a pessoas:** motivadas, comprometidas e produtivas
Documentação gera confiança	**Comunicação** gera confiança
Sucesso = entregar o planejado	**Sucesso** = entregar o desejado
Gerência = "comando e controle" voltado para trabalho em massa, ênfase no papel do gerente, com forte planejamento e disciplina fortes	**Gerência** = liderança/orientação trabalhadores do conhecimento, ênfase na criatividade, flexibilidade, atenção às pessoas
Desenvolvedor hábil (variedade)	**Desenvolvedor** ágil (colaborador)
Cliente pouco envolvido	**Cliente** comprometido (autonomia)
Requisitos conhecidos, estáveis	**Requisitos** emergentes, mutáveis
Retrabalho/reestruturação caro	**Retrabalho/reestruturação** barata

Planejamento direciona os resultados (incentiva a controlar).	**Resultado** direciona o planejamento (incentiva a mudar)
Conjunto de processos, com metodologia definida	**Conjunto de valores**, com atitudes e princípios definidos
Premia a garantia da qualidade	**Premia** o valor rápido obtido
Foco: grandes projetos ou os com restrições de confiabilidade, planej. estratégico/priorização (exigem mais formalismo)	**Foco:** projetos de natureza exploratória e inovadores, com equipes pequenas e médias (exigem maior adaptação)
Objetivo: controlar, possibilitando alcançar o objetivo planejado (tempo, orçamento, escopo)	**Objetivo:** simplificar processo de desenvolvimento, minimizando e dinamizando tarefas e artefatos

Tabela 1 – Comparação entre a abordagem tradicional e a abordagem ágil

3 YP – EASYPROCESS

O *easYProcess* (YP) é uma metodologia de desenvolvimento de *software* criada com o intuito de atender o desenvolvimento de projetos acadêmicos. Esta metodologia foi idealizada pela Professora Drª Francilene Procópio Garcia da Universidade Federal de Campina Grande (UFCG) e foi concebida no ambiente do grupo PET[7] Computação desta mesma universidade, no ano de 2003.

O YP foi definido com base nas características de um projeto de desenvolvimento de *software* acadêmico, como por exemplo, o escopo pequeno, em virtude da duração de um semestre, e a utilização de tecnologias consideradas estado da arte. Além disto, esta metodologia prima pela boa produtividade do projeto, amparado pelo uso de boas ferramentas de apoio e pela geração mínima de artefatos de *software*. O *easYProcess* é uma metodologia simples e de fácil entendimento pelos integrantes da equipe, porém é completo e robusto a ponto de gerar produtos de qualidade.

Esta metodologia está apoiada em práticas do XP, RUP e *Agile Modeling*, utilizando o que há de mais adequado em cada uma delas em relação ao ambiente acadêmico. Para a confecção deste capítulo foi utilizado como base o utilizado no Trabalho de Conclusão de Curso dos alunos Arthur Martins Pereira e Adriano Manoel Demétrio (PEREIRA; DEMÉTRIO, 2007).

As fases do *easYProcess*, conforme Garcia (GARCIA et al, 2007), são descritas nos itens deste capitulo a seguir.

[7] Inicialmente denominado de Programa Especial de Treinamento, sendo hoje intitulado de Programa de Educação Tutorial- PET, foi criado em 1979, e a partir do ano 2000, o Programa passou a ser vinculado à Secretaria de Educação Superior- SESu/MEC. O PET tem uma característica única com relação aos outros programas de capacitação do ensino superior, pois é o único que atinge o tripé das universidades: a pesquisa, o ensino e a extensão.

3.1 IDENTIFICAÇÃO DO ESCOPO DO PROBLEMA

Consiste no estudo inicial do escopo do problema, a fim de já obter conhecimento prévio que dê base ao entendimento e argumentação para a fase de conversa com o cliente, a qual será descrita posteriormente.

3.2 ESPECIFICAÇÃO DE PAPÉIS

É esperado que a equipe que participa do desenvolvimento do *software* seja dividida em papéis, os quais têm um conjunto de responsabilidades que determinada pessoa deverá desempenhar durante partes específicas do processo.

O *easYProcess* recomenda a utilização de cinco papéis:

- **Cliente**: É a pessoa que solicitou o desenvolvimento do *software*. Suas principais atividades são: definir os requisitos do sistema, priorizar as funcionalidades – identificando quais partes do sistema são mais urgentes – ajudar na elaboração do plano de releases – cliente e desenvolvedor devem concordar sobre que partes do sistema devem estar prontas em determinada data –, explicitar os testes de aceitação – os quais servem de validação da eficiência do sistema ao cumprir uma situação antes imposta –, identificar o perfil do usuário, identificar os objetivos de usabilidade, validar o protótipo de interface, validar o projeto arquitetural – somente após a validação do projeto arquitetural é que os desenvolvedores podem passar para a implementação do sistema –, e estar ativo no processo de desenvolvimento do sistema – o ideal seria que o cliente estive presente o máximo de tempo possível com os desenvolvedores, seja na definição do sistema, no refinamento do protótipo ou na implementação dos testes de aceitação. Melhorar a forma de apresentação do parágrafo acima. Há muitos – (travessões) que deixam o texto muito “picado”;
- **Usuário**: Esta é a pessoa que irá utilizar o sistema produzido, portanto o desenvolvimento do *software* deve atentar para as

características desta pessoa. Logo, a presença do mesmo no desenvolvimento do *software* seria de grande valia quando possível. Suas principais responsabilidades são: ajudar na definição dos testes de aceitação, ajudar na identificação dos objetivos de usabilidade, validar o protótipo de interface e avaliar continuamente a interface do sistema.

- **Gerente**: É aquele que toma decisões referentes aos ricos e rumos do projeto, sendo responsável por coordenar as atividades de todos os outros membros do projeto. Suas competências abrangem: conduzir os planejamentos e as ações dos desenvolvedores, elaborar o plano de desenvolvimento, avaliar sistematicamente os riscos descobertos, gerenciar configurações, coletar e analisar métricas, alocar testadores, resolver conflitos internos, tornar a documentação do projeto sempre atualizada e acessível.
- **Desenvolvedor**: É responsável por modelar os requisitos do sistema e produzir a codificação eficiente e correta para tais requisitos. O desenvolvedor deve utilizar-se de testes unitários a fim de encontrar erros em seu código e corrigir. As principais responsabilidades da pessoa que assume este papel são: levantar os requisitos funcionais e não funcionais junto com o cliente, auxiliar o gerente na elaboração de um plano de desenvolvimento, analisar e modelar a tarefa, gerar um protótipo da interface, identificar os objetivos de usabilidade, gerar testes de unidade, elaborar o projeto arquitetural – o qual deve ser validado pelo cliente –, construir o modelo lógico de dados, manter a integração contínua de código.
- **Testador**: É responsável por realizar testes de integração do sistema, por revisar os códigos dos desenvolvedores, refatorar se possível àqueles códigos a fim de torná-lo mais simples, legível, flexível e claro, porém caso haja um refatoramento de código, é necessário que o código seja testado tanto antes quanto depois do

refatoramento, garantindo assim que as funcionalidades permanecem intactas. Além disso, o testador deve gerar os testes de aceitação e elaborar o material para realização dos testes de usabilidade.

3.3 CONVERSA COM O CLIENTE

- Trata-se da primeira conversa entre o cliente e os desenvolvedores do sistema, tendo como objetivo fazer com que ambas as partes tenham uma idéia comum a respeito do sistema em questão. Ao final desta fase o escopo do problema deve estar bem definido, e deve-se conhecer o perfil dos usuários e suas necessidades, além saber quais os requisitos funcionais e não funcionais do sistema e deve-se também listar os riscos do projeto. Para tal, o *easYProcess* utiliza-se de elementos tais como os enumerados a seguir.

3.3.1 DOCUMENTO DE VISÃO

- Após a conversa inicial com o cliente, o documento de visão deve ser criado utilizando uma linguagem de fácil entendimento para o cliente, visto que este será um elo de comunicação entre o cliente e o desenvolver. O documento de visão deverá conter as idéias gerais sobre o que o sistema se propõe a fazer. Depois de concluído este documento de visão pelos desenvolvedores, o mesmo deve ser validado pelo cliente, a fim de verificar se aqueles compreenderam a visão do sistema, uma vez que este documento é de suma importância, pois serve como uma espécie de contrato entre as partes envolvidas.

3.3.2 REQUISITOS

- Dentro do Documento de Visão deve haver uma sessão em que são descritos os requisitos funcionais e não funcionais do projeto. Os requisitos são características e funções de um sistema. Os requisitos funcionais descrevem as funções que o *software* deve possuir. Já os requisitos não funcionais descrevem as propriedades e restrições do sistema. Os requisitos não

funcionais podem ser, por exemplo: de usabilidade, manutenabilidade e desempenho. Estes requisitos devem ser descritos pelos desenvolvedores em conjunto com o cliente.

3.3.3 PERFIL DO USUÁRIO

São as descrições das características dos usuários reais do sistema que são relevantes para o desenvolvimento do *software*. Estas descrições devem revelar às habilidades, as preferências, as limitações, os interesses dos usuários e o conhecimento prévio dos mesmos em relação às atividades que o sistema irá desempenhar e ao uso do computador. A descrição do perfil dos usuários pode aparecer no Documento de Visão. A seguir é mostrada uma lista de algumas características relevante a serem observadas na primeira conversa com cliente, segundo a documentação do *easYProcess*:

- Sexo;
- Canhoto, destro ou ambidestro;
- Uso de lentes corretivas;
- Faixa etária;
- Experiência prévia no uso de sistemas computacionais;
- Tempo de uso de sistemas computacionais;
- Freqüência de uso de sistemas computacionais;
- Plataforma computacional que utiliza com maior freqüência;
- Conhecimento prévio de outra versão do sistema que esta sendo desenvolvido (melhorado);
- Freqüência de uso de tal versão;
- Familiaridade com a língua inglesa.

3.3.4 OBJETIVOS DE USABILIDADE

Define quais metas de usabilidade devem ser atingidas pelo *software* desenvolvido. Servem para avaliar a usabilidade do sistema com base no desempenho do usuário e deve ser desenvolvido pelo cliente e pelo usuário.

- Geralmente estas metas referem-se à eficácia, eficiência, capacidade do sistema de ser memorável e aprendido pelos usuários, e pela a segurança. A eficácia diz respeito à forma com que as tarefas são realizadas e a disposição das informações necessárias para que a tarefa seja realizada. A eficiência trata do auxílio prestado ao usuário, pelo sistema, na realização de suas atividades.

3.4 INICIALIZAÇÃO

- Após a fase de conversa com o cliente e de criação do Documento de Visão, dá-se inicio a fase de inicialização, a qual é composta por cinco atividades, descritas a seguir.

3.4.1 MODELAGEM DA TAREFA

Sua função é esclarecer os detalhes de cada tarefa do sistema. A modelagem da tarefa auxilia na geração da interface do sistema, visto que a mesma identifica as relações de hierarquias e dependência entre as sub-tarefas. Este modelo pode ser realizado com a utilização da ferramenta iTAOS. Um exemplo de modelagem pode ser observado na Figura 4 a seguir.

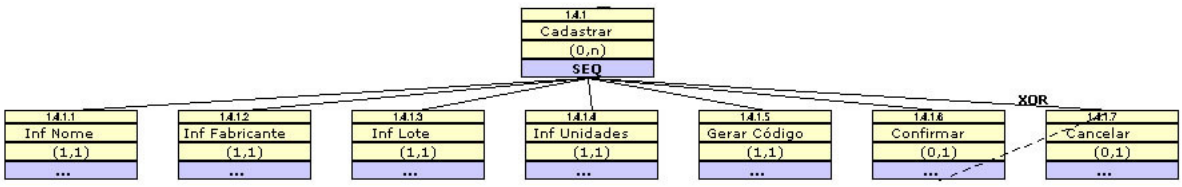


Figura 4 – Modelo da Tarefa

- O *software* iTAOS é uma ferramenta gráfica para análise e modelagem de tarefa que se utiliza do formalismo TAOS. Por sua vez, o TAOS é um formalismo que inicialmente visou à construção de um Sistema Baseado em Conhecimento e foi validado no ramo da biologia molecular. Porém este formalismo também é usado para a modelagem de tarefas, uma vez que o iTAOS abrange todos os requisitos que a modelagem da tarefa exige.

3.4.2 DEFINIÇÃO DAS USER STORIES

Esta é a fase do projeto em que se definem as *User Stories* e estima-se o tempo de desenvolvimento para cada uma delas. *User Stories* são funções que o sistema deve desempenhar e que são definidas pelo cliente e pelo desenvolvedor. Após a definição das *User Stories* envolvidas no projeto, e que pode sofrer alterações no decorrer do tempo, é possível construir uma tabela com as *User Stories* do sistema e suas respectivas estimativas, como observado na Tabela 2 a seguir, retirada do site oficial da metodologia YP. *(http://www.dsc.ufcg.edu.br/~yp/)*

User Stories	Tempo Estimado (Horas)
US1 – Cadastro dos clientes, incluindo dados profissionais e pessoais.	8
US2 – Cadastro dos produtos que serão comercializados, descrevendo as particularidades de cada produto.	6
US3 – Cadastro de categorias. Estas categorias dizem respeito às possíveis classificações a que um cliente pode pertencer.	4
US4 – Fazer registro das vendas efetuadas, incluindo informações dos produtos vendidos e do cliente.	8
US5 – Implementar mecanismo de busca para produtos (por nome, por código, por preço) e clientes (por nome, por categoria, por CPF).	4

Tabela 2 – Tabela de *User Stories*

3.4.3 PROTÓTIPO DE INTERFACE

Permite visualizar como se dá a interação do usuário com o sistema. O protótipo de interface pode ser construído com base no modelo de tarefas criado anteriormente.

3.4.4 MODELO LÓGICO DE DADOS

Trata-se da modelagem das entidades de domínio em um banco de dados se houver a necessidade.

3.4.5 PROJETO ARQUITETURAL

O Projeto Arquitetural é um artefato que permite explicitar como as partes do sistema interagem entre si ou com outros sistemas. Ele descreve o *software* com um alto nível de abstração. Com este diagrama, o qual pode também apresentar uma descrição do mesmo, é possível identificar os pontos críticos do sistema, as dependências e os requisitos não funcionais levantados junto com o cliente.

A fim de tornar ciente a arquitetura geral do sistema é importante a presença de todos os desenvolvedores na elaboração do mesmo, e para que haja a validação do que foi elaborado, faz-se necessário a presença do cliente.

3.5 PLANEJAMENTO

Esta fase do processo é marcada pelo planejamento de como o projeto será desenvolvido no tempo. Considerando que o YP foi elaborado com base em projetos acadêmicos, tal metodologia aconselha a duração de três meses para o desenvolvimento do *software*, sendo este espaço de tempo dividido em três partes, denominado *releases*, sendo estes *releases* divididos em duas partes, chamadas de iterações. Logo, esta fase de planejamento divide-se em planejamento de *release* e de iteração.

O planejamento de *release* diz respeito à alocação das *User Stories* nos *releases* existentes, atentando sempre para as exigências do cliente em relação a qual *User Story* deve ser realizada primeiramente, e para o fato de que se deve primeiro alocar as *User Stories* que representam um maior risco para o projeto, pois o risco do projeto ser abortado depois de muito tempo é minimizado. É notável lembrar que todas as *User Stories* devem ter pelo menos um teste de aceitação.

Portanto, uma *User Story* somente poderá ser considerada finalizada após a execução dos testes de aceitação.

Após a fase de planejamento de *release* dá-se a fase de planejamento de iteração. O planejamento de iteração é a divisão das *User Stories*, as quais estavam anteriormente englobadas em *releases*, em iterações. Cada *User Story* pode, se preciso, ser dividida em atividades com o intuito de facilitar a implementação de tal *User Story* e o gerenciamento da mesma. Portanto, cada iteração será um conjunto de atividades a serem desenvolvidas, e por sua vez, cada *release* será um conjunto de iterações.

Para auxiliar no planejamento das *releases* e das iterações é possível construir uma matriz de competência, mapeando as habilidades de cada membro da equipe e suas respectivas horas de dedicação semanal ao projeto.

As atividades oriundas das divisões das *User Stories* devem estar explicitadas na Tabela de Alocação de Atividades (TAA) contendo, para cada atividade, o tempo de desenvolvimento e o responsável pela mesma. Um detalhe interessante a ressaltar é que o tempo de desenvolvimento de cada atividade é estimado pelo próprio desenvolver responsável, respeitando o tempo estabelecido pela iteração.

Por fim, o *easYProcess* define que em hipótese nenhuma deve haver alteração no tempo dos *releases* ou iterações, e que caso haja uma necessidade de reajuste, o escopo é que deve ser alterado. Além disso, é de extrema importância a disponibilização, por parte do gerente, de todo o planejamento realizado em um local onde todos os membros da equipe possam acessar.

3.6 IMPLEMENTAÇÃO

As atividades definidas no plano de iteração são realizadas nesta fase. O principal artefato construído é o código do sistema. Com o intuito de gerar uma boa codificação, o *easYProcess* apresenta algumas práticas a serem seguidas:

3.6.1 PROPRIEDADE COLETIVA DE CÓDIGO

Toda a equipe é responsável por cada arquivo de código do sistema, não necessitando assim que se peça autorização para o criador do arquivo para que se possam fazer modificações no mesmo. Portanto, o código é propriedade coletiva e conseqüentemente, toda a equipe é responsável por ele.

Com esta prática é possível atingir uma melhoria continua do *software* e uma melhor recuperação de falhas, uma vez que trechos problemáticos de código podem ser identificados e resolvidos com mais facilidade por determinados membros da equipe, enquanto que outros não teriam a capacidade suficiente para realizar tal tarefa.

Porém, para que a propriedade coletiva do código seja efetuada com sucesso, é preciso que os códigos desenvolvidos sejam limpos, claros e de fácil entendimento, além de haver um controle de versão, a fim de saber quem está alterando cada parte de código em determinado momento, evitando conflitos.

3.6.2 BOAS PRÁTICAS DE CODIFICAÇÃO

Com o intuito de gerar um código limpo, o *easYProcess* recomenda a aplicação de práticas como: *Design* Simples, Refatoramento, Padrões de Projeto e Padrões de Codificação. Essas práticas são descritas a seguir.

3.6.2.1 *Design* Simples

O objetivo desta prática é atingir a geração do melhor código possível, ou seja, um código de fácil entendimento, auto-explicativo e de bom desempenho. Portanto, faz-se necessário atentar para a utilização correta da flexibilização de código, uma vez que esta prática, quando em excesso, pode levar a produção de código desnecessário.

3.6.2.2 Padrões de Codificação

Com o objetivo de facilitar o entendimento do código gerado por todos os membros da equipe, os mesmos devem antes de começar o desenvolvimento do projeto, acordar questões relativas ao padrão do código a ser desenvolvido.

Nomenclatura de variáveis e métodos, tamanho da identação utilizada, posição dos parênteses e chaves, são exemplos de questões a serem acordadas.

3.6.2.3 Padrões de Projeto

A fim de ganhar tempo, garantindo eficiência, e fazer com que o produto final se aproxime de um produto de qualidade, faz-se importante a utilização de padrões de projeto, os quais são soluções previamente pensadas e testadas, de qualidade assegurada, para a solução de diversos problemas comuns ao desenvolvimento de *software*.

3.6.2.4 Refatoramento

São pequenas modificações sobre o código considerado testado completamente e aceito de estar funcionando 100%, que visam melhorar algumas qualidades não funcionais, como por exemplo: simplicidade, flexibilidade, desempenho e clareza do código. Para tanto, é preciso que o tempo necessário para a execução do refatoramento seja previsto no planejamento da iteração. Além disso, o *easYProcess* recomenda utilizar o controle de versão para dar suporte a está prática.

3.6.3 INTEGRAÇÃO CONTÍNUA

Como forma de melhorar o gerenciamento do projeto e o trabalho dos desenvolvedores, caso estes não possuam horários em comum, o *easYProcess* utiliza-se da integração continua de código.

Na medida em que o código é produzido e testado é importante que o mesmo já seja integrado continuamente ao sistema como um todo, e para isso faz-se necessário a utilização de ferramentas com tal funcionalidade como, por exemplo, o CVS[8] (*Current Version System*) e o SVN[9] (*Subversion*).

---

[8] O CVS, ou *Concurrent Version System* (Sistema de Versões Concorrentes) é um sistema de controle de versão que permite que se trabalhe com diversas versões de arquivos organizados em um diretório e localizados local ou remotamente, mantendo-se suas versões antigas e os *logs* de quem e quando manipulou os arquivos. (WIKIPEDIA)

[9] SVN, também conhecido por *Subversion*, é um sistema de controle de versão desenhado especificamente para ser um substituto moderno do CVS. (WIKIPEDIA)

- Com a utilização desta prática, a coleta de métricas por parte do gerente do projeto fica mais consistente, facilitando a análise real do desenvolvimento, uma vez que o produto do desenvolvimento está todo centralizado.

3.6.4 TESTES

Para assegurar, na medida do possível, a qualidade e corretude do sistema, e conseqüentemente a satisfação do cliente e do usuário, o YP recomenda principalmente três tipos de teste:

3.6.4.1 Teste de Unidade

Testa a estrutura interna do código, ou seja, a lógica e o fluxo de dados. Recomenda-se que os testes de unidade sejam realizados antes da codificação, pois esta é uma conduta que ajuda na solução do problema e melhora a qualidade do código desenvolvido. Devem ser validados aspectos como: condições de limite, tratamento de erros, manipulação de dados inconsistentes e impróprios.

3.6.4.2 Teste de Aceitação

São situações definidas pelo cliente sobre como medir o sucesso do projeto. Usuários finais podem ser ouvidos também na definição destes testes para reforçar aspectos levantados pelo cliente. A elaboração destes testes é feita durante o levantamento das *User Stories*, porém a realização dos mesmos é feita após a finalização da *User Story.*

3.6.4.3 Teste de Usabilidade

Serve para verificar se os testes de usabilidade definidos pelo cliente e pelos usuários foram satisfeitos, servindo também como forma de verificar o grau de iteração dos usuários com o sistema. Para tal, o *easYProcess* utiliza-se do roteiro de atividades e do questionário pós-teste. Este último tem por objetivo identificar o grau de satisfação do usuário ao utilizar o sistema e deve ser executado após a execução do roteiro de atividades. Durante a execução do roteiro de atividades deve-se atentar ao levantamento de indicadores quantitativos, como por exemplo: número de

erros repetidos, número de acessos à área de ajuda do *software* e tempo para realização das tarefas.

Portanto, o YP recomenda a seguinte seqüência de passos para a execução dos testes de usabilidade: elaborar o roteiro de atividades; recrutar os usuários para teste; observar os usuários executando as atividades do roteiro; colher os indicativos quantitativos para análise; aplicar questionários pós-teste; analisar os resultados obtidos e sanar as falhas detectadas.

3.7 FINALIZAÇÃO DA ITERAÇÃO

Ao final da implementação e teste de uma iteração, deve-se preencher os campos de *status* e tempo real de desenvolvimento de cada atividade da iteração na Tabela de Alocação de Atividades (TAA). Caso haja mais alguma iteração ou release a ser desenvolvida, deve-se voltar para a fase de planejamento de *release* e seguir toda a seqüência de passos seguintes.

3.7.1 REUNIÕES DE ACOMPANHAMENTO

Esta fase do processo deve ocorrer semanalmente em reuniões que visam recolher e analisar métricas, utilizando-se do *Big Chart* – explicado a seguir – e da Tabela de Alocação de Atividades. Nesta fase deve-se atentar para os riscos do projeto e para a necessidade explícita de refatoramento de código.

Com as reuniões de acompanhamento, o gerente do projeto tem uma visão dos resultados obtidos pela equipe em uma semana de trabalho. A partir daí o gerente é capaz de identificar os pontos fortes e fracos do projeto, podendo então tomar as atitudes cabíveis a fim de melhorar o andamento e o sucesso do mesmo.

O *Big Chart* pode ser visto como a análise quantitativa do andamento do projeto. As métricas são definidas pelo gerente do projeto de acordo com os pontos a serem analisados, como por exemplo, número de classes existentes, número de linhas de código, número de testes de aceitação prontos, testes de unidades que estão rodando perfeitamente, dentre outros. Com este gráfico é possível identificar se o projeto não apresentou mudanças no decorrer da semana em questão, ou se

ocorreu algo inesperado, como exemplo, o aumento do número de classes e a manutenção no número de testes de unidade.

Durantes estas reuniões os desenvolvedores podem solicitar que os seus códigos sejam refatorados quando percebe que os mesmos não foram gerados da melhor forma possível, esta é a chamada necessidade de refatoramento explicito. Porém esta prática deve acontecer com pouca freqüência, visto que é responsabilidade do desenvolvedor criar o seu melhor código.

Além disso, a gerência de risco deve estar sempre em pauta nestas reuniões uma vez que essa metodologia foi proposta para diminuir os riscos através da verificação constante da situação do projeto. Riscos são situações indesejáveis que ocorrem durante o processo de desenvolvimento. Alguns exemplos comuns são: custo do projeto, prazo determinado para o desenvolvimento do projeto e uso de tecnologias desconhecidas por parte da equipe. Para realizar a gerência de riscos, o *easYProcess* utiliza-se da tabela de riscos, uma tabela contendo o risco identificado, a data de identificação, o responsável pela solução, o grau de prioridade e a solução encontrada. Com a tabela de risco é possível manter um histórico dos riscos encontrados e suas respectivas soluções a fim de agilizar a solução dos riscos futuramente identificados.

Outro fator que deve ser considerado em reuniões de acompanhamento é a necessidade de mudanças. Existem vários tipos de mudanças e cada uma tem o seu grau de impacto no andamento do projeto. Algumas implicam apenas em modificações de implementação, outras em alocações de tarefas, outras em alterações no planejamento ou ainda do modelo lógico e projeto da arquitetura. Logo, cada mudança identificada deve ser analisada cuidadosamente pelo gerente do projeto, o qual tomará as atitudes cabíveis em resposta ao acontecimento. Cabe lembrar que o tempo determinado para as iterações e *releases* não deve ser alterado, mesmo em decorrência destas mudanças, restando à alteração do escopo do prometido, uma vez que acordado com o cliente anteriormente.

3.8 VERSÃO DO PRODUTO

Após a finalização do sistema, realizam-se os testes de usabilidades sobre o mesmo, feito pelos usuários, e caso o resultado seja positivo, gera-se a versão final do produto.

4 SCRUM

Scrum é uma metodologia ágil de desenvolvimento de *software* que utiliza processo de desenvolvimento iterativo e incremental. Foi desenvolvido na década de 90 por Jeff Sutherland, Ken Schwaber e Mike Beedle baseando-se em experiências próprias no desenvolvimento de sistemas e processos. (CRUZ, 2006)

Seu nome *Scrum* surgiu de uma metáfora a uma tática utilizada no *Rugby*[10] para recuperar uma bola perdida. (ZANATTA, 2004) É baseado em princípios aos do XP onde os requisitos são pouco estáveis ou desconhecidos tendo suas iterações curtas promovendo visibilidade, podendo ser inspecionado e adaptado. (FAGUNDES, 2005)

ZANATTA destaca, que o principal objetivo do *Scrum* é:

> entregar um software com a maior qualidade possível dentro de séries, compostas por pequenos intervalos de tempo definidos chamados *Sprints*, que tem aproximadamente um mês de duração. (Beedle, 2002).

E FAGUNDES (2005) descreve que de acordo com Schwaber e Beedle (2002), o *Scrum* é um método para gerenciar processo de desenvolvimento de software, porém não requer ou fornece qualquer técnica e ferramenta específica para o desenvolvimento do software, apenas define como as equipes devem se organizar onde os requisitos mudam constantemente.

Nos próximos tópicos, serão apresentadas as práticas do *Scrum*, como se dá o início de um projeto feito com *Scrum*, seus processos e artefatos de acordo com *Scrum For Team System*.

---

[10] *Rugby* é um esporte tendo como objetivo que duas equipes de quinze, dez ou sete jogadores cada uma, jogando de forma leal e de acordo com as Leis e o espírito desportivo, portando, passando, chutando ou apoiando a bola, marquem a maior quantidade de pontos possível. A equipe que marcar mais pontos será a vencedora da partida. E, *SCRUM* é o nome dado a disputa da bola. Disponível em:<http://www.rugbynews.com.br>.

4.1 PREPARAÇÃO

Como as demais metodologias de desenvolvimento de *software*, *Scrum* também possui as fases tradicionais para o desenvolvimento de *software* que inclui a captura dos requisitos, análise, projeto e entrega.

De acordo com Ken Schwaber (*Scrum For Team System*, 2007) no *Scrum*, o processo de "preparação" é dividido em várias etapas e o primeiro passo requer uma análise do tamanho do projeto para poder prever a quantidade de trabalho que será necessário na produção do projeto a fim de organizar uma estrutura e gastar o menor tempo possível garantindo os prazos para o desenvolvimento do *software*.

Scrum, como outras metodologias ágeis, não incentiva a criação de documentação desnecessária do projeto, entretanto, é importante que a equipe inteira compreenda a essência do projeto ou produto que estão construindo. Para isso, chamamos esta etapa na criação de um documento de visão do projeto onde informações breves, porém precisas, estarão acessíveis para consultas se precisar.

Com a visão do produto, inicialmente, se defini uma lista das funcionalidades que são necessários para compor o produto final. Esta lista se chama *Product Backlog* e é o ponto de partida do *Scrum* onde cada funcionalidade recebe uma prioridade.

O *Product Backlog* lista todas as funcionalidades para a solução do produto final. Se todas essas funcionalidades forem desenvolvidas ao mesmo tempo, certamente informações serão desencontradas causando grandes problemas no processo de desenvolvimento do produto acarretando prejuízo e perda de tempo. Por essas razões, no *Scrum* dividi-se o *Product Backlog* (Figura 5) em *releases* onde as funcionalidades de maior prioridade partem para a produção.

Um *release* no *Scrum* é denominado *Sprint*, que são pequenos intervalos de tempo e que possuem aproximadamente um mês de duração para produção das funcionalidades do *Product Backlog*.

Por fim, é interessante nesta fase de preparação, se ter a equipe formada para a execução do projeto. Uma vez tendo a equipe formada, seus papéis de cada

membro são identificados, e se da o pontapé inicial com a primeira reunião da equipe com os seguintes tópicos:

- Escopo do projeto;
- Revisão das funcionalidades com prioridades altas do Backlog;
- Discussões técnicas em geral;
- Acordo inicial de como a equipe trabalhará junto. (*Scrum For Team System*, 2007).

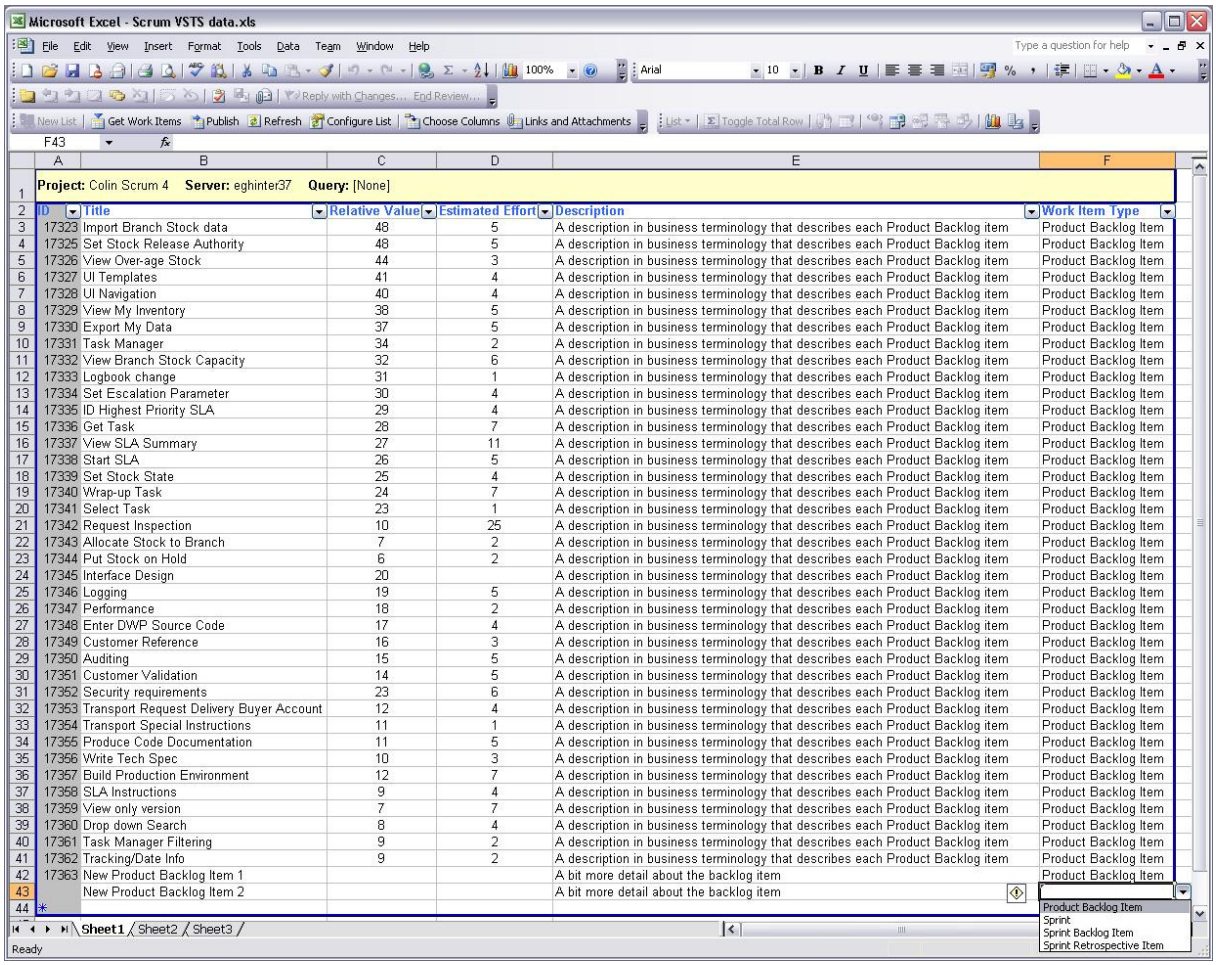


Figura 5 – Exemplo de itens do Product Backlog.
Fonte: http://scrumforteamsystem.com/

4.2 PAPÉIS

O *Scrum* é implementado por diferentes papéis e responsabilidades que desempenham tarefas durante o processo de desenvolvimento do *software*. De acordo com Ken Schwaber (*Scrum For Team System*, 2007), estes papéis são: o *Product Owner*, Team Members, e o *ScrumMaster* como descritos nos itens a seguir.

4.2.1 PRODUCT OWNER

O “Dono do Produto” é a pessoa responsável pelos interesses de todos no projeto, cuida do retorno financeiro do produto ROI (*Return On Investment*) e define as características do produto decidindo a data e o que deve conter em cada *Sprint*.

Product Owner é a pessoa responsável pelo *Product Backlog*. (FAGUNDES, 2005) Ele prioriza as funcionalidades do *Product Backlog* e libera para serem desenvolvidas no *Sprint Backlog*. Ao término de cada *Sprint*, o *Product Owner* pode aceitar ou rejeitar o resultado.

4.2.2 TEAM MEMBERS

O segundo papel na equipe de Scrum é a “equipe do projeto” que é formado por um grupo de no máximo sete pessoas. Grupo onde cada pessoa desempenha um papel específico de acordo com a sua habilidade.

A equipe junto com o *Product Owner* define o que será feito em cada *Sprint* ao final, a equipe mostra o resultado para o *Product Owner* e o mesmo decida o que será feito. Uma vantagem do *Scrum* é que a equipe pode se auto-organizar de forma participativa tendo a liberdade de realizar o que quiser dentro de cada *Sprint* para cumprir o objetivo da iteração. (*Scrum For Team System*, 2007)

4.2.3 SCRUMMASTER

O *ScrumMaster* é a pessoa considerada gerente do projeto e tem a responsabilidade de assegurar que o processo esta sendo desenvolvido de acordo com as práticas e regras que o *Scrum* estabelece.

É trabalho de o ScrumMaster evitar todo o impedimento interno e externo à equipe onde possa prejudicar o objetivo de cada funcionalidade dentro de cada *Sprint*.

De acordo com o site da *Scrum For Team System*, o *ScrumMaster* é um líder, um facilitador onde:

- Procura melhorar e facilitar a produtividade da equipe de desenvolvimento;
- Permite a cooperação próxima entre a equipe através de todos os papéis e funções removendo as barreiras indesejáveis;
- Protege a equipe de interferências externas;
- Assegura que o processo esteja em evolução;
- Interagem com as pessoas nas reuniões diárias, revisando e planejando cada *Sprint* e o andamento do processo;
- Remove as barreiras existentes entre a equipe de desenvolvimento e o cliente de modo que o cliente diga quais funcionalidades ele deseja para o seu produto;
- Por fim, procura melhorar as funcionalidades utilizando ferramentas e práticas de engenharia.

4.3 PROCESSO

O processo do *Scrum* consiste em um ciclo regular onde o foco é ter ao final de cada *Sprint* uma versão do produto desenvolvido com qualidade, ou seja, funcionalidades bem desenvolvidas para apresentação ao cliente.

Nesta etapa, são utilizadas as práticas de reuniões diárias, *Sprint*, Planejamento do *Sprint*, revisão e retrospectiva do *Sprint*.

No começo de cada *Sprint*, o *Product Owner* define as prioridades permitindo que a equipe selecione as funcionalidades com as prioridades mais importantes para definir as tarefas a serem implementadas na reunião de planejamento do *Sprint* formando o *Sprint Backlog*.

Ao final de cada *Sprint*, uma nova versão do produto é criada e entregue ao *Product Owner* e outras partes interessadas como o Cliente para a revisão do *Sprint*. Os participantes avaliam a versão do produto e decidem sobre as novas atividades seguintes de acordo com as prioridades do *Product Backlog*.

Antes de definir o *Sprint* seguinte, a equipe pode discutir seu desempenho a fim de fazer melhorias aos que trabalham no *Sprint*. A equipe seguirá este ciclo até que todas as funcionalidades estejam desenvolvidas e que o produto esteja pronto.

4.3.1 PLANEJANDO O SPRINT

A finalidade de cada *Sprint* é tornar as funcionalidades definidas no *Product Backlog* em incrementos para a versão final do produto. O *Product Owner*, o *ScrumMaster* e a equipe antes de cada *Sprint* definem que funcionalidades a equipe trabalhará. O *Product Owner* apresenta o *Product Backlog* com as prioridades e a equipe seleciona o que acredita que pode ser construído no *Sprint*.

A reunião de planejamento do *Sprint* consiste em duas partes podendo durar até 4 horas. Na primeira parte conhecida como *Sprint Planning 1*, o *Product Owner* apresenta a equipe de desenvolvimento, a lista de prioridades do *Product Backlog*. A equipe de desenvolvimento discute quais funcionalidades podem ser desenvolvidas até o próximo *Sprint* e seleciona a funcionalidades para o *Sprint* em planejamento. (*Scrum For Team System*, 2007).

Na segunda parte *Sprint Planning 2*, é planejado a carga de trabalho do *Sprint*. A equipe define a arquitetura das funcionalidades que foram selecionadas e define então as tarefas e divisão de trabalho para estas tarefas serem desenvolvidas até o *Sprint* seguinte. (*Scrum For Team System*, 2007)

A Figura 7 demonstra uma melhor visualização do processo de planejamento do *Sprint*.

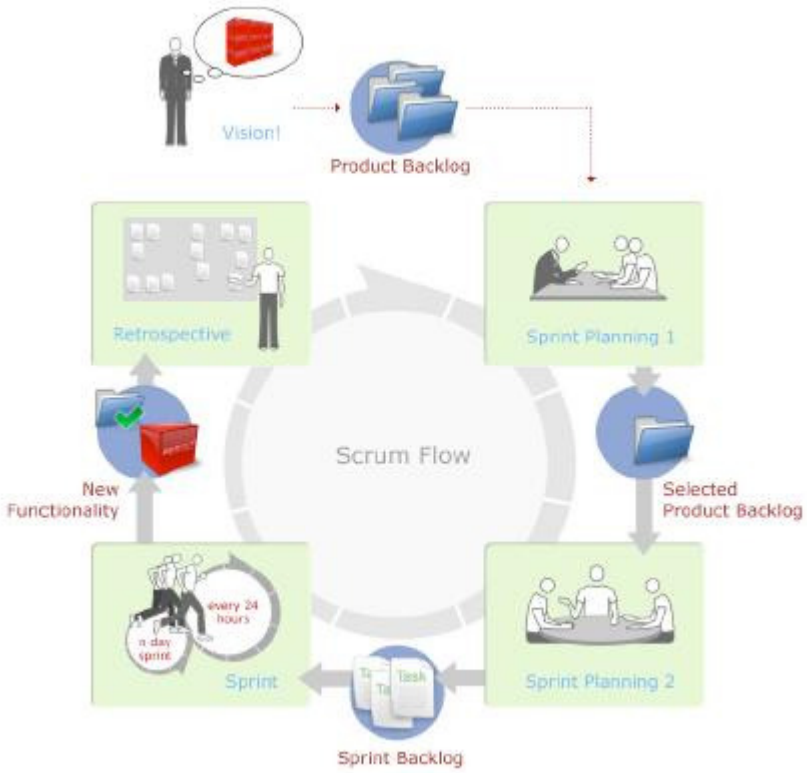


Figura 6 – Visão geral do processo de planejamento do Sprint.
Fonte: http://www.glogerconsulting.de/

De acordo com a reunião de planejamento do *Sprint*, podem-se destacar alguns aspectos chaves:

- O *Product Owner*, *ScrumMaster* e a equipe determinam as funcionalidades que pode ser desenvolvido para o próximo *Sprint*;
- As funcionalidades são selecionadas do alto da lista de acordo com a sua prioridade pela equipe;
- A equipe obtém todos os detalhes necessários com o *Product Owner* e das partes interessadas do produto (cliente ou representante do cliente) para que possa estimar o que é viável para o próximo *Sprint*;

- O *Product Owner* e a equipe estabelecem um objetivo para o *Sprint*;
- Deve ser escolhido apenas às funcionalidades que de fato serão implementadas no *Sprint*;
- As funcionalidades escolhidas são quebradas em tarefas do *Sprint Backlog*;
- As estimativas da equipe são estabelecidas de acordo com os *Sprints* anteriores e com a complexidade das tarefas selecionadas para o *Sprint*;
- Caso for detectado que o *Sprint* vai consumir mais que 20% de trabalho da equipe do que foi planejado após o segundo dia da reunião de planejamento é sinal que necessita fazer um planejamento melhor. (*Scrum For Team System*, 2007).

4.3.2 SPRINT

Considerado a principal fase do *Scrum*, *Sprint* é um período de tempo reservado para produção das funcionalidades do produto de acordo com o *Product Backlog*. Cada período é composto por aproximadamente trinta dias, ou seja, um mês ou vinte dias de trabalho.

No *Scrum* não existem processos pré-definidos, mas reuniões diárias conhecidas como *Daily Scrum* onde são coordenadas as tarefas e planejadas para realização no *Sprint*.

Em cada *Sprint*, acontece a integração com as partes já desenvolvidas onde são feitos testes garantido um progresso real das atividades. E ao final de cada *Sprint*, uma versão do produto é gerada e uma pequena demonstração é criada para ilustrar ao cliente o que está sendo desenvolvido afirma Beedle (2001).

Com as funcionalidades desenvolvidas ao final do *Sprint* com a versão do produto gerada, tem-se a tarefa da reorganização do *Product Backlog* a fim de se inicializar um novo *Sprint*. Na Figura 6, ilustra a fase de desenvolvimento do Scrum

onde as funcionalidades são apresentadas no *Product Backlog*, e desenvolvidas no *Sprint* gerando uma versão do produto no final do *Sprint*.

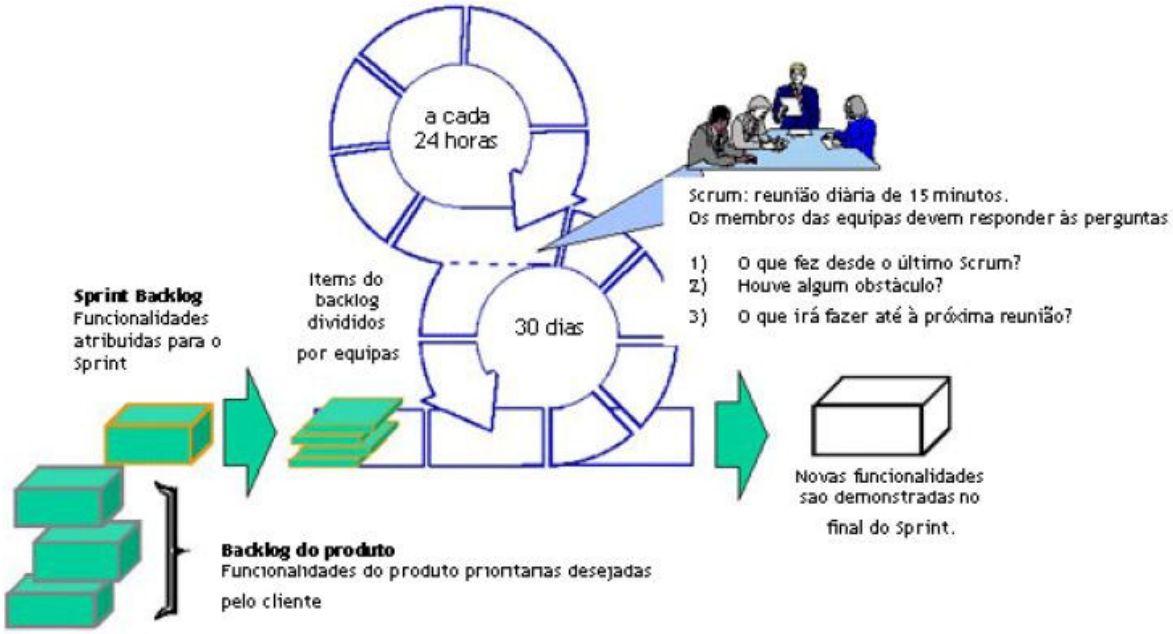


Figura 7 – Processo completo para o Sprint no Scrum.
Fonte: SCRUM Um Modelo Ágil para Gestão de Projetos de Software.

4.3.3 REUNIÃO DIÁRIA DO SCRUM

É uma reunião curta de aproximadamente quinze minutos definido por cada equipe. Durante esta reunião, os membros da equipe sincronizam o progresso de seus trabalhos e relatam suas dificuldades ao *ScrumMaster* para uma possível solução. Schwaber apresenta algumas dificuldades que a equipe pode enfrentar durante o *Sprint* como: “A rede de computadores ou os servidores estão lentos”, ou “não sei proceder em determinada situação”.

Uma prática rotineira ao *ScrumMaster* durante a reunião é levantar três questões para cada membro da equipe destaca Schwaber em *Scrum For Team System*.

1. O que você fez desde o último encontro?
2. Quais foram as suas dificuldades durante o trabalho?

3. Quais atividades você pretende fazer até o próximo encontro?

Diante destas questões, o *ScrumMaster* registra as resposta de cada membro e posiciona cada membro de acordo com suas respostas até o próximo encontro a fim de proporcionar uma maior progresso para a equipe.

Esta reunião apesar de serem curta, é extremamente útil e Schwaber destaca alguns benefícios:

- Problemas identificados são apresentados a gerência para uma possível solução assim que ocorrerem mantendo o desenvolvimento em dia;
- Evitar duplicação de trabalho;
- Melhor compreensão sobre as interdependências entre membros da equipe;
- Trabalho em equipe;
- Produção dinâmica entre a equipe;
- Produção mais eficaz com a pressão de estar à frente da equipe e dizer o que você tem feito. (*Scrum For Team System*, 2007).

4.3.4 REVISÃO DO SPRINT

A revisão do *Sprint* vai indicar o progresso das funcionalidades desenvolvidas ao final de cada *Sprint*. E de acordo com este progresso, podem ser feita adaptações ao projeto. Após a execução do *Sprint*, a equipe apresenta o incremento que foi implementado.

O *Product Owner*, a gerência, e o cliente avaliam o versão que foi implementado e escutam os fatos que a equipe tem para dizer sobre a trajetória do *Sprint* como problemas e acertos que ocorreram. Como os fatos contados pela equipe, o *Product Owner* pode fazer uma decisão informal do que fazer em seguida. Ou seja, se a equipe pode avançar para o próximo *Sprint*.

Durante a reunião, todos visualizam as funcionalidades que foram implementadas durante o *Sprint* e decidem que funcionalidades podem ser adicionadas ao próximo *Sprint*. Caso os incrementos implementados gerarem novas idéias, essas idéias são adicionadas ao *Product Backlog*.

Scrum For Team System apresenta algumas regras da revisão do *Sprint*:

- Procurar não ultrapassar uma hora na reunião de revisão do *Sprint*;
- As funcionalidades que não foram implementadas não podem ser apresentadas;
- Os artefatos que não são funcionalidades também não podem ser apresentados exceto quando são usadas para compreender o produto;
- Procura apresentar as funcionalidades implementadas no próprio ambiente de trabalho ou no ambiente mais próximo à produção;
- A revisão do *Sprint* começa com um membro da equipe apresentando os objetivos do *Sprint*, mostrando as funcionalidades selecionadas do *Product Backlog* que foram implementadas e as que não foram implementadas. Os membros fora da equipe discutem os pontos positivos e negativos do *Sprint*;
- A maioria do tempo com a revisão do *Sprint* são gastos com os membros da equipe apresentando as funcionalidades, respondendo perguntas a respeito das funcionalidades da apresentação, e anotando mudanças;
- Ao final da apresentação, as partes interessadas do produto avaliam as mudanças desejadas e indicam as prioridades destas mudanças;
- As partes interessadas do produto comentam, fazem observação e/ou criticam as funcionalidades implementadas do produto da apresentação;

- O Dono do Produto podem identificar as funcionalidades que não foram entregues de acordo como o esperado e pedir que seja colocado novamente ao *Product Backlog*;
- As partes interessadas do produto podem identificar as funcionalidades novas enquanto vêem a apresentação e pedir que a funcionalidade seja adicionada ao *Product Backlog*;
- O *ScrumMaster* é quem determina quem vai assistir a reunião de revisão do *Sprint* e acomodar seus lugares para inicio da reunião;
- No fim da reunião de revisão do *Sprint*, o *ScrumMaster* anuncia o lugar e a data da próxima revisão do *Sprint* ao Dono do Produto e as partes interessadas. (*Scrum For Team System*, 2007).

Após a reunião de revisão do *Sprint*, algumas conseqüências podem ser levantadas:

- Restaurar as funcionalidades que não foram implementadas do *Product Backlog* e priorizá-las;
- Remover as funcionalidades do *Product Backlog* que já foram implementadas;
- Trabalhar com o *ScrumMaster* para reformular a equipe se necessário; (*Scrum For Team System*, 2007).

4.3.5 RETROSPECTIVA DO SPRINT

A retrospectiva do *Sprint* é uma reunião feita pelo *ScrumMaster* onde a equipe discute o *Sprint* que foi concluído e determina o que poderia ser mudado de modo a tornar o próximo *Sprint* mais agradável e produtivo.

Uma coisa que pode ser tratado na reunião de retrospectiva do *Sprint* são as práticas, a comunicação, o ambiente, os artefatos e as ferramentas que são utilizados durante cada *Sprint*.

Scrum não é uma metodologia fixa, mas tem de preferências uma estrutura que deve ser adaptada apropriadamente de acordo com cada projeto, equipe e circunstâncias especificas. E a retrospectiva do *Sprint* é um mecanismo importante permitindo que a equipe evolua e melhore continuamente durante o desenvolvimento do projeto.

A retrospectiva do *Sprint* conta com alguns elementos chaves que se destacam como:

- As melhorias dos processos são feitas no fim de cada *Sprint* assegurando que a equipe do projeto esteja sempre melhorando a maneira que trabalha;
- A retrospectiva é um processo colaborativo entre todos os membros da equipe, o Dono do Produto e o *ScrumMaster*;
- Todos os membros da equipe identificam o que foi bem e o que pode ser melhorado;
- O *ScrumMaster* dá prioridades as ações e as lições aprendidas baseadas na equipe;
- A equipe planeja a solução para os problemas irritantes; (*Scrum For Team System*, 2007).

4.4 ARTEFATOS

O *Scrum* oferece uma série de artefatos ou ferramentas que auxiliam o processo de desenvolvimento a andar na trilha (*Scrum For Team System*, 2007). A seguir veremos com mais detalhes cada um deles.

4.4.1 PRODUCT BACKLOG

O *Product Backlog* é o coração do *Scrum*, é onde tudo começa (KNIBERB, 2007). Trata-se de uma lista priorizada e estimada das necessidades do projeto. A prioridade deve ser atribuída levando em consideração o ROI (*Return On*

Investment) de cada tarefa, e a lista de prioridades deve ser alterada conforme as alterações na análise do negócio e mudanças de tecnologia.

Os itens do *Product Backlog* podem ser requisitos tanto funcionais como não-funcionais. A precisão das estimativas depende diretamente do nível de detalhamento e prioridade de cada item.

Toda idéia ou requisição podem ser adicionadas a qualquer momento no P*roduct Backlog*, desde o mais importante requisito funcional a um detalhe relativo à tecnologia. Isto faz com que este artefato seja muito útil para acalmar os clientes do negócio, e além de que pode agregar mais valor ao projeto.

4.4.1.1 Principais características

Como características importantes do *Product Backlog*, pode-se citar:

- Lista das funcionalidades e tecnologias;
- Emergente, priorizado e estimado;
- Maiores detalhes em itens mais prioritários;
- Uma lista para múltiplas equipes;
- *Product Owner* responsável pelo ajuste das prioridades;
- Qualquer um pode contribuir com os itens;
- Visivelmente mantido e editado;
- Derivado dos requisitos de negócio. (*Scrum For Team System*, 2007)

Outra característica importante deste artefato é a sua refatoração. A cada *Sprint* devem ser reavaliados 20% dos itens com maior prioridade da lista.

4.4.1.2 Campos do *Product Backlog*

Esta lista priorizada dos requisitos do projeto é muito flexível e podemos ter diversas informações contidas nela. Abaixo alguns campos que podemos ter no *Product Backlog*.

- ID: Uma identificação única, auto-incrementada, para organizar melhor os itens e não se perder entre eles caso seus nomes sejam alterados durante o projeto.
- Nome: Um nome curto, mas que identifique com poucas palavras este requisito no projeto.
- Descrição: Uma descrição mais na parte de negócio do item.
- Importância: É avaliação de importância da tarefa pelo *Product Owner*. Por exemplo, entre zero e 100, onde quanto maior mais importante.
- Estimativa Inicial: Avaliação da equipe de quanto trabalho será necessário para a execução de cada tarefa. Esta estimativa é medida em *"ideal man-days", ou seja, homens necessários para a execução da tarefa em um dia. Por exemplo, uma tarefa com estimativa de 10 para ser realizada por dois homens levará cinco dias.*
- Observação: Alguma outra informação relevante, referências, esclarecimentos, etc.

Um exemplo de um *Product Backlog* pode-se ver na Figura 5 na página 58.

4.4.2 SPRINT BACKLOG

O *Sprint Backlog* é uma lista de tarefas que define o trabalho da equipe a cada *Sprint*. A lista surge durante o *Sprint Planning*. As tarefas do *Sprint Backlog* são o que a equipe definiu como sendo necessário para a fluência da realização dos itens do *Product Backlog* nas funcionalidades do sistema. Cada tarefa é identificada pelo seu responsável e a sua quantidade estimada de trabalho restante (*Scrum For Team System*, 2007).

Dentre as principais características do *Sprint Backlog* tem-se:

- Contém as tarefas do *Product Backlog*;

- Suas tarefas devem atender as exigências dos itens do *Product Backlog* selecionados para o *Sprint* atual;
- As tarefas são estimadas em horas, geralmente entre 1 e 16;
- As tarefas com mais de 16 horas são quebradas no *Sprint Planning*, ou durante a execução do *Sprint*;
- Qualquer membro da equipe pode adicionar apagar ou alterar as tarefas do *Sprint Backlog*;
- Se a tarefa for pesada definir uma quantidade de tempo estimada maior, e depois se necessário quebrá-la para baixo;
- As estimativas das tarefas são atualizadas conforme se conhece as características do projeto na execução de outras tarefas;
- Somente a equipe pode adicionar itens ao *Sprint Backlog*.

Uma maneira utilizada pra controlar as tarefas do *Sprint Backlog* é um quadro de cartolina com quatro colunas conforme a figura 8 abaixo:

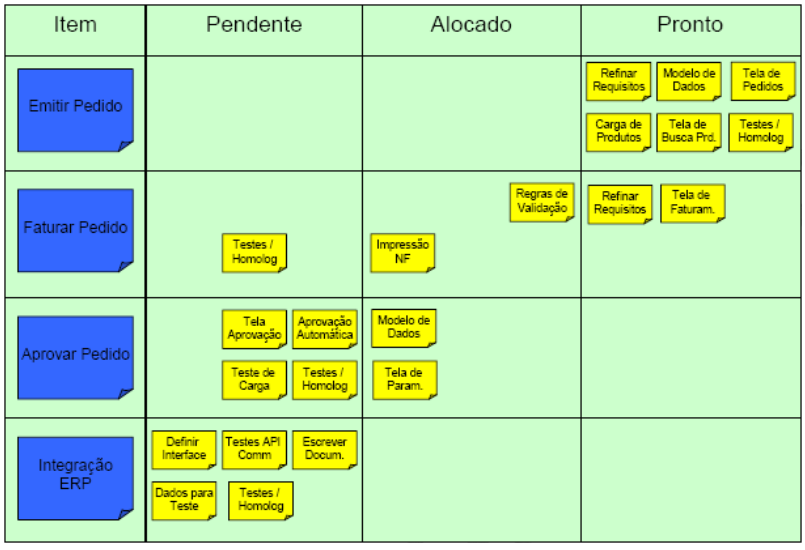


Figura 8 – *Sprint Backlog*
Fonte: Rodrigo Yoshima. Gerenciamento de Projetos com Scrum.

4.4.3 SPRINT BURNDOWN

O *Sprint Burndown* dá a equipe uma indicação diária da velocidade e da progressão do trabalho inacabado do *Sprint* atual. Em uma dimensão temos as tarefas do *Sprint Backlog* e na outra temos os dias deste *Sprint*. Abaixo temos uma imagem que mostra um *Sprint Burndown* seguindo para ter todas as tarefas terminadas dentro do *Sprint*.

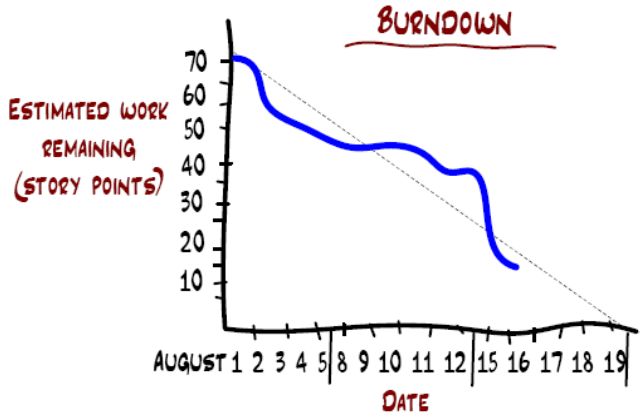


Figura 9 – *Sprint Burndown* (No prazo.).
Fonte: KNIBERB, 2007.

Através do gráfico do *Sprint Burndown* pode-se analisar se o *Sprint* está atrasado, por exemplo. Na Figura 8 temos um gráfico onde a linha real está acima da linha de estimativa para terminar no prazo, caracterizando um atraso nas tarefas. Diante de uma situação desta, alguma atitude deverá ser tomada no projeto, como retirar tarefas do *Sprint*.

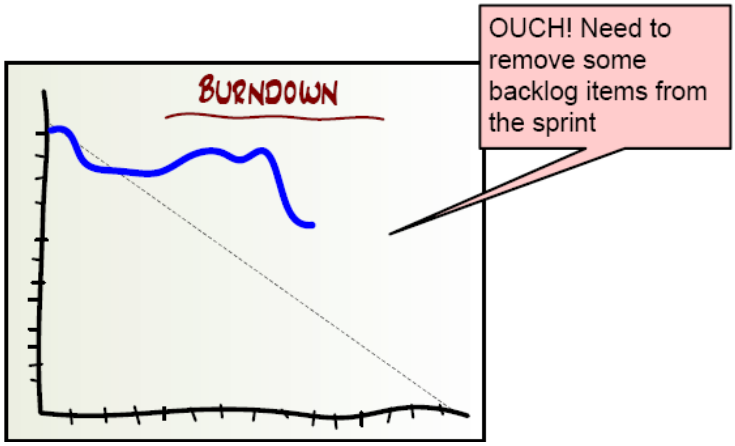


Figura 10 – *Sprint Burndown* (Atrasado).
Fonte: KNIBERB, 2007.

Outra situação possível é o *Sprint* estar adiantado, neste caso uma atitude a ser tomada é a inserção de novas tarefas do *Product Backlog* ao *Sprint* atual. Esta situação está caracterizada na Figura 9 abaixo:

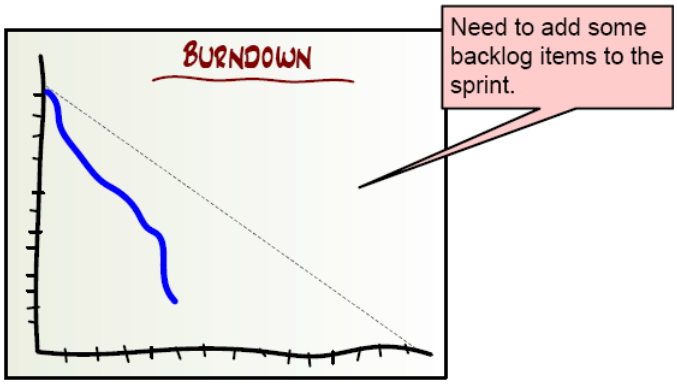


Figura 11 – *Sprint Burndown* (Adiantado).
Fonte: KNIBERB, 2007.

4.4.4 RELATÓRIOS

Os relatórios do *Scrum* estão diretamente ligados com o fato de a equipe ser auto-sustentável, dando base para a equipe analisar durante a execução de um *Sprint* pontos como: Progresso para liberação, análise de produtividade, mudanças de prioridades e estimativas, entre outros (*Scrum For Team System*, 2007).

Abaixo citaremos alguns destes relatórios, outros relatórios e mais detalhes pode-se encontrar no site da *Scrum For Team System*: http://scrumforteamsystem.com/ProcessGuidance/Artefacts/Reports.html.

- ***Sprint Overview***: Neste relatório temos as tarefas, erros e impedimentos do *Sprint*. Pode-se também gerá-lo filtrando por membros da equipe.

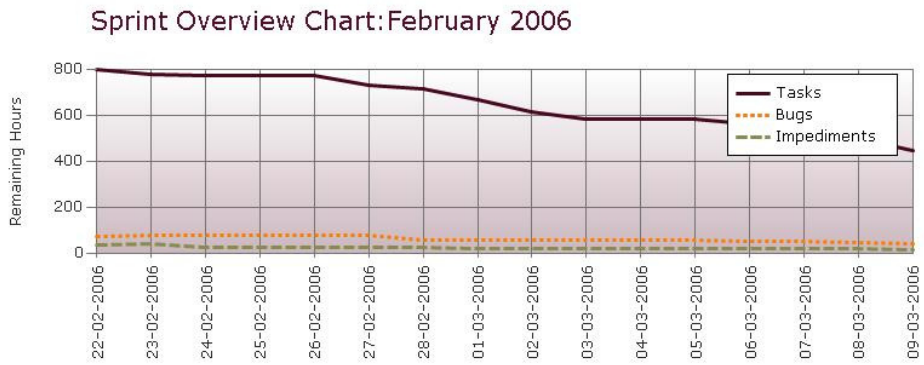


Figura 12 – *Sprint Overview*.
Fonte: Scrum For Team System, 2007.

- ***Product Backlog Composition***: Este relatório apresente os itens do *Product Backlog* alocados para cada *Sprint*, e dentro de cada *Sprint* temos a composição destes itens em tarefas do *Sprint Backlog*.

Product & Sprint Backlog Items for Project Mark016

Id	Sprint Name	Product Backlog Item Name	Sprint Backlog Item Name	Estimated Effort	Work Remaining
4501	December 2005				
5171		PBI 48		33	0
4801			SBI 297		0
5172		PBI 49		66	0
4802			SBI 298		0
4803			SBI 299		0
4804			SBI 300		0
4805			SBI 301		0
4806			SBI 302		0
4807			SBI 303		0
4808			SBI 304		0
4809			SBI 305		0
5173		PBI 50		45	0
4810			SBI 306		0
4811			SBI 307		0
4812			SBI 308		0
4813			SBI 309		0
4814			SBI 310		0
5174		PBI 51		55	5
4815			SBI 311		0
4816			SBI 312		0
4817			SBI 313		5
4818			SBI 314		0
4819			SBI 315		0
4820			SBI 316		0
4821			SBI 317		0
5175		PBI 52		36	0
4822			SBI 318		0

Figura 13 – *Product Backlog Compositon*.

Fonte: *Scrum For Team System*, 2007.

5 PROPOSTA

A proposta do projeto é a realização de um estudo comparando entre as duas metodologias ágeis abordadas neste trabalho.

5.1 INTRODUÇÃO

Utilizando as metodologias ágeis *YP* e *Scrum* será realizada uma análise com cada metodologia para o desenvolvimento inicial de um componente de *software* simples aplicado ao setor de farmácia de postos municipais de saúde.

Após este processo será possível avaliar com maior propriedade as metodologias estudas verificando os pontos fortes e fracos de cada uma.

Como resultado da aplicação da metodologia teremos o inicio do desenvolvimento de um componente de software para farmácias de postos de saúde. Este que será disponibilizado no repositório do projeto Via Digital.

5.2 DESCRIÇÃO DO COMPONENTE

A região da Grande Florianópolis apresentou, nas últimas décadas, transformações significativas na sua estrutura e dinâmica populacional, formando o fenômeno de conurbação[11] entre a capital e três outros municípios vizinhos: São José, Palhoça e Biguaçú. O fenômeno da conurbação designa uma extensa área urbana que surge espontaneamente da junção de duas ou mais cidades.

Na medida em que a urbanização se acelera, ficam estabelecidos numerosos desafios aos governantes e aos próprios urbanistas, visto que a vida nestas cidades passa a existir de modo integrado e seus habitantes passam a fazer uso, por exemplo, dos serviços de saúde da cidade onde moram ou da cidade onde trabalham.

O conceito de Assistência Farmacêutica evoluiu no tempo e o desenvolvimento do trabalho em farmácia passou por diferentes fases. No município

de Florianópolis, a história da Assistência Farmacêutica inicia em 1982, quando pela primeira vez, uma profissional farmacêutica assumiu a responsabilidade técnica perante o Conselho Regional de Farmácia, sobre as ações que envolviam recebimento, armazenamento, estocagem, distribuição e controle de medicamentos junto à Secretaria de Saúde. Até então tudo era feito por auxiliares leigos (LUNA, 2006).

Ainda segundo Luna (2006), em Florianópolis, entre 1985 e 1991, a Secretaria de Saúde contou com quatro profissionais farmacêuticos, que lotados no nível central elaboravam programação anual de necessidades, efetuavam reposição de estoques das unidades. Com o crescimento da rede e da demanda, da instalação de unidades que necessitam de medicamentos controlados como os CAPSs - Centros de Atendimento Psico-Social e até mesmo da expansão de unidade maiores do tipo Pronto-atendimento, fica inviável manter um controle manualmente (LUNA, 2006).

Diante desta carência de recursos que os profissionais farmacêuticos dispõem, surge a idéia da construção de um componente de *software* a fim de solucionar tal carência.

O *SysFarma* servirá como um controle de estoque da farmácia das unidades de saúde para ajudar no processo de reposição de estoques dos medicamentos e um planejamento melhor a fim de evitar erros. O sistema ainda contará com um controle de usuários que adquirem seu medicamento. Efetuar pesquisas por princípio ativo ou sua descrição e gerar relatórios.

Com o *SysFarma*, a farmácia terá uma lista fiel de seus medicamentos disponíveis, quantos medicamentos estão em estoque. Para manipular o sistema, ele contará com sistema de *login* e senha sendo apenas pessoas autorizadas a manipular o sistema.

---

[11] Conjunto urbano constituído por uma cidade principal e os seus subúrbios. Fonte: Priberam.

5.3 PRINCIPAIS PROCESSOS

Um dos grandes benefícios que o Sistema Único de Saúde (SUS) tem para a população brasileira é a assistência farmacêutica. Com ela, pessoas que tem dificuldades de adquirir seu medicamento em estabelecimentos privados podem se beneficiar adquirindo seus medicamentos que necessitam.

O acesso ao benefício é garantido mediante apresentação de receituário médico ou odontológico, prescrito de acordo com a legislação vigente contendo um ou mais medicamentos na receita.

Para cada medicamento que é entregue, é feito um registro de baixa de estoque do produto. Ao final do mês tem-se o controle, de estoque atualizado somando o estoque anterior com a entrada do mês subtraindo a saída do mês.

Tendo em vista que todo este processo é feito manualmente pelos agentes de saúde do posto torna-se difícil o controle, sendo os erros quase que inevitáveis.

Tendo um controle de estoque dos medicamentos da farmácia, tornará mais fácil o controle de medicamentos. Sendo possível, por exemplo, ter relatórios precisos de quantos medicamentos possui a farmácia.

Para isso, o sistema contará com um cadastro de medicamentos tendo a quantidade em que a farmácia possui. O sistema também poderá fazer um controle de pedido de medicamentos e um cadastro de usuários que adquiriram os medicamentos da farmácia.

5.4 MODELAGEM

Com cada uma das duas metodologias abordadas neste trabalho foi realizada a modelagem do componente proposto. Estas modelagens servirão de base para que possamos avaliar com maior propriedade as características de cada uma delas.

Primeiramente foi realizada esta análise utilizando a metodologia YP, que pode ser visualizada integralmente no Anexo A. Percebemos que esta metodologia apresenta seus processos bem definidos e delimitados, apresentando o que fazer e como fazer. Ou seja, para cada etapa do processo é definido um documento para a sua realização.

Partindo da premissa que esta metodologia tem um fim acadêmico, estas características são perfeitamente compreendidas. Outro ponto relevante que foi encontrado é a existência de artefatos que não agregam valor ao processo de modelagem do sistema. Como por exemplo, o modelo de tarefas cuja função é esclarecer os detalhes de cada tarefa do sistema, porém esta função pode ser obtida em conversa com o cliente ou com a construção do protótipo de interface. Na modelagem do SysFarma este artefato não foi de nenhuma valia, somente tomou um considerável tempo para sua criação.

Posteriormente foi realizada a modelagem utilizando a metodologia *Scrum*, que pode ser visualizada no Anexo B. Observou-se que esta metodologia se preocupa mais no processo em si, e não em documentos e modelos pré-definidos. Ao contrário da metodologia YP, o Scrum não apresenta uma característica didática forte, pois não apresenta processos pré-definidos e documentos detalhados de como cada tarefa deve ser realizada.

As empresas de desenvolvimento de software têm uma forte dificuldade em adotar algum processo de desenvolvimento, pois geralmente as pessoas têm uma grande barreira às mudanças. Com o *Scrum* esta dificuldade é amenizada, pois ele não fornece qualquer método específico para o desenvolvimento de software, apenas estabelece um conjunto de regras e práticas gerenciais.

Com esta característica as empresas podem adequar o seu processo atual, documentos utilizados e ferramentas, com o processo de desenvolvimento com *Scrum*, melhorando sua gerência e organização.

5.5 COMPARAÇÃO DOS MÉTODOS ÁGEIS YP E SCRUM

Agora, será realizada uma análise comparativa entre as metodologias estudas neste trabalho. São elas: *easYProcess* e *Scrum*. A realização da análise tem como base a satisfação dos princípios ágeis classificando cada prática de acordo com os critérios definidos na tabela X.

Classificação		**Critério**
NS	Não Satisfeito	Pouca evidência de que o principio foi satisfeito ou o método propõe uma proposta diferente ao manifesto ágil.
PS	Parcialmente Satisfeito	Existem evidências de satisfação ao principio ágil de modo sistemático.
S	Satisfeito	Existe satisfação no princípio existindo evidências significativas no método.

Tabela 3 – Critério para análise comparativa entre as metodologias

Os princípios ágeis são a base do pensamento ágil servindo como um guia para todos os métodos ágeis. Eles auxiliam na definição do quão ágil é o método, baseado, principalmente, na forma em que são usados pelos métodos. (ZANATTA, 2004)

A seguir serão apresentadas as justificativas juntamente com suas notas atribuídas às metodologias estudadas quando comparados aos princípios ágeis.

De acordo com o princípio 1: A prioridade é satisfazer ao cliente através de entregas de software de valor contínuo e freqüentes. A tabela 4 apresenta a justificativa em relação às metodologias *YP* e *Scrum*.

Metodologia	**Categoria**	**Justificativa**
easYProcess	S	Como é uma metodologia voltada para o meio acadêmico, o processo de desenvolvimento ocorre no máximo em duas

		semanas satisfazendo a o “cliente” neste caso o professor.
Scrum	S	De acordo com seus *Sprint* de duas a quatro semanas, as funcionalidades são desenvolvidas e apresentadas para o cliente visando à satisfação ao cliente.

Legenda: NS – Não Satisfeito, PS – Parcialmente Satisfeito, S – Satisfeito.

Tabela 4 – Justificativa princípio 1

A tabela 5 apresenta a justificativa em relação às metodologias com o princípio 2: Receber bem as mudanças de requisito, mesmo em uma fase avançada, dando aos clientes vantagens competitivas.

Metodologia	**Categoria**	**Justificativa**
easYProcess	PS	Tanto os requisitos funcionais como os não-funcionais são decididos no início do projeto durante a conversa com o cliente. Caso surge novos requisitos, eles podem ser adicionados ao projeto sendo necessário reavaliar todos os planos de iteração.
Scrum	S	Como ao final de cada *Sprint* são mostradas as funcionalidades ao cliente, no decorrer do projeto, sempre novas funcionalidades vão surgindo podendo ser adicionadas ao *Product Backlog*.

Legenda: NS – Não Satisfeito, PS – Parcialmente Satisfeito, S – Satisfeito.

Tabela 5 – Justificativa princípio 2

A tabela 6 apresenta a justificativa em relação ao princípio 3: Entregar software em funcionamento com freqüência de algumas semanas ou meses, sempre na menor escala de tempo.

Metodologia	**Categoria**	**Justificativa**
easYProcess	S	Como o tempo de desenvolvimento de cada release acontece em 4 semanas, freqüentemente novas versões são liberadas.

Scrum	S	Satisfaz de acordo com os *Sprint*, que varia entre 2 a 4 semanas.

Legenda: NS – Não Satisfeito, PS – Parcialmente Satisfeito, S – Satisfeito.

Tabela 6 – Justificativa princípio 3

A tabela 7 apresenta a justificativa em relação ao princípio 4: Equipes de negócio e de desenvolvimento devem trabalhar juntas diariamente durante todo o projeto.

Metodologia	**Categoria**	**Justificativa**
easYProcess	PS	Com as reuniões de acompanhamento somente ao final do período de implementação, é parcialmente satisfatória o acompanhamento entre as pessoas ligadas ao projeto com os desenvolvedores.
Scrum	S	A integração entre o cliente e a equipe é estimulada durante todo o projeto.

Legenda: NS – Não Satisfeito, PS – Parcialmente Satisfeito, S – Satisfeito.

Tabela 7 – Justificativa princípio 4

A tabela 8 apresenta a justificativa em relação ao princípio 5: Manter uma equipe motivada fornecendo ambiente, apoio e confiança necessária para a realização do trabalho.

Metodologia	**Categoria**	**Justificativa**
easYProcess	S	A comunicação entre a equipe serve de motivação, favorecendo o ambiente integrado onde se concentra a equipe.
Scrum	S	Como o gerente dá total liberdade para os membros da equipe na realização dos processos além de ter total apoio, este princípio satisfaz este princípio.

Legenda: NS – Não Satisfeito, PS – Parcialmente Satisfeito, S – Satisfeito.

Tabela 8 – Justificativa princípio 5

A tabela 9 apresenta a justificativa em relação ao princípio 6: A maneira mais eficiente da informação circular dentro da equipe é a através de uma conversa face a face.

Metodologia	**Categoria**	**Justificativa**
easYProcess	S	O trabalho no mesmo ambiente e reuniões de acompanhamento satisfazem o princípio.
Scrum	S	Sendo um dos maiores valores à comunicação entre a equipe e o gerente do projeto durante as reuniões diárias do *Sprint* satisfaz o princípio.

Legenda: NS – Não Satisfeito, PS – Parcialmente Satisfeito, S – Satisfeito.

Tabela 9 – Justificativa princípio 6

A tabela 10 apresenta uma justificativa das categorias atribuídas aos métodos ágeis quando comparados ao Princípio 7: Ter o software funcionando é a melhor medida de progresso.

Metodologia	**Categoria**	**Justificativa**
easYProcess	S	Ao final de cada iteração são realizados os testes de aceitação e liberada uma nova versão para o cliente.
Scrum	S	Após a revisão de cada *Sprint*, as funcionalidades (versão para o cliente) são liberadas com aprovação do *Product Owner*.

Legenda: NS – Não Satisfeito, PS – Parcialmente Satisfeito, S – Satisfeito.

Tabela 10 – Justificativa princípio 7

A tabela 11 apresenta uma justificativa das categorias atribuídas aos métodos ágeis quando comparados ao Princípio 8: Processos ágeis promovem o

desenvolvimento sustentável. Todos envolvidos devem ser capazes de manter um ritmo de desenvolvimento constante.

Metodologia	Categoria	Justificativa
easYProcess	S	Projetos simples e mudanças incrementais auxiliam o desenvolvimento a manter um ritmo sustentável.
Scrum	S	Com as reuniões diárias a equipe promove um desenvolvimento sustentável, onde impedimentos e sucessos são compartilhados.

Legenda: NS – Não Satisfeito, PS – Parcialmente Satisfeito, S – Satisfeito.

Tabela 11 – Justificativa princípio 8

A tabela 12 apresenta uma justificativa das categorias atribuídas aos métodos ágeis quando comparados ao Princípio 9: Atenção contínua a excelência técnica e um bom projeto aumentam a agilidade.

Metodologia	Categoria	Justificativa
easYProcess	S	Propôs boas práticas de codificação utilizando técnicas como refatoramento e Design simples.
Scrum	PS	Não possui uma especificação direta em relação a este princípio porém pode ser aplicado de acordo com a experiência da equipe.

Legenda: NS – Não Satisfeito, PS – Parcialmente Satisfeito, S – Satisfeito.

Tabela 12 – Justificativa princípio 9

A tabela 13 apresenta uma justificativa das categorias atribuídas aos métodos ágeis quando comparados ao Princípio 10: Simplicidade é essencial.

Metodologia	Categoria	Justificativa

easYProcess	S	Com práticas, papéis e planejamento bem definidas, facilitam a organização maximizando a quantidade de trabalho e a simplicidade.
Scrum	S	A simplicidade é um dos pontos chaves da metodologia *Scrum* onde procura trabalhar constantemente com soluções simples.

Legenda: NS – Não Satisfeito, PS – Parcialmente Satisfeito, S – Satisfeito.

Tabela 13 – Justificativa princípio 10

A tabela 14 apresenta uma justificativa das categorias atribuídas aos métodos ágeis quando comparados ao Princípio 11: As melhores arquiteturas, requisitos e projetos provêm de equipes organizadas.

Metodologia	**Categoria**	**Justificativa**
easYProcess	S	Equipes auto-organizadas, arquiteturas e requisitos são práticas desta metodologia.
Scrum	S	Sendo as equipes auto-organizadas de acordo com seus papéis e responsabilidades, *Scrum* satisfaz a este princípio.

Legenda: NS – Não Satisfeito, OS – Parcialmente Satisfeito, S – Satisfeito.

Tabela 14 – Justificativa princípio 11

A tabela 15 apresenta uma justificativa das categorias atribuídas aos métodos ágeis quando comparados ao Princípio 12: Em intervalos regulares, as equipes devem refletir sobre como se tornarem mais eficazes e então se ajustar e adaptar seu comportamento.

Metodologia	**Categoria**	**Justificativa**
easYProcess	PS	Não existe práticas ou um conceito especificando este principio deixando em aberto. Talvez nas reuniões de acompanhamento

		pudessem implantar tal prática.
Scrum	S	Além das reuniões diárias no *Scrum* temos ao final de cada *Sprint* as reuniões de revisão e retrospectiva.

Legenda: NS – Não Satisfeito, PS – Parcialmente Satisfeito, S – Satisfeito.

Tabela 15 – Justificativa princípio 12

A seguir uma tabela completa com todos os princípios do manifesto ágil que foram utilizados nesta comparação resumindo o comparativo de acordo com suas categorias entre as metodologias *YP* e *Scrum* estudadas.

N.	**Princípio Ágil**	**easYProcess**	**Scrum**
1	A prioridade é satisfazer ao cliente através de entregas de software de valor contínuo e freqüentes	S	S
2	Receber bem as mudanças de requisito, mesmo em uma fase avançada, dando aos clientes vantagens competitivas.	PS	S
3	Entregar software em funcionamento com freqüência de algumas semanas ou meses, sempre na menor escala de tempo.	S	S
4	Equipes de negócio e de desenvolvimento devem trabalhar juntas diariamente durante todo o projeto.	PS	S
5	Manter uma equipe motivada fornecendo ambiente, apoio e confiança necessária para a realização do trabalho.	S	S
6	A maneira mais eficiente da informação circular dentro da equipe é a através de uma conversa face a face.	S	S

7	Ter o software funcionando é a melhor medida de progresso.	S	S
8	Processos ágeis promovem o desenvolvimento sustentável. Todos envolvidos devem ser capazes de manter um ritmo de desenvolvimento constante.	S	S
9	Atenção contínua a excelência técnica e um bom projeto aumentam a agilidade.	S	PS
10	Simplicidade é essencial.	S	S
11	As melhores arquiteturas, requisitos e projetos provêm de equipes organizadas.	S	S
12	Em intervalos regulares, as equipes devem refletir sobre como se tornarem mais eficazes e então se ajustar e adaptar seu comportamento.	PS	S

Fonte: (COCKBURN, 2000)

Tabela 16 – Resumo da comparação das metodologias de acordo com os princípios ágeis

Como podemos perceber na tabela 16, apenas 1 princípio foi parcialmente satisfatório pelo Scrum, enquanto os demais foram satisfatórios. Enquanto no YP obtemos 3 princípios como parcialmente satisfatórios.

A metodologia *YP* se destaca com suas práticas sistemáticas dando um grande valor a engenharia de processos, como podemos perceber na análise comparativa do princípio 9. Já a metodologia Scrum da um maior destaque na gerencia do projeto onde reuniões diárias acontecem durante o *Sprint* favorecendo o controle do projeto pelo *ScrumMaster*.

Na prática as duas modelagens atendem suas necessidades. O *YP* por ser uma metodologia voltada para projetos acadêmicos é mais detalhado em documentos e artefatos. Já o *Scrum* que é mais voltado para a gerência do processo

de desenvolvimento se destaca mais em aspectos de gerência e organização da equipe durante o processo.

Notamos que em muitos aspectos as duas metodologias se equivalem. Como por exemplo: A definição dos requisitos em *YP* se dá na construção do documento de visão onde serão listadas e geradas as *Users Stories*, No *Scrum*, a definição dos requisitos acontece com a criação do *Product Backlog* a partir da fase de preparação.

Outro aspecto que as duas metodologias se equivalem é a preparação para o processo de desenvolvimento onde no *YP* as *User Stories* são divididas em planos de releases e iteração. Já no Scrum, os requisitos listados no Product Backlog são divididos em *Sprint Backlog* onde serão dividido em *Sprints* para o desenvolvimento. Após cada release e *Sprint*, uma nova versão do produto é lançada para apresentação ao cliente.

Como as duas metodologias focam no contato constante com o cliente e na comunicação pessoal periódica entre a equipe, em projetos de software pode-se utilizar da tecnologia de comunicação atual e realizar estas reuniões através de sistemas de mensagens instantâneas, onde se pode até realizar teleconferência. Como exemplos destes sistemas podemos citar: *Microsoft MSN*, *Skype* e *Google Talk*.

De posse desta comparação, decidimos em utilizar uma metodologia híbrida, ou seja, utilizaremos as vantagens de cada uma na implementação do SysFarma.

5.6 IMPLEMENTAÇÃO

Para o desenvolvimento do sistema, conforme descrito anteriormente, utilizaremos as duas metodologias estudadas. Foi utilizada na modelagem uma arquitetura em três camadas onde provê maior escalabilidade, facilitando a manutenção e suporte.

Como do escopo do trabalho foi utilizado tecnologia *Open Source*, Java como plataforma de desenvolvimento e seus *frameworks*, *JBoss* como servidor de aplicação e banco de dados *MySQL*.

O código fonte da implementação pode ser visto no ANEXO C onde conterá também os arquivos de configuração que foram utilizados na implementação do SysFarma. Por fim, no ANEXO D, pode-se visualizar as telas do sistema.

6 CONCLUSÃO

Neste capítulo, serão apresentadas as conclusões deste trabalho, mostrando de forma sucinta o que foi realizado ao longo do mesmo possibilitando a identificação dos objetivos alcançados.

6.1 QUANTO AOS OBJETIVOS

6.1.1 OBJETIVO ESPECÍFICO I

I. Estudar as características gerais das metodologias ágeis e tradicionais de desenvolvimento de software;

No capitulo 2 foi atendido este objetivo, onde foi estudado as metodologias ágeis e as tradicionais comparando suas principais características. Como metodologia tradicional, foi estudado o RUP e como ágeis, foi estudada as metodologias XP, YP e Scrum.

6.1.2 OBJETIVO ESPECÍFICO II

II. Estudar mais detalhadamente as metodologias ágeis easYProcess e Scrum;

Este objetivo foi atendido com a realização dos capítulos 3 e 4 onde foi mais bem detalhado o funcionamento das metodologias YP e Scrum respectivamente.

Foi verificada ao longo da pesquisa que a metodologia YP é uma ótima alternativa, porém sendo específica em situações como, por exemplo, em projetos acadêmicos e tem como aspecto positivo a possibilidade de execução de todas as suas tarefas entre a equipe de desenvolvimento e o cliente.

Já a metodologia *Scrum* descreve melhor a organização de como o projeto deve ser executado, não definindo muitos documentos de especificação técnica do componente. Por exemplo, o *Scrum* não define que temos que criar um

modelo lógico de dados, porém a metodologia não impede que este artefato seja criado para facilitar a implementação do sistema sendo que este artefato pode ser definido como um item do *Product Backlog*.

6.1.3 OBJETIVO ESPECÍFICO III

III. Modelar, com as duas metodologias, um componente de software que servirá como experiência para avaliação;

Com a modelagem, pode-se perceber na prática que, como citado no objetivo específico II, o YP destaca-se por detalhar artefatos e especificação técnica e o *Scrum* no processo em si.

Um dos pontos negativos durante a modelagem *YP* é a definição de artefatos que julgamos desnecessários ou improdutivos como o modelo de tarefas. Já na modelagem *Scrum*, percebemos que é uma metodologia voltada para pessoas que já tenham um grau de conhecimento em projetos de software e que queiram otimizá-lo.

6.1.4 OBJETIVO ESPECÍFICO IV

IV. Avaliar a utilização de ambas as metodologias;

Após a modelagem avaliamos comparativamente as duas metodologias. Percebemos que, formalmente, ambas as metodologias comparadas satisfazem todos os princípios ágeis de forma satisfatória ou parcialmente satisfatória.

Com a modelagem, percebemos que muitos aspectos se equivalem entre as duas metodologias e também como descrito no objetivo específico III, cada metodologia tem seus pontos fortes e fracos. Para a implementação do sistema, optou-se em utilizar uma metodologia híbrida com os pontos fortes de cada uma delas. Como por exemplo os testes de aceitação e as reuniões diárias que o *YP* e o *Scrum* propõe respectivamente.

6.1.5 OBJETIVO ESPECÍFICO V

V. Iniciar o desenvolvimento do componente baseando-se na modelagem mais apropriada.

Em posse das duas modelagens, foi iniciado o desenvolvimento do componente SysFarma. Percebemos que com uma definição clara de todo o componente a ser desenvolvido, assim como a listagem de todas as tarefas a serem realizadas, o processo de desenvolvimento fluiu de uma forma rápida e objetiva.

As DailyScrum, reuniões diárias do Scrum, foram de ótima valia neste processo. Onde os integrantes da equipe trocaram informações de como está o andamento do Sprint/Release. Eventuais contratempos também foram tratados nestas reuniões, onde problemas foram resolvidos logo de princípio não gerando atrasos ao cronograma.

Percebemos que o uso de uma metodologia ágil é muito eficiente no andamento de um projeto de software, onde provavelmente a utilização de uma metodologia tradicional traria muita burocracia e documentação excessivamente desnecessária. E que o não uso de algum método de desenvolvimento poderia trazer divergências de especificação e falta de organização durante o desenvolvimento do software.

6.2 DIFICULDADES ENCONTRADAS

A principal dificuldade encontrada durante este trabalho foi a realização da implementação do componente. Devido à falta de conhecimento técnico da utilização da linguagem Java e seus *frameworks* na criação de sistemas *Web*.

6.3 QUANTO ÀS PERSPECTIVAS DE CONTINUIDADE

Como sugestão de trabalhos futuros objetivando as perspectivas de continuidade pode-se relacionar o seguinte:

- A criação formal de uma nova metodologia baseada no Scrum e no YP.

De acordo com seus pontos fortes e fracos.

Alternativamente, partindo do Scrum, pode-se buscar o aperfeiçoamento desta metodologia inserindo práticas mais diretas no processo de desenvolvimento, como criação do modelo lógico, testes do software e diagramas necessários. Esta proposta deve-se ao fato do Scrum centrar-se em processos e não em artefatos.

- Outra continuidade seria a conclusão do componente disponibilizado no repositório do projeto Via Digital, através da modelagem, para que o mesmo possa ser utilizado pelas farmácias de postos de saúde em prefeituras por todo o Brasil.

7 REFERÊNCIAS

AGILE MODELLING**.** Disponível em: <http://www.agilemodeling.com/>. Acessado em 11 agosto 2007.

AMBLER, Scott W. **Modelagem Ágil: Práticas eficazes para a Programação eXtrema e o Processo Unificado.** Porto Alegre: Bookman, 2004.

BEEDLE, Mike et al. **SRUM: Na extension pattern language for hyperprodutive software development.** Disponível em: <http://www.controlchaos.com>. Acessado em setembro de 2007.

CAMARA, Fabio. **Processos Ágeis e MSF.** 2005. Disponível em: <http://www.linhadecodigo.com.br/artigos.asp?id_ac=833>. Acessado em 12 agosto 2007.

CAMARA, Fabio. **Experimente um Projeto Ágil**. Disponível em: <http://www.linhadecodigo.com.br/artigos_impressao.asp?id_ac=1285>. Acessado em 30 agosto 2007.

COCKBURN, Alistair. **Agile Software Development.** Adisson-Wesley, 2001.

CÔRTES, Mario Lúcio; CHIOSSI, Thelma C. dos Santos. **Modelos de Qualidade de Software.** Campinas: Unicamp, Instituto de Computação, 2001.

CRUZ, Leandro Rodrigo Saad. **Introdução ao Scrum.** Disponível em: <http://scrum.squarespace.com/journal/2006/8/5/introduo-ao-scrum.html>. Acesso em 27 de agosto de 2007.

CRUZ, Leandro Rodrigo Saad. **Visão Geral.** Disponível em: <http://scrum.squarespace.com/journal/2006/8/16/viso-geral.html>. Acesso em 27 de agosto de 2007.

EASYPROCESS**.** Disponível em: <http://www.dsc.ufcg.edu.br/~yp/>. Acessado em 11 dez 2006.

EXTREME PROGRAMMING**.** Disponível em: <http://www.extremeprogramming.org/>. Acessado em 11 dez 2006.

FAGUNDES, Priscila Bastos. **Framework para Comparação e Análise de Métodos Ágeis.** Dissertação de Mestrado. Universidade Federal de Santa Catarina. Florianópolis: 2005.

FELIPPE, Cecília Machado. **Aplicação de Metodologia Ágil no Desenvolvimento de um Componente de Software para Prefeituras**. Trabalho de Conclusão de Curso. Universidade Federal de Santa Catarina: 2007.

FERREIRA, Décio; COSTA, Felipe; ALONSO, Filipe; ALVES, Pedro; NUNES, Tiago. **SCRUM Um Modelo Ágil para Gestão de Projetos de Software**.

FILHO, Edes Garcia da Costa. **Métodos Ágeis.** 2006. Disponível em: <http://www.dc.ufscar.br/~junia/MetAgEds.pdf>. Acessado em 30 março 2007.

FILHO, Edes Garcia da Costa; PENTEADO, Rosângela; SILVA, Júnia Coutinho Anacleto Silva; BRAGA, Rosana Teresinha Vaccare. **Padrões e Métodos Ágeis: Agilidade no Processo de Desenvolvimento de Software.** 2005. Disponível em: <http://sugarloafplop2005.icmc.usp.br/papers/9673.pdf>. Acessado em 25 março 2007.

FOWLER, Martin. **The New Methodology.** 2005. Disponível em: <http://www.martinfowler.com/articles/newMethodology.html>. Acessado em 15 abril 2007.

GARCIA, Francile Procópio; LIMA, Aliandro Higino Guedes; FERREIRA, Danilo de Sousa; JÚNIOR, Fábio Luiz Leite; ROCHA, Giselle Regina Chaves da; MENDES, Gustavo Wagner Diniz; PONTES, Renata França de; ROCHA, Verlaynne Kelley da Hora; DANTAS, Vinicius Farias. ***easYProcess* – Um Processo de Desenvolvimento de Software.** Universidade Federal de Campina Grande. Campina Grande: 2007.

GLOGER, Boris. **The Zen of Scrum**. Disponível em: <http://www.glogerconsulting.de>. Acessado em Setembro de 2007.

GONÇALVES, Patrícia. **Programação Extrema – XP.** Disponível em: <http://inf.unisinos.br/~barbosa/pipca/consipro1/a6.pdf >. Acesso em 11 de junho de 2007.

HAZZAN, O.; DUBINSKY, Y. **eXtreme Programming as a Framework for Student-Project Coaching in Computer Science Capstone Courses,** 2003.

HIGHSMITH, Jim. **Agile Software Development Ecosystems.** Adisson-Wesley, 2002.

HILMAN. **Metodologias Ágeis.** 2004. Disponível em: <http://www.redes.unb.br/material/ESOO/Metodologias%20%c1geis.pdf>. Acessado em 20 abril 2007.

IEEE CS Press. 1991.

KNIBERB Henrik. **Scrum and XP from the Trenches.** *E-book* disponível em: <http://www.crisp.se/henrik.kniberg/ScrumAndXpFromTheTrenches.pdf>. Acesso em 08 de setembro de 2007.

KRUCHTEN, Philippe. **Introdução ao RUP - Rational Unified Process.** Rio de Janeiro: Ciência Moderna, 2003.

KUHN, Giovane; PAMPLONA, Vitor. **Apresentando XP. Encante seus Clientes com Extreme Programming.** Disponível em: <http://www.javafree.org/content/view.jf?idContent=5>. Acesso em 11 de junho de 2007.

LAKATOS, Eva Maria, MARCONI, Marina de Andrade. **Fundamentos de metodologia científica.** 3. ed. São Paulo: Atlas, 1991.

LARMAN, Craig. **Applying UML And Patterns.** 2nd Edition, 2002.

LUNA, Maria. **Plano Municipal de Assistência Farmacêutica**, 2006. Disponível em: <http://www.pmf.sc.gov.br/saude/Documentos/plano_de_assistencia_farmaceutica.doc>. Acessado em Setembro de 2007.

MAGALHÃES, Ana Liddy C. C. **O Gerenciamento de Projetos de Software Desenvolvidos à Luz das Metodologias Ágeis.** 2004. Disponível em:

<http://ftp.mct.gov.br/Temas/info/Dsi/PBQP/EQPS_Campinas_2004/Projeto%202_10.pdf>. Acessado em 25 março 2007.

MANIFESTO FOR AGILE SOFTWARE DEVELOPMENT. Disponível em: < http://www.agilemanifesto.org/>. Acessado em Outubro de 2007

MARÇAL, Ana Sofia Cysneiros; FREITAS, Bruno Celso Cunha; SOARES, Felipe Santana Furtado; MACIEL, Teresa Maria Medeiros; BELCHIOR, Arnaldo Dias. **Estendendo o SCRUM segundo as Áreas de Processo de Gerenciamento de Projetos do CMMI.** CLEI2007: XXXIII Conferencia Latinoamericana de Informática, San Jose, Costa Rica. Disponível em: <http://www.cesar.org.br/files/file/SCRUMxCMMI-CLEI-2007.pdf>. Acesso27 de agosto de 2007.

MARTINS, José Carlos Cordeiro. **Gerenciando projetos de desenvolvimento de software com PMI, RUP e UML.** Rio de Janeiro: Brasport, 2006.

NETO, Oscar Nogueira de Souza. **Análise Comparativa das Metodologias de Desenvolvimento de Softwares Tradicionais e Ágeis.** Trabalho de Conclusão de Curso. Universidade da Amazônia. Belém: 2004.

PEREIRA, Arthur; DEMÉTRIO, Adriano. **ALOHA Um Ambiente Virtual de Aprendizado.** Trabalho de Conclusão de Curso. Universidade Federal de Santa Catarina: 2007.

PRESSMAN, Roger. **Software Engineering – A Pratitictioner's Approach.** McGraw-Hill, 5th Edition, 2001.

RATIONAL UNIFIED PROCESS. Disponível em: <http://www.wthreex.com/rup/>. Acessado em 11 dezembro 2006.

SANTOS, Alexandre; MARTINS, Jefferson; LEAL, Manoel. **Agile Modeling – Overview.** Disponível em: <http://www.pr.gov.br/batebyte/edicoes/2003/bb131/agile.shtml>. Acesso em 03 de junho de 2007.

SEGUNDO, Volmar P.; MOURA, Alexandre M. **Introdução de metodologias ágeis de desenvolvimento de software nos currículos de referência do ensino universitário.** 2004. Disponível em: <http://www.frb.br/ciente/Impressa/Info/2004.2/Introducao%20de%20metodologias.pdf >. Acessado em 24 fevereiro 2007.

SCHWABER, Ken, BEEDLE, Mike. **Agile Software Development with SCRUM. Prentice Hall**, 2002.

SCRUM ALLIANCE. Disponível em: <http://www.scrumalliance.org> Acesso em Setembro de 2007.

SCRUM FOR TEAM SYSTEM. Disponível em: <http://scrumforteamsystem.com/en/default.aspx>. Acesso em 28 de agosto de 2007.

SOARES, Michel dos Santos. **Comparação entre Metodologias Ágeis e Tradicionais para o Desenvolvimento de Software.** 2004. Disponível em: <http://www.dcc.ufla.br/infocomp/artigos/v3.2/art02.pdf> . Acessado em 11 janeiro 2007.

SOMMERVILLE, Ian**. Engenharia de Software**. São Paulo: Addison-Wesley, 2003.

SOMMERVILLE, Ian. **Software Engineering.** 7th Edition, 2004.

SOMMERVILLE, I. (1996). Software engineering. New York, Addison-Wesley. Song, X. and Osterweil, L. J. **Comparing planejamento methodologies through process modeling**. 1st International Conference on Software Process, Los Alamitos, Calif.,

THE STANDISH GROUP. Disponível em: <http://www.standishgroup.com/>. Acessado em 12 dezembro 2006.

VERSIONONE. **State of Agile Development**. Disponível em: <http://www.versionone.com/pdf/StateofAgileDevelopmentSurvey.pdf>. Acesso em 15 de setembro de 2007.

VIA DIGITAL. **Caminho inteligente para informatização pública.** Disponível em: <http://www.viadigital.org.br/ >. Acesso em 06 de outubro de 2007.

YOSHIMA, Rodrigo. **Gerenciamento de Projetos com Scrum**. Disponível em: <http://www.aspercom.com.br/ead/mod/resource/view.php?id=245>. Acesso em 13 de setembro de 2007.

WASLAWICK, Raul Sidney. **Análise e Projetos de Sistemas de Informação Orientado a Objetos.** Editora Campus, 2004.

WIKIPEDIA. Disponível em: <http://en.wikipedia.org/wiki/Main_Page>. Acessado em 20 fevereiro 2007.

ZANATTA, Alexandre. **xScrum: Uma Proposta de Extensão do xScrum para Adequação ao CMMI**. Dissertação de Mestrado. Universidade Federal de Santa Catarina. Florianópolis: 2004.

8 ANEXO A – MODELAGEM YP

8.1 DEFINIÇÃO DE PAPÉIS

Conforme a metodologia *easYProcess*, a primeira etapa do processo de desenvolvimento consiste na especificação de papéis. O *YP* sugere cinco papéis sendo cada um responsável por sua tarefa.

Uma mesma pessoa pode desenvolver mais de um papel simultaneamente ainda mais se tratando de equipes pequenas de desenvolvimento.

O desenvolvimento deste sistema contará com duas pessoas na equipe conforme Tabela 3 abaixo.

EQUIPE	PAPÉIS
Diogo	Desenvolvedor e Testador
Jonatas	Gerente, Desenvolvedor e Testador
Cliente	Cliente
Enfermeiras do Posto	Usuário

Tabela 17 – Definição dos papéis do componente.

8.2 CONVERSA COM O CLIENTE

O próximo passo a ser realizado é a conversa com o cliente. Onde informações importantes como o escopo do problema serão adquiridas. Para isso, um conjunto de perguntas (Tabela 4) foi elaborado a fim de extrair o máximo de informações importantes sobre o que realmente é importante para o sistema.

ROTEIRO DE PERGUNTAS PARA CONVERSA COM O CLIENTE:
1) Quais são as principais característica da farmácia?
2) Como é o funcionamento da farmácia?

3) Quais os procedimentos existentes no dia a dia da farmácia?
4) Quais são os documentos necessários para adquirir seu medicamento na farmácia?
5) Quem vai utilizar este sistema?
6) Quantas pessoas vão utilizar o sistema?
7) O que um sistema de controle de estoque de farmácia deve ter para atender as necessidades de quem vai utilizá-lo?
8) Quais as funcionalidades básicas que o sistema deve ter para que se consiga administrar o controle de estoque da farmácia?
9) Como essas funcionalidades devem se relacionar para uma melhor administração do controle do estoque da farmácia?
10) Quais as funcionalidades são mais importantes?

Tabela 18 – Roteiro de perguntas para primeira conversa com o cliente.

ROTEIRO DE PERGUNTAS PARA CONVERSA COM O CLIENTE:

1) Quais são as principais característica da farmácia?

A farmácia é um estabelecimento que visa a saúde de populaçoes com a atenção farmacêutica integral voltada na dispensação, orientação ao usuário, quanto ao uso racional dos medicamentos.

2) Como é o funcionamento da farmácia?

A farmácia funciona das 07:00hs às 19:00hs de segunda à sexta feira, fazendo atendimento ao usuário com 4 funcionarios atendente e 1 farmacêutico responsável.

3) Quais os procedimentos existentes no dia a dia da farmácia?

Para fazer dispensação do medicamento é necessário a apresentação do receituário médico, juntamente com os documentos necessários.

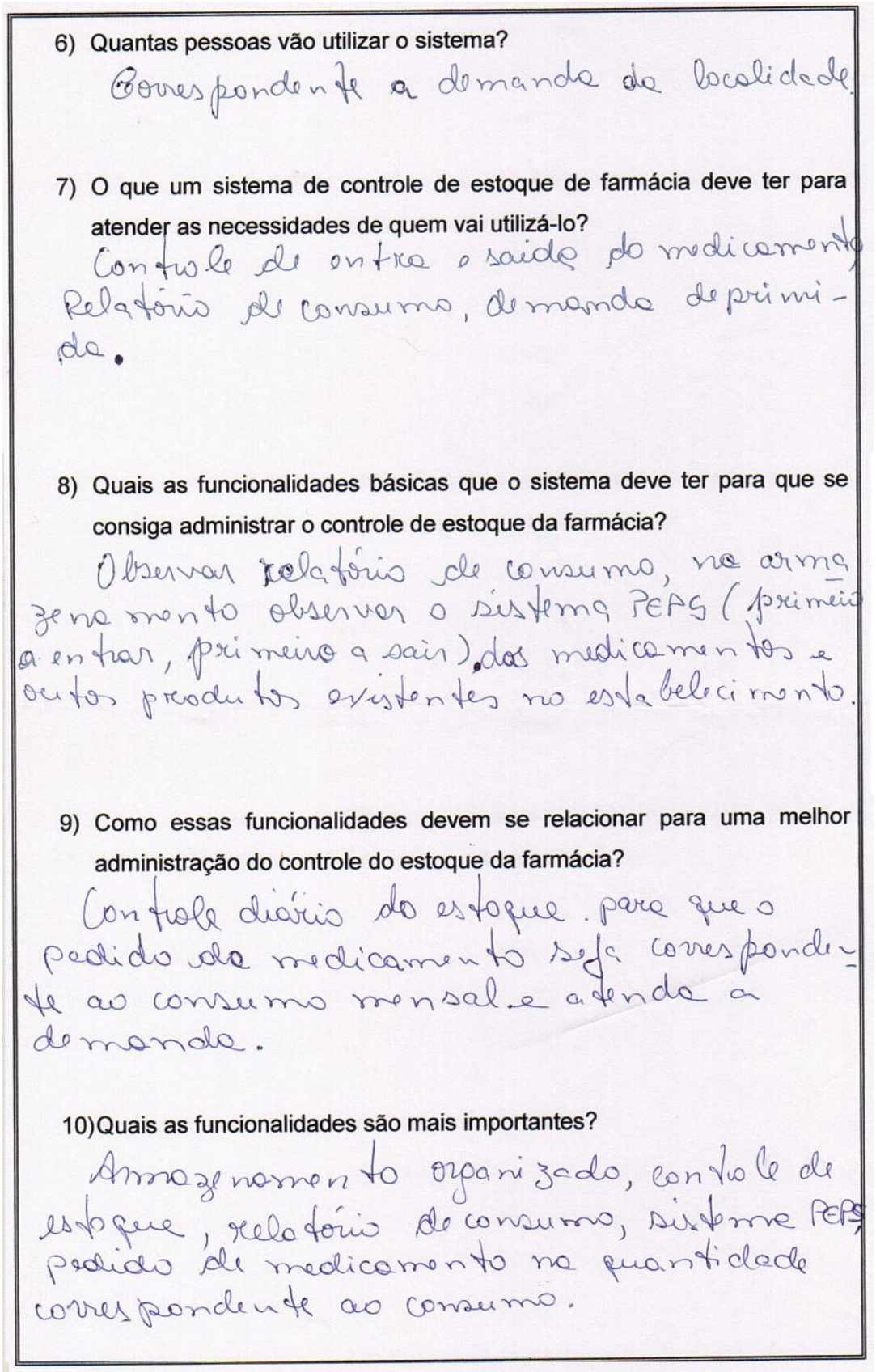
4) Quais são os documentos necessários para adquirir seu medicamento na farmácia?

Os documentos necessários são: o cartão do SUS; RG e receituário do médico.

5) Quem vai utilizar este sistema?

Toda a população.

Figura 14 – Respostas do Roteiro de Perguntas I



6) Quantas pessoas vão utilizar o sistema?

Correspondente a demanda da localidade.

7) O que um sistema de controle de estoque de farmácia deve ter para atender as necessidades de quem vai utilizá-lo?

Controle de entrada e saida do medicamento, Relatório de consumo, demanda de primeira.

8) Quais as funcionalidades básicas que o sistema deve ter para que se consiga administrar o controle de estoque da farmácia?

Observar relatório de consumo, no armazenamento observar o sistema PEPS (primeiro a entrar, primeiro a sair), dos medicamentos e outros produtos existentes no estabelecimento.

9) Como essas funcionalidades devem se relacionar para uma melhor administração do controle do estoque da farmácia?

Controle diário do estoque. para que o pedido de medicamento seja correspondente ao consumo mensal e atenda a demanda.

10) Quais as funcionalidades são mais importantes?

Armazenamento organizado, controle de estoque, relatório de consumo, sistema PEPS, pedido de medicamento na quantidade correspondente ao consumo.

Figura 15 – Respostas do Roteiro de Perguntas II

Após a primeira conversa com o cliente, a equipe de desenvolvimento tem que ter bem claro o escopo do problema. Ter o perfil dos usuários do sistema bem como os requisitos funcionais e não-funcionais e se possível os riscos do sistema.

Tendo a conversa com o cliente finalizada, o próximo passo é criar o documento de visão do sistema. Onde conterá informações do que o sistema se propõe fazer de forma a apoiar o processo de desenvolvimento do *software*.

8.2.1 DOCUMENTO DE VISÃO

O SysFarma é um sistema que servirá como um controle de estoque de farmácia das unidades de saúde. Um sistema que ajudará no processo de reposição de estoques dos medicamentos e para se ter um melhor planejamento a fim de se evitar erros de estoque.

O sistema contará com um controle de usuários onde haverá um usuário administrador geral que fará o controle de usuários no sistema e usuários que farão as entradas e saídas dos medicamentos.

Com o sistema, o usuário poderá efetuar pesquisas por princípio ativo ou sua descrição e gerar relatórios da situação geral dos medicamentos.

O maior objetivo do SysFarma, é ter um controle fiel de seus medicamentos disponíveis, quantos medicamentos estão em estoque e quantos medicamentos a comunidade usufruiu.

8.2.2 REQUISITOS

8.2.2.1 Funcionais

Os requisitos funcionais do Projeto *SysFarma* são:

- Cadastro de medicamentos;
- Adicionar medicamentos em estoque;
- Buscar informações dos medicamentos;
- Saída de medicamentos;

- Geração de relatórios.

8.2.2.2 Não-Funcionais

Os requisitos não funcionais do Projeto SysFarma são:

- Interface WEB;
- Segurança (Sistema de login);
- Banco de Dados MySQL;
- Arquitetura em camadas;
- Utilização dos padrões de projeto (VO e DAO).

8.2.3 PERFIL DO USUÁRIO

Agentes da secretaria de saúde do município e/ou farmacêuticos que em geral possuem pouco conhecimento de informática. Os agentes e/ou farmacêuticos possuem um nível médio de curiosidade em relação ao *software* tornando desnecessário treinamento para a utilização do sistema, pois ambos têm iniciativa de alto aprendizado.

8.2.4 OBJETIVOS DE USABILIDADE

Os objetivos de usabilidade estão descritos na tabela 5 abaixo:

OBJETIVO	MENSURAÇÃO
Manter clareza na estrutura	Número de ações incorretas e de erros repetidos.
Possuir telas simples	Observar dificuldades de navegação

Tabela 19 – Objetivos de Usabilidade

8.3 INICIALIZAÇÃO

Após a equipe de desenvolvimento ter a idéia principal a se resolver, tem o inicio de algumas atividades de análise do sistema como o modelo de tarefa e a definição das *User Stories* pelo cliente.

8.3.1 MODELO DA TAREFA

O modelo de tarefa representa de forma gráfica como o sistema deve se comportar diante do usuário a fim de melhorar a compreensão do desenvolvedor com relação à natureza da tarefa, as partes que compõe, e a ordem com que devem ser realizadas.

Como o presente projeto não tem a finalidade de implementação completa do sistema, mas sim uma comparação entre duas metodologias ágeis, foi construído o Modelo de Tarefa para as funcionalidades de Login e Cadastro de Medicamentos, a fim de ter um exemplo deste processo, como pode ser visto na figura 14 e figura 15 respectivamente.

- Login:

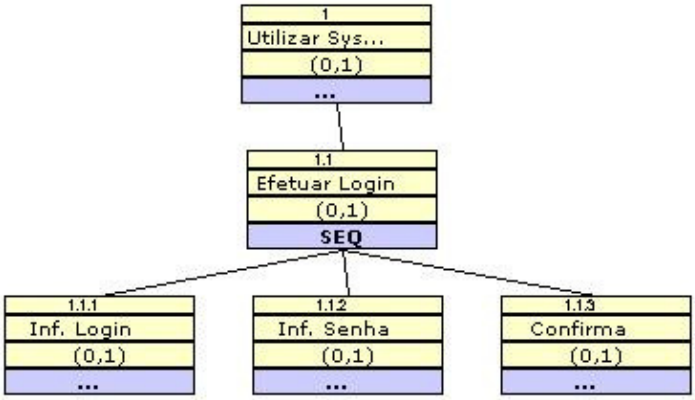


Figura 16 – Modelo de Tarefa Login

- Cadastro de Medicamentos:

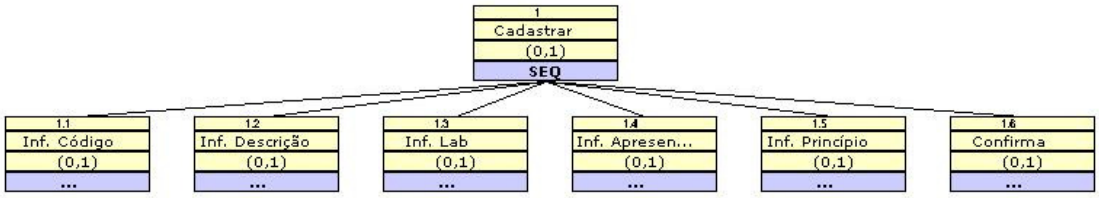


Figura 17 – Modelo de Tarefa Cadastro de Medicamentos

8.3.2 USER STORIES

User Stories são pedaços do sistema que o cliente define junto com a equipe de desenvolvimento. É importante que neste momento o cliente defina quais são prioridades para estar pronto e funcionando em primeiro. A Tabela 6 abaixo temos as *User Stories* do *SysFarma* com suas respectivas estimativas de tempo:

Lista de *User Stories*	
US01	**Incluir funcionalidade de autenticação (restrição)**
	TA1.1 - Acessar o sistema a partir de um login válido e ativo (autenticação feita com sucesso); TA1.2 - Acessar o sistema a partir de um login inválido ou inativo (mensagem de erro deve ser exibida); TA1.3 – A partir de um usuário administrador cadastrar novo usuário com todos os campos obrigatórios (Cadastro efetuado com sucesso); TA1.4 – A partir de um usuário administrador cadastrar novo usuário sem informar todos os campos obrigatórios (Cadastro não deve ser efetuado); TA1.5 - A partir de um usuário administrador alterar dados de um usuário cadastrado (Alteração efetuada com sucesso);
	Estimativa Inicial: 8 horas.
US02	**Programar funcionalidades de cadastro de medicamentos**
	TA2.1 - Cadastrar medicamentos da unidade informando todos os campos obrigatórios (Cadastro efetuado com sucesso); TA2.2 - Cadastrar medicamentos da unidade sem informar todos os campos obrigatórios (Cadastro não deve ser efetuado). TA2.3 - Alterar dados de um medicamento cadastrado a partir do seu código ou descrição (Cadastro alterado com Sucesso); TA2.4 - Excluir um medicamento cadastrado a partir de seu código ou descrição e que tenham movimentação (Mensagem de erro deve ser exibida); TA2.5 - Excluir um medicamento cadastrado a partir de seu código ou descrição e que não tenham movimentação (Exclusão efetuado com sucesso);
	Estimativa Inicial: 8 horas
US03	**Programar funcionalidades de saída de medicamentos**
	TA3.1 - Registrar saída de medicamentos da unidade informando todos os campos obrigatórios (Saída efetuado com sucesso); TA3.2 - Registrar saída de medicamentos da unidade sem informar todos os campos obrigatórios (Saída não deve ser efetuado).

	Estimativa Inicial: 8
US04	**Programar funcionalidade de entrada de medicamentos**
	TA1.1 - Registrar entrada de medicamentos da unidade informando todos os campos obrigatórios (Entrada efetuado com sucesso); TA1.2 - Registrar entrada de medicamentos da unidade sem informar todos os campos obrigatórios (Entrada não deve ser efetuado).
	Estimativa Inicial: 4
US06	**Programar funcionalidade de pesquisa de medicamentos**
	TA6.1 - Recuperar dados gerais do medicamento a partir de um código existente (Dados devem ser retornados com sucesso); TA6.2 - Recuperar dados gerais do medicamento a partir de um código inexistente (Mensagem de erro deve ser retornada); TA6.3 - Recuperar dados gerais do medicamento a partir de uma descrição existente (Dados devem ser retornados com sucesso); TA6.4 - Recuperar dados gerais do medicamento a partir de uma descrição inexistente (Mensagem de erro deve ser retornada); TA6.5 - Recuperar dados gerais do medicamento a partir de um princípio existente (Dados devem ser retornados com sucesso); TA6.6 - Recuperar dados gerais do medicamento a partir de um princípio inexistente (Mensagem de erro deve ser retornada).
	Estimativa Inicial: 8
US07	**Programar funcionalidades de relatórios**
	TA7.1 - Gerar relatório dos medicamentos em estoque; TA7.2 - Gerar relatório das saídas dos medicamentos por pacientes; TA7.3 - Gerar relatório de saída dos medicamentos por movimentação; TA7.4 - Gerar relatório de entrada/saída dos medicamentos a partir de um período válido.
	Estimativa Inicial: 8

Tabela 20 – User Stories.

8.3.3 TESTES DE ACEITAÇÃO

Os testes de aceitação têm por finalidade verificar se todos os requisitos do sistema foram implementados corretamente. Estes teste de aceitação podem ser melhor visualizados nas páginas seguintes com o Plano de Iteração na Tabela 8.

8.3.4 PROTÓTIPO DA INTERFACE

A partir do modelo de tarefa, foi construído o protótipo de interface. Como o YP preconiza a construção de protótipos simples (esboço), não foi utilizado nenhuma ferramenta na construção do protótipo de interface SysFarma como podem ser visualizados nas figuras seguintes.

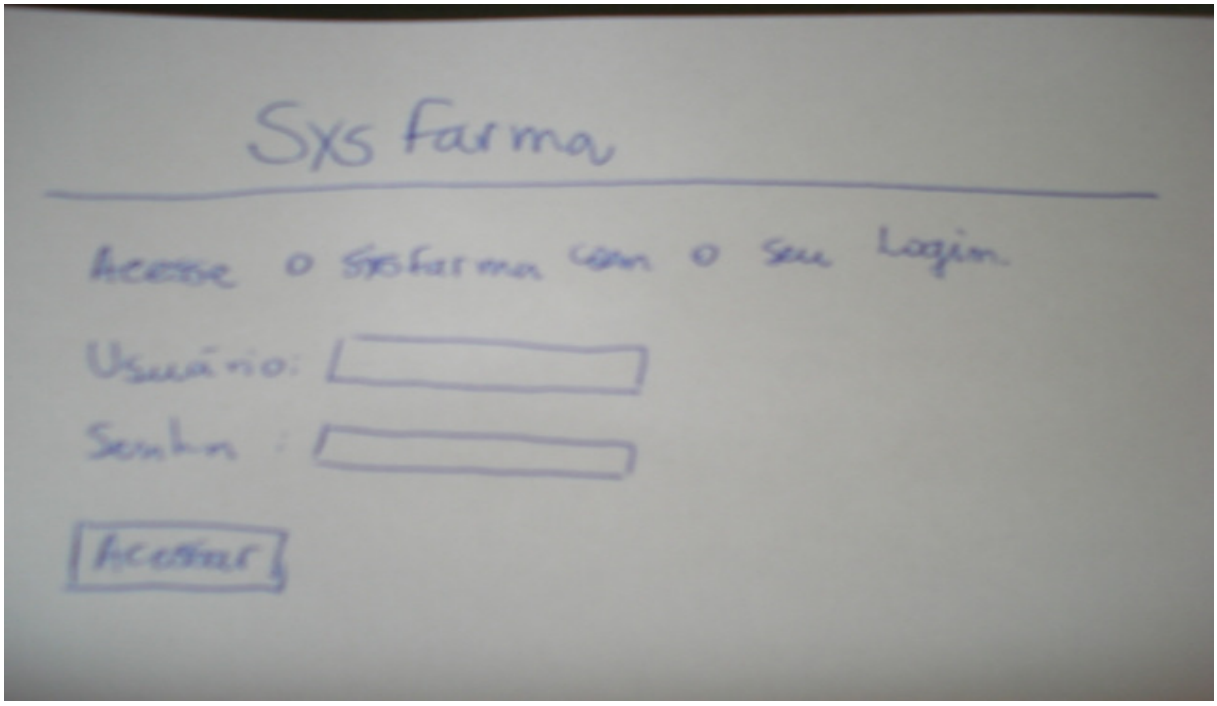


Figura 18 – Protótipo de Interface para Login do SysFarma

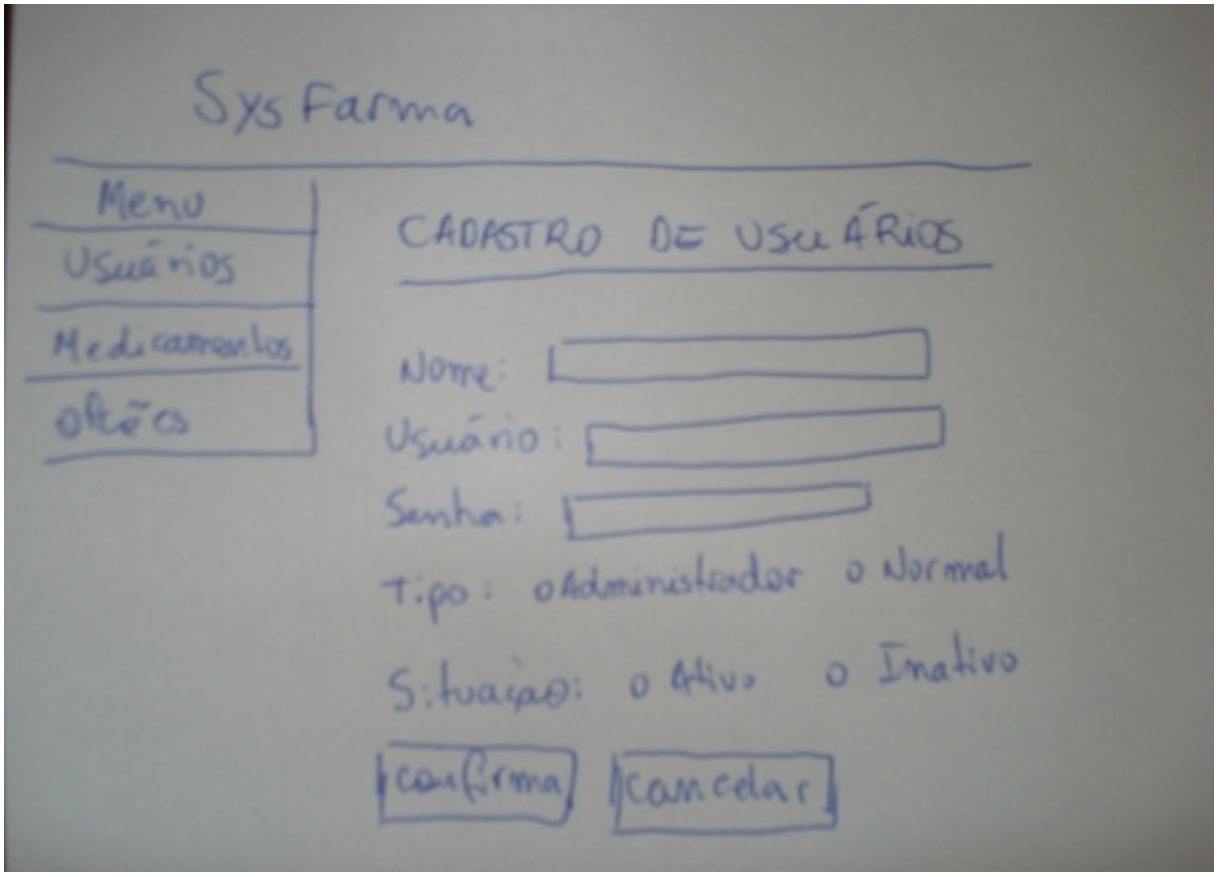


Figura 19 – Protótipo de Interface para Cadastro de Usuários

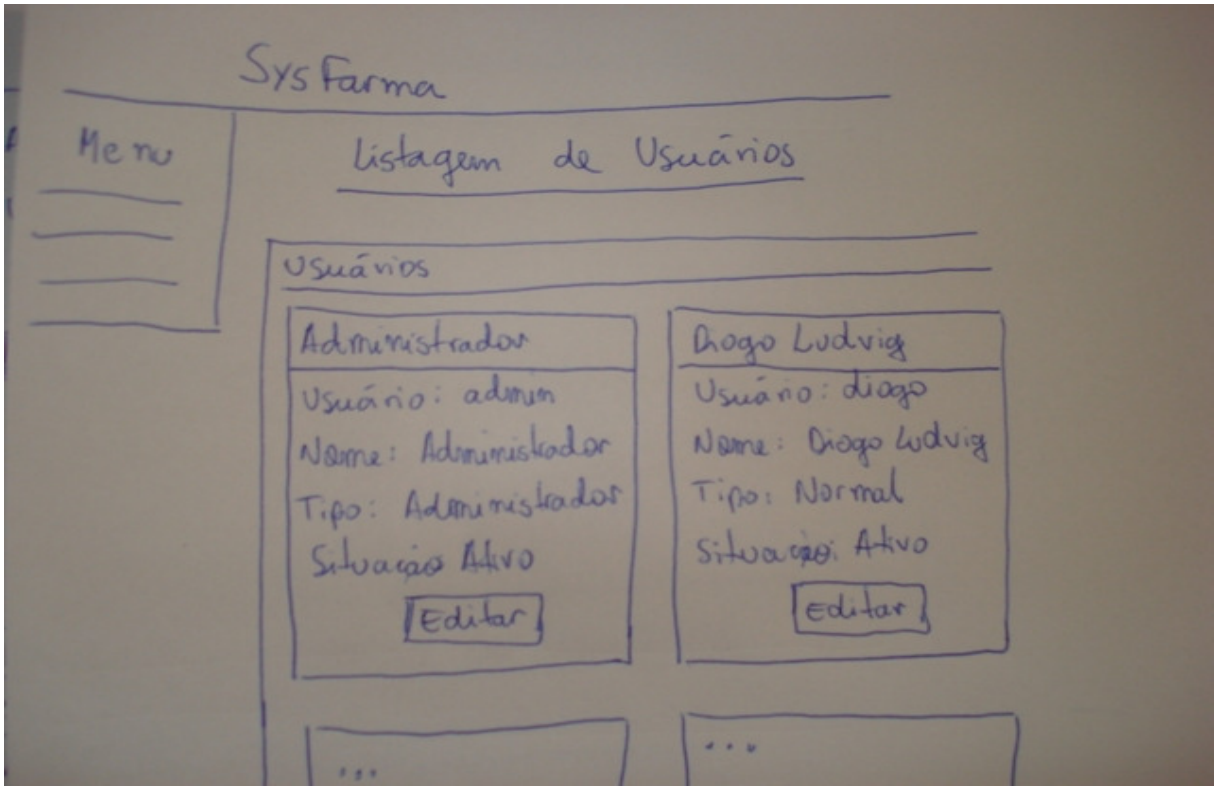


Figura 20 – Protótipo de Interface para Listagem de Usuários

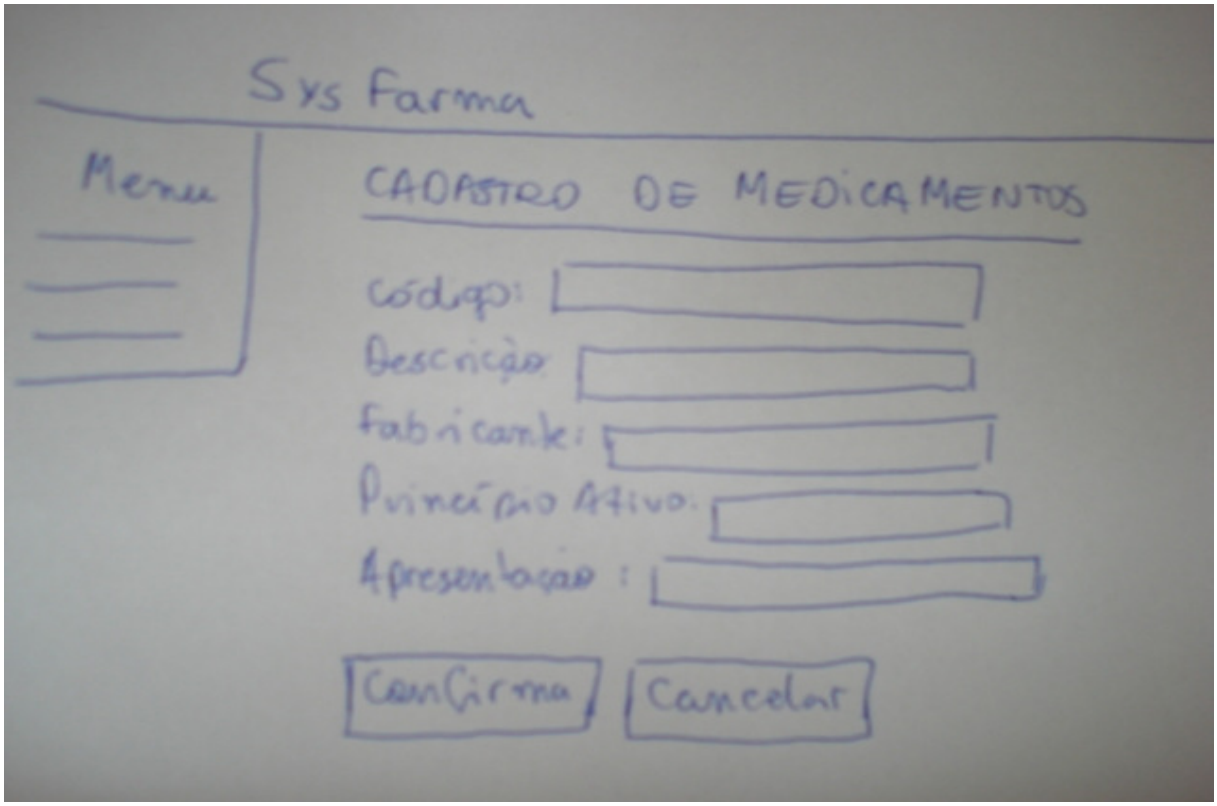


Figura 21 – Protótipo de Interface para Cadastro de Medicamentos

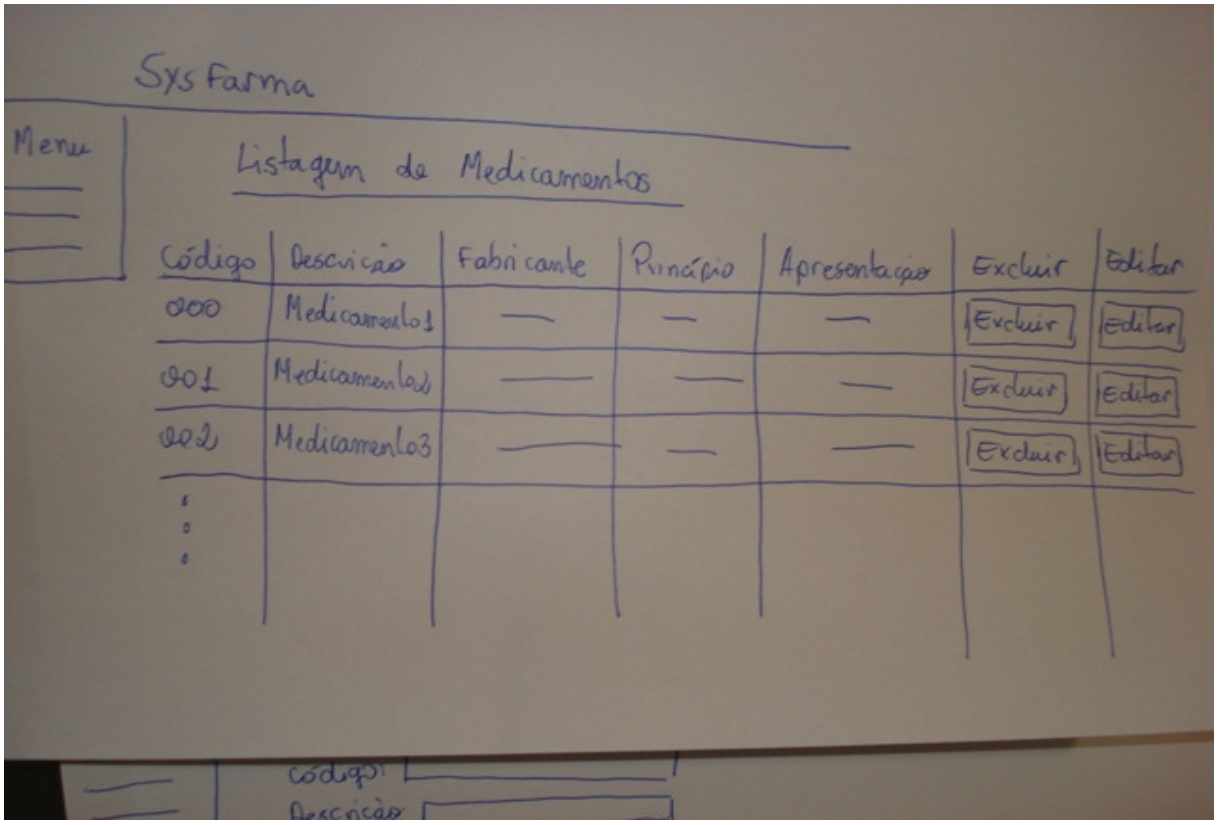


Figura 22 – Protótipo de Interface para Listagem de Medicamentos

8.3.5 PROJETO ARQUITETURAL

A arquitetural utilizada para a construção do componente segue o padrão em camadas. Camada de apresentação (ou visão), camada de negócio e camada de persistência de dados. De acordo com o padrão, as camadas serão implementadas usando tecnologia web, abertas e *Open Source*.

Na camada de apresentação, serão utilizadas paginas em HTML. A camada de negócio e camada de persistência estará de acordo com as características descritas abaixo:

- Servidor de aplicação JBoss 4.2.1, que implementa as especificações J2EE.
- Páginas JSF (*Java Server Faces* 1.2) onde as requisições serão feitas pelo browser utilizando as bibliotecas *RichFaces* e *Ajax4jsf*.

- Todos os algoritmo de regra de negócio e de acesso aos dados será implementado em componentes *Managed Beans*, que serão referenciados nas páginas JSF;
- A persistência dos objetos estarão implementados sob o projeto DAO, utilizando JPA (Java Persistence Api) implementado através do Hibernate 3.2.1 e EJB3 (os Entitys Beans)
- Para a persistência dos dados, será adotado o SGBD relacional MySQL pois suporta os requisitos existentes e por ser um SGBD livre.

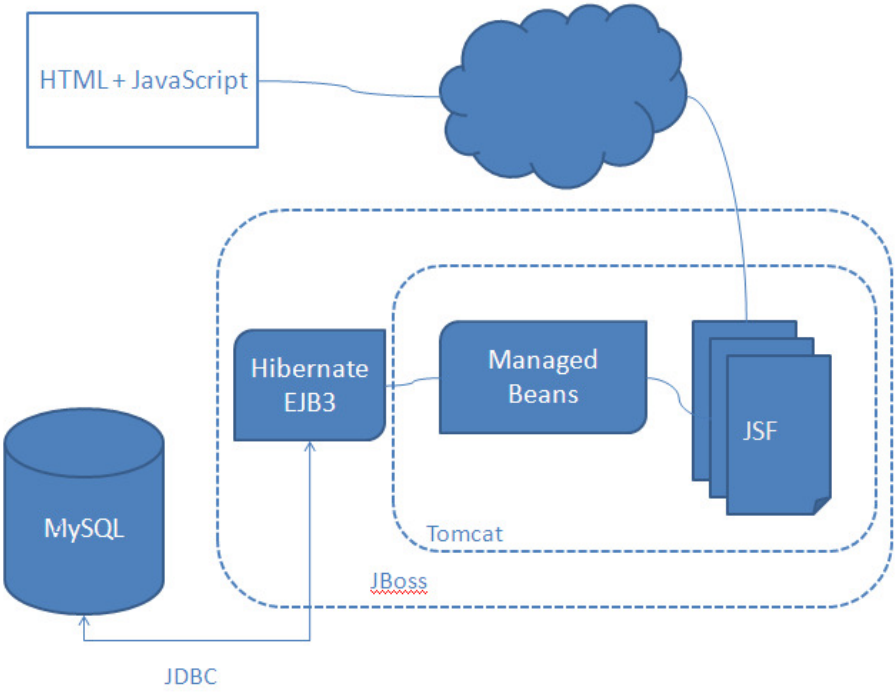


Figura 23 – Projeto Arquitetural

8.3.6 MODELO LÓGICO DE DADOS

O modelo lógico de dados tem como objetivo definir o comportamento do banco de dados diante do sistema. É uma representação da estrutura lógica dos dados armazenados que pode ser visualizado na Figura abaixo.

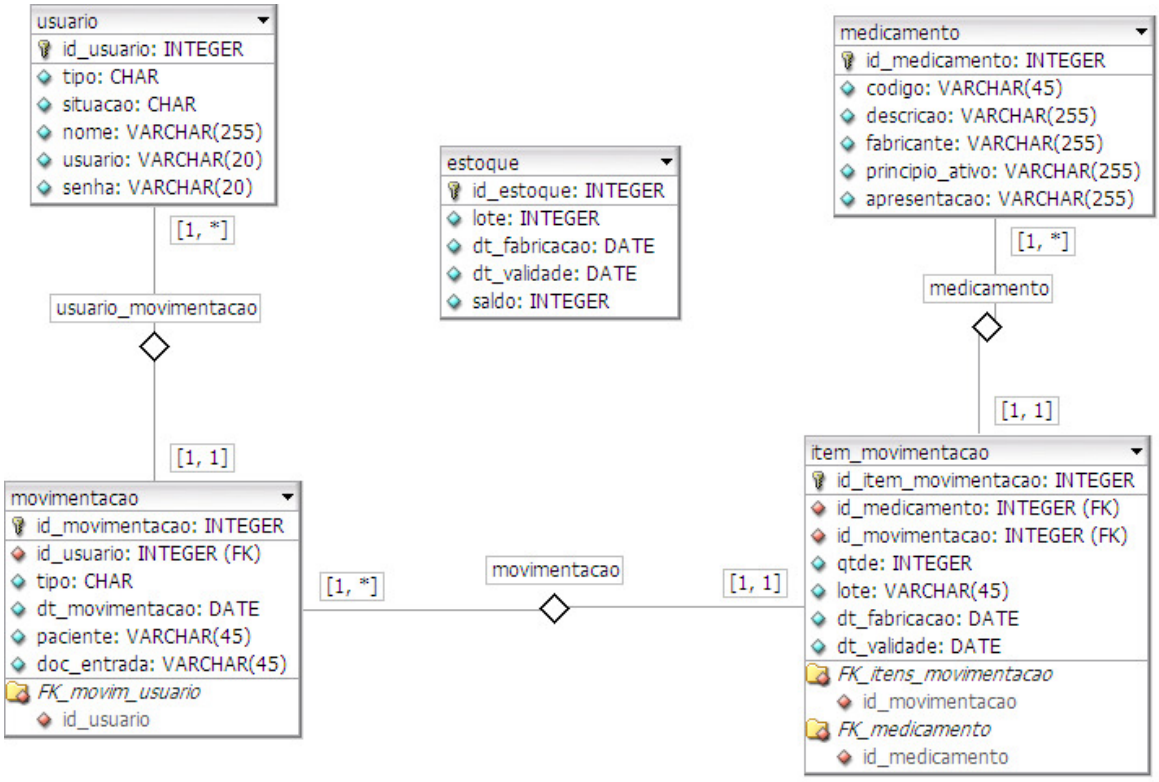


Figura 24 – Modelo Lógico de Dados

8.4 PLANEJAMENTO

Tendo as atividades de inicialização finalizadas, tem-se o início do planejamento do projeto. A fase de planejamento é composta por *Releases* e iterações necessárias para conclusão do projeto. É importante estar ciente do tempo de desenvolvimento do projeto de acordo com as *User Stories* definidas.

Inicialmente temos o planejamento dos *Releases* de acordo com alocação das *User Stories* definido juntamente com o cliente qual *Release* deve ser implementado primeiro. Completando o planejamento dos *Releases*, tem-se a execução dos testes de aceitação das *User Stories*, que são elaborados pelo cliente. Vale lembrar que somente é finalizada uma *User Stories* após o teste de aceitação for aceito pelo cliente.

Após a fase de planejamento de *Release* dá-se a fase de planejamento de iteração. O planejamento de iteração é a divisão das *User Stories*, as quais estavam anteriormente englobadas em *releases*. Portanto, cada iteração será um conjunto de atividades a serem desenvolvidas, e por sua vez, cada release será um conjunto de iterações.

Um cronograma das iterações com a data de início e término foi criado a fim de se ter um melhor gerenciamento do andamento das tarefas. Segundo a metodologia, para auxiliar o gerente na gerencia, se faz uso da tabela de alocação de atividades (TAA) onde são especificados as *User Stories*, tarefas, responsáveis, estimativa de tempo, tempo real consumido e status da tarefa.

8.4.1 PLANO DE RELEASE

As tabelas seguintes apresentam os Planos de Release com suas respectivas *User Stories* e período de realização.

Release 01		**Gerente - Diogo**
Iteração	***User Stories***	**Período**
Iteração 01	US01	24/09/2007 – 26/09/2007
Iteração 02	US02	26/09/2007 – 28/09/2007

Tabela 21 – Plano de Release 1

Release 02		**Gerente – Jonatas / Diogo**
Iteração	***User Stories***	**Período**
Iteração 03	US03	
Iteração 04	US04	

Tabela 22 – Plano de Release 2

Release 03		Gerente - Jonatas
Iteração	***User Stories***	**Período**
Iteração 05	US05	
Iteração 06	US06, US07	

Tabela 23 – Plano de Release 3

8.4.2 PLANO DE ITERAÇÃO

A seguir são demonstrados os dois Planos de Iteração executados no Projeto SysFarma juntamente com seus testes de aceitação.

Iteração 01 – (24/09/2007 – 30/09/2007)					
US01 - Programar funcionalidades de LOGIN					
Teste de Aceitação					***Status***
TA1.1	Acessar o sistema a partir de um login válido e ativo (autenticação feita com sucesso)				
TA1.2	Acessar o sistema a partir de um login inválido ou inativo (mensagem de erro deve ser exibida)				
TA1.3	A partir de um usuário administrador cadastrar novo usuário com todos os campos obrigatórios (Cadastro efetuado com sucesso)				
TA1.4	A partir de um usuário administrador cadastrar novo usuário sem informar todos os campos obrigatórios (Cadastro não deve ser efetuado)				
TA1.5	A partir de um usuário administrador alterar dados de um usuário cadastrado (Alteração efetuada com sucesso)				
Atividade	**Descrição**	**Responsável**	**Estimativa de tempo (horas)**	**Tempo real (horas)**	***Status***
A1.1	Implementar interface de Login	Diogo	4		
A1.2	Gerar script para configuração do MySQL	Diogo	2		
A1.3	Instalar aplicação em Servidor	Diogo	1		
A1.4	Gerar script dos testes de aceitação	Diogo	1		

Tabela 24 – Plano de Iteração 1 e seus respectivos Testes de Aceitação

Iteração 02 – (xx/xx/2007 – xx/xx/2007)		
US02 - Programar funcionalidades de cadastro de medicamentos		
Teste de Aceitação		***Status***
TA2.1	Cadastrar medicamentos da unidade informando todos os campos obrigatórios (Cadastro efetuado com sucesso)	
TA2.2	Cadastrar medicamentos da unidade sem informar todos os campos obrigatórios (Cadastro não deve ser efetuado)	

TA2.3	Alterar dados de um medicamento cadastrado a partir do seu código ou descrição (Cadastro alterado com Sucesso);	
TA2.4	Excluir um medicamento cadastrado a partir de seu código ou descrição e que tenham movimentação (Mensagem de erro deve ser exibida);	
TA2.5	Excluir um medicamento cadastrado a partir de seu código ou descrição e que não tenham movimentação (Exclusão efetuado com sucesso);	

Atividade	**Descrição**	**Responsável**	**Estimativa de tempo (horas)**	**Tempo real (horas)**	***Status***
A1.1	Implementar interface para cadastro dos medicamentos	Diogo	4		
A1.2	Gerar script para configuração do MySQL	Diogo	2		
A1.3	Instalar aplicação em Servidor	Diogo	1		
A1.4	Gerar script dos testes de aceitação	Diogo	1		

Tabela 25 – Plano de Iteração 2 e seus respectivos Testes de Aceitação

8.4.3 MATRIZ DE COMPETÊNCIAS

Na tabela abaixo esta a matriz de competência do Projeto SysFarma.

EQUIPE	COMPETÊNCIA	TEMPO DE DEDICAÇÃO (HORAS/SEMANA)
Jonatas	- HTML - SQL - Java - Facilidade de Comunicação	8
Diogo	- HTML - SQL - Java - Facilidade de Comunicação	8

Tabela 26 – Matriz de Competencia

8.4.4 TAA PARCIAL

A TAA (Tabela 10 e Tabela 11) somente é preenchida durante a reunião de acompanhamento onde as informações referentes ao tempo real de desenvolvimento e os *Status* estarão disponíveis.

8.5 IMPLEMENTAÇÃO

As atividades definidas no plano de iteração são realizadas nesta fase. Para a implementação do sistema, a metodologia adota algumas práticas tais como: *Design Simples*, Padrões de Projeto e Padrões de Códigos a fim de se produzir um código com maior qualidade.

Padrões altamente aceitos e utilizados pela comunidade de desenvolvimento foram adotados, como: Padrão MVC, VO e DAO.

8.5.1 VERSÃO DO SISTEMA

Ao termino da implementação e teste de uma iteração, se tem a versão do sistema. Após a reunião de acompanhamento, a TAA deve ser preenchida no campo *status* que podem assumir os valores de: **C – Concluído**; **D – em Desenvolvimento**; **A – Abortada**, junto com o tempo real de desenvolvimento de cada iteração. Enquanto houver releases ou iterações a serem desenvolvidas, o processo se repete até o final de todo o preenchimento da TAA.

8.5.2 ROTEIRO DE ATIVIDADES

De acordo com a metodologia, alguns usuários são recrutados a utilizar o sistema de acordo com o perfil do cliente a fim de identificar falhas de interação entre o usuário e a interface do sistema. Este roteiro de atividades utilizado no Projeto SysFarma pode ser conferido na tabela abaixo.

Neste teste de usabilidade, você assumirá as funções de um agente da secretaria de saúde do município de Florianópolis, e atuará como usuário da SysFarma. Este sistema permite automatizar todo o processo de saída de medicamentos. Como agente de saúde você terá acesso a todos as funcionalidades do sistema.

OBS.:Sinta-se à vontade para consultar caso precise de ajuda encontrando alguma dificuldade que comprometa a realização da atividade.

Atividade 01: Efetuar login no sistema

Roteiro: Nesta atividade você irá entrar no sistema SysFarma, através de login e senha.

Instruções:

- Abra o browser de sua preferência;
- Carregue a página do SysFarma (www.jsysfarma.com);
- Informe o seu login, sua senha;
- Acesse o sistema;
- Verifique se está na página principal do SysFarma.

Atividade 02: Cadastrar novos medicamentos

Roteiro: Nesta atividade, você irá cadastrar novos medicamentos que a unidade irá distribuir.

Instruções:

- Cadastre o medicamento DESCONGEX PLUS preenchendo as informações requeridas no formulário correspondente;
    - DESCONGEX PLUS, maleato de bronfeniramina 12mg, cloridrato de fenilefrina 15 mg
- Verifique se os dados foram cadastrados corretamente;
- Volte para a tela principal do SysFarma.

Atividade 03: Localizar e Editar os dados de um medicamento

Roteiro: Nesta atividade, você irá buscar por medicamentos (previamente cadastrados no sistema) e irá editar os dados referentes ao medicamento.

Instruções:

- Procure pelo registro de DESCONGEX PLUS;
- Identifique e selecione o registro requerido;
- Altere os dados referente ao medicamento de DESCONGEX PLUS
    - Contém 12 comprimidos;
- Verifique se as alterações foram efetuadas corretamente;
- Volte para a tela principal do SysFarma.

...

Tabela 27 – Roteiro de Atividades (Teste de Usabilidade)

8.5.3 QUESTIONÁRIO PÓS-TESTE

Após a execução do roteiro de atividades, o usuário deve responder um questionário pós-teste (Tabela 14) a fim de indicar o seu grau de satisfação ao utilizar o produto. Questionário no qual foi construído com base o exemplo utilizado pela própria metodologia.

Pesquisa de Satisfação do Usuário
Uso e Navegação do Sistema.. Respostas: Muito Fácil – Fácil – Nem Fácil Nem Difícil – Difícil – Muito Difícil
Uso do produto na realização de tarefas de interesse;
Comunicação com o produto (terminologia, linguagem, realimentação da informação e das ações em geral);
Comunicação/Localização dos itens de menu associados às tarefas;
Visualização das instruções e advertências do produto;
Compreensão das instruções e advertências do produto;
Navegação pelas janelas de diálogo do produto;
Compreensão dos termos e solicitações apresentadas nas janelas de diálogo do produto;
Recuperação de situações de erro;
Recuperação de situações de travamento;
Compreensão das mensagens de erro apresentadas;
Navegação através das diferentes opções do menu, janelas de diálogo e barras de ícones do produto;
Uso das funcionalidades mais comuns do produto;
Compreensão da estruturação dos menus, barras de ícones ou listas de informações disponibilizadas pelo produto;

Processo de entrada e saída de dados durante o uso do produto.

Tabela 28 – Questionário Pós-Teste

8.6 REUNIÕES DE ACOMPANHAMENTO

Todo o processo de desenvolvimento é acompanhado pelo gerente através de Reuniões de Acompanhamento semanal. Nesta reunião se faz o uso de *Big Chart* que é gerado toda semana para acompanhamento das tarefas e da tabela de riscos.

8.6.1 BIG CHART

O *Big Chart*, que é apresentado no item 3.7.1 deste trabalho é apresentado abaixo.

Acompanhamento da Coleta de Métricas							
Data	Classes	Scripts	Testes de Aceitação	Testes de Unidade	Páginas *	User Stories	Observações
24/09/07	9	0	0	0	0	0	Implementação do modelo de persistência VO/DAO e conexão.
25/09/07	11	0	0	0	6	0	Inclusão e consulta de Usuários.
26/09/07	12	0	0	0	7	1	Autenticação (login)
27/09/07	14	0	0	0	8	0	Inclusão e consulta de Medicamentos
28/09/07	14	0	0	0	8	2	Edição de Usuário e Medicamento.

Tabela 29 – *Big Chart*

8.6.2 TAA COMPLETA

Ao final de cada iteração, a TAA é preenchida conforme a tabela abaixo.

Iteração 01 – (24/09/2007 – 28/09/2007)	
US01 - Programar funcionalidades de LOGIN	
Teste de Aceitação	***Status***

TA1.1	Acessar o sistema a partir de um login válido e ativo (autenticação feita com sucesso)	C
TA1.2	Acessar o sistema a partir de um login inválido ou inativo (mensagem de erro deve ser exibida)	C
TA1.3	A partir de um usuário administrador cadastrar novo usuário com todos os campos obrigatórios (Cadastro efetuado com sucesso)	C
TA1.4	A partir de um usuário administrador cadastrar novo usuário sem informar todos os campos obrigatórios (Cadastro não deve ser efetuado)	C
TA1.5	A partir de um usuário administrador alterar dados de um usuário cadastrado (Alteração efetuada com sucesso)	C

Atividade	Descrição	Responsável	Estimativa de tempo (horas)	Tempo real (horas)	*Status*
A1.1	Implementar interface de Login	Diogo	4	4	*C*
A1.2	Gerar script para configuração do MySQL	Diogo	2	2	*C*
A1.3	Instalar aplicação em Servidor	Diogo	1	1	*C*
A1.4	Gerar script dos testes de aceitação	Diogo	1	1	*C*

Tabela 30 – TAA 1 completa

Iteração 02 – (24/09/2007 – 28/09/2007)		
US02 - Programar funcionalidades de cadastro de medicamentos		
Teste de Aceitação		***Status***
TA2.1	Cadastrar medicamentos da unidade informando todos os campos obrigatórios (Cadastro efetuado com sucesso)	C
TA2.2	Cadastrar medicamentos da unidade sem informar todos os campos obrigatórios (Cadastro não deve ser efetuado)	C
TA2.3	Alterar dados de um medicamento cadastrado a partir do seu código ou descrição (Cadastro alterado com Sucesso);	C
TA2.4	Excluir um medicamento cadastrado a partir de seu código ou descrição e que tenham movimentação (Mensagem de erro deve ser exibida);	A
TA2.5	Excluir um medicamento cadastrado a partir de seu código ou descrição e que não tenham movimentação (Exclusão efetuado com sucesso);	C

Atividade	Descrição	Responsável	Estimativa de tempo (horas)	Tempo real (horas)	*Status*
A1.1	Implementar interface para cadastro dos medicamentos	Diogo	4	4	*C*
A1.2	Gerar script para configuração do MySQL	Diogo	2	2	*C*
A1.3	Instalar aplicação em Servidor	Diogo	1	1	*C*
A1.4	Gerar script dos testes de aceitação	Diogo	1	1	*C*

Tabela 31 – TAA 2 completa

8.6.3 ANÁLISE DE RISCOS

A tabela abaixo representa os riscos identificados no Projeto SysFarma. Como os riscos listados pode classificar a prioridade de superação e se a solução foi superada.

Data	Risco	Prioridade	Responsável	Status	Providência/Solução
24/09	Uso da tecnologia desconhecida (DAO, VO)	Alta	Todos	Superado	Estudar o assunto, considerando o tempo necessário no plano de iteração.
25/09	Utilizar tecnologia Hibernate (desconhecida) ao sistema	Média	Todos	Superado	Foi pesquisado e consultado sobre o assunto com pessoas que à dominam.
25/09	Utilização de JSF ao invés de JSP.	Alta	Todos	Superado	Foi pesquisado e consultado sobre o assunto com pessoas que à dominam.

Tabela 32 – Análise de Riscos

Observações:

- Data da identificação do Risco;
- Prioridade: Alta, Média, Baixa;
- Status: Vigente, Superado, Abortado.

9 ANEXO B – MODELAGEM SCRUM

9.1 PREPARAÇÃO

Como as demais metodologias, no Scrum é preciso fazer uma análise, uma preparação para o projeto onde um documento conterá a visão do projeto. Informações breves e precisas do que o projeto será a fim de criar o *Product Backlog* e para consultas eventuais.

Abaixo, uma breve visão do projeto SysFarma:

O SysFarma é um sistema que servirá como um controle de estoque de farmácia das unidades de saúde. Um sistema que ajudará no processo de reposição de estoques dos medicamentos e para se ter um melhor planejamento a fim de se evitar erros de estoque.

O sistema contará com um controle de usuários onde haverá um usuário administrador geral que fará o controle de usuários no sistema e usuários que farão as entradas e saídas dos medicamentos.

Com o sistema, o usuário poderá efetuar pesquisas por princípio ativo ou sua descrição e gerar relatórios da situação geral dos medicamentos.

O maior objetivo do SysFarma, é ter um controle fiel de seus medicamentos disponíveis, quantos medicamentos estão em estoque e quantos medicamentos a comunidade usufruiu.

9.1.1 PRODUCT BACKLOG

Como descrito no item 4.4.1 deste trabalho, a função do *Product Backlog* é de listar todas as funcionalidades do projeto. A tabela abaixo apresenta todas as funcionalidades do projeto com suas respectivas prioridades.

Prioridade	Item	Descrição	Tempo
Muito Alta			

	1	Criação do modelo de dados e infra-estrutura	8
	2	Módulo de Autorização	8
	3	Cadastro de usuários	8
	4	Cadastro de Medicamentos	8
Alta			
	5	Funcionalidades de entrada de medicamentos	8
	6	Funcionalidades de saída de medicamentos	8
Média			
	7	Funcionalidades de Pesquisa de medicamentos	8
	8	Funcionalidades de Relatório	8

Tabela 33 – Product Backlog

9.2 PAPÉIS

Na modelagem *Scrum*, o projeto é desenvolvido por três principais papéis. São eles *Product Owner*, *Team Members* e *ScrumMaster* que pode ser conferido na Tabela.

MEMBRO	PAPÉIS
Diogo	Product Owner, Team Members
Jonatas	ScrumMaster, Team Members

Tabela 34 – Equipe Scrum

Por se tratar de um projeto acadêmico, com poucas pessoas envolvidas, cada membro desenvolveu mais de um papel no projeto.

9.3 PROCESSO

Com o *Product Backlog* definido, parte-se para a fase de planejamento do *Sprint* onde se define quais funcionalidades farão parte do primeiro *Sprint Backlog*. No Scrum a reunião de planejamento consiste em duas partes. Na primeira parte (*Sprint Planning 1*), o *Product Owner* apresenta a lista de prioridades do *Product Backlog* e a equipe discute que funcionalidades podem desenvolver até o próximo

Sprint. Com isso foi construído uma tabela 3 com as funcionalidades selecionadas pelo *Product Owner* juntamente com a equipe para o *Sprint* 1.

Itens Selecionados				
Prioridade	Item	Descrição	Tempo	Responsável
Muito Alta				
	1	Criação do modelo de dados e infra-estrutura	8	Diogo
	2	Módulo de Autorização	8	Diogo
	3	Cadastro de usuários	8	Diogo
	4	Cadastro de Medicamentos	8	Jonatas

Tabela 35 – Itens do Product Backlog selecionadas para o Sprint 1.

Tendo as funcionalidades selecionadas, a segunda parte do planejamento (*Sprint Planning 2*), consiste em dividir as tarefas até o próximo *Sprint*.

Sprint Backlog					
Prioridade	Item	Itens do Product Backlog	Tempo	Itens Sprint Backlog	Res.
Muito Alta					
	1.	Criação do modelo de dados e infra-estrutura	4	1 - Criação do modelo de dados;	Diogo
			4	2 - Configuração da Infra-estrutura.	Diogo
	2.	Modulo Autorização	4	1 - Acessar o sistema a partir de um login válido e ativo (autenticação feita com sucesso);	Diogo
			4	2 - Acessar o sistema a partir de um login inválido ou inativo (mensagem de erro deve ser exibida)	Diogo
	3.	Cadastro de Usuários	4	1 - A partir de um usuário administrador cadastrar novo usuário com todos os campos obrigatórios (Cadastro efetuado com sucesso);	Diogo
			2	2 - A partir de um usuário administrador cadastrar novo usuário sem informar todos os	Diogo

				campos obrigatórios (Cadastro não deve ser efetuado);	
			2	3 - A partir de um usuário administrador alterar dados de um usuário cadastrado (Alteração efetuada com sucesso).	Diogo
	4.	Cadastro de Medicamentos	2	1 - Cadastrar medicamentos da unidade informando todos os campos obrigatórios (Cadastro efetuado com sucesso);	Jonatas
			1	2 - Cadastrar medicamentos da unidade sem informar todos os campos obrigatórios (Cadastro não deve ser efetuado);	Jonatas
			2	3 - Alterar dados de um medicamento cadastrado a partir do seu código ou descrição (Cadastro alterado com Sucesso);	Jonatas
			2	4 - Excluir um medicamento cadastrado a partir de seu código ou descrição e que tenham movimentação (Mensagem de erro deve ser exibida);	Jonatas
			1	5 - Excluir um medicamento cadastrado a partir de seu código ou descrição e que não tenham movimentação (Exclusão efetuado com sucesso).	Jonatas

Tabela 36 – *Sprint Backlog*

Finalizada o planejamento do *Sprint*, parte-se para o primeiro *Sprint* onde serão desenvolvidas as funcionalidades que foram destacadas no *Sprint Backlog*. No primeiro Sprint, foi realizada a criação do modelo de dados e a configuração da infra-estruta. Itens não detectados durante a primeira fase do planejamento do Sprint Backlog e atualizados no Product Backlog.

Durante o período do *Sprint*, são feitas reuniões diárias onde a equipe pode expressar dificuldades que estão enfrentando durante o dia-a-dia do *Sprint*.

9.3.1 SPRINT

Devido ao tempo, e por ser um projeto de estudo, o *Sprint* foi realizado em um intervalo de tempo de uma semana. Durante estes dias foram realizadas as tarefas incluídas no *Sprint Backlog*, e este andamento pode ser visto de acordo com a figura 25 abaixo:

Sprint 1

24/9/2007			Dias do Sprint		24 Seg	25 Ter	26 Qua	27 Qui	28 Sex
	5 dias de trabalho no Sprint		Horas restantes		25	18	10	4	0
	Product Backlog Item	Product Backlog	Responsável	Estimativa					
1	1	Criação do modelo de dados e infra-estrutura	Diogo	8	3	1	0		
2	2	Modulo Autorização	Diogo	8	8	6	2	0	
3	3	Cadastro de Usuários	Diogo	8	7	5	2	2	0
4	4	Cadastro de Medicamentos	Jonatas	8	7	6	6	2	0

Figura 25 – Sprint.

Ao final do *Sprint* foi gerado o gráfico *Sprint Burndown*, figura 26, que mostra como foi à execução das tarefas durante o *Sprint*.

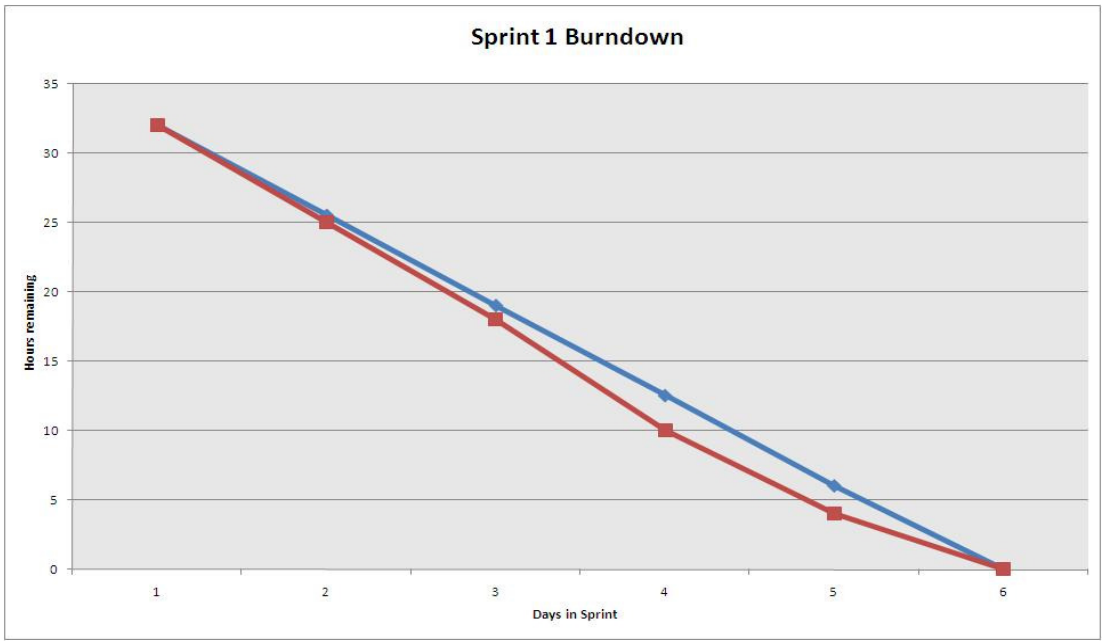


Figura 26 – Sprint Burndown.

9.3.2 REUNIÃO DIÁRIA DO *SCRUM*

Abaixo um exemplo das reuniões diárias que aconteceu durante o Sprint.

Reunião do dia 24/09/2007.

1 - O que você fez desde o último encontro?

Criação do modelo de dados, definição da arquitetura e inicio da criação das entidades do projeto.

2 - Quais foram as suas dificuldades durante o trabalho?

Dificuldade com o modelo de implementação em JAVA.

3 - Quais atividades você pretende fazer até o próximo encontro?

Concluir o cadastro de Usuários e terminar a entidade de medicamentos.

9.3.3 REVISÃO DO SPRINT

Na revisão do *Sprint* foi constatado que a maioria das tarefas foi realizada no tempo estimado. Ficando somente um item em aberto: “4 - Excluir um medicamento cadastrado a partir de seu código ou descrição e que tenham movimentação (Mensagem de erro deve ser exibida);”.

Foi decidido no andamento do *Sprint* que esta funcionalidade seria melhor elabora após a implementação das movimentações de entrada e saída dos medicamentos, ficando assim para ser executada no *Sprint* correspondente.

Além disso, foi constatada uma situação que não foi prevista na elaboração das tarefas, que é o impedimento de se cadastrar Usuários com o atributo usuario igual. O problema foi resolvido sem maiores complicações não evidenciando qualquer atraso no cronograma do *Sprint*.

9.3.4 RETROSPECTIVA DO SPRINT

Na reunião de retrospectiva do *Sprint* foi acordado entre a equipe que as mensagens de validações e erros do sistema devem ser mais bem detalhadas e

definidas para não haver pausas e atrasos na implementação devido à falta destas definições.

10 ANEXO C – CÓDIGO FONTE

- ✓ Facesconfig.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<faces-config version="1.2" xmlns="http://java.sun.com/xml/ns/javaee"
 xmlns:xi="http://www.w3.org/2001/XInclude"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-facesconfig_1_2.xsd">
 <managed-bean>
  <managed-bean-name>usuarioMb</managed-bean-name>
  <managed-bean-class>br.com.sysfarma.mb.UsuarioMb</managed-bean-class>
  <managed-bean-scope>session</managed-bean-scope>
 </managed-bean>
 <managed-bean>
  <managed-bean-name>medicamentoMb</managed-bean-name>
  <managed-bean-class>br.com.sysfarma.mb.MedicamentoMb</managed-bean-class>
  <managed-bean-scope>session</managed-bean-scope>
 </managed-bean>
 <navigation-rule>
  <from-view-id>*</from-view-id>
  <navigation-case>
   <from-outcome>lista_usuarios</from-outcome>
   <to-view-id>/listaUsuarios.jsp</to-view-id>
  </navigation-case>
  <navigation-case>
   <from-outcome>cadastra_usuario</from-outcome>
   <to-view-id>/cadastraUsuario.jsp</to-view-id>
  </navigation-case>
  <navigation-case>
   <from-outcome>cadastra_medicamento</from-outcome>
   <to-view-id>/cadastraMedicamento.jsp</to-view-id>
  </navigation-case>
  <navigation-case>
   <from-outcome>lista_medicamentos</from-outcome>
   <to-view-id>/listaMedicamentos.jsp</to-view-id>
  </navigation-case>
  <navigation-case>
   <from-outcome>principal</from-outcome>
   <to-view-id>/principal.jsp</to-view-id>
  </navigation-case>
  <navigation-case>
   <from-outcome>login</from-outcome>
   <to-view-id>/login.jsp</to-view-id>
  </navigation-case>
 </navigation-rule>
</faces-config>
```

- ✓ Web.xml

```
<?xml version="1.0"?>
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
 <display-name>sysfarma</display-name>
 <context-param>
  <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
  <param-value>server</param-value>
 </context-param>
 <listener>
  <listener-class>com.sun.faces.config.ConfigureListener</listener-class>
 </listener>
 <!-- Faces Servlet -->
 <servlet>
  <servlet-name>Faces Servlet</servlet-name>
  <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
  <load-on-startup>1</load-on-startup>
```

```
</servlet>
<!-- Faces Servlet Mapping -->
<servlet-mapping>
 <servlet-name>Faces Servlet</servlet-name>
 <url-pattern>*.jsf</url-pattern>
</servlet-mapping>

<filter>
        <filter-name>SysFarmaFilter</filter-name>
        <filter-class>br.com.sysfarma.filter.SysFarmaFilter</filter-class>
</filter>
<filter-mapping>
        <filter-name>SysFarmaFilter</filter-name>
        <servlet-name>Faces Servlet</servlet-name>
</filter-mapping>

<context-param>
        <param-name>org.ajax4jsf.SKIN</param-name>
        <param-value>default</param-value>
</context-param>
<filter>
        <display-name>Ajax4jsf Filter</display-name>
        <filter-name>ajax4jsf</filter-name>
        <filter-class>org.ajax4jsf.Filter</filter-class>
</filter>
<filter-mapping>
        <filter-name>ajax4jsf</filter-name>
        <servlet-name>Faces Servlet</servlet-name>
        <dispatcher>REQUEST</dispatcher>
        <dispatcher>FORWARD</dispatcher>
        <dispatcher>INCLUDE</dispatcher>
</filter-mapping>
<login-config>
 <auth-method>BASIC</auth-method>
 </login-config>
</web-app>
```

- ✓ Persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="1.0" xmlns="http://java.sun.com/xml/ns/persistence" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd">
        <persistence-unit name="sysfarma" transaction-type="RESOURCE_LOCAL">
                <properties>
                <property name="hibernate.connection.url" value="jdbc:mysql://localhost/sysfarma"/>
                        <property name="hibernate.connection.driver_class" value="com.mysql.jdbc.Driver"/>
                        <property name="hibernate.dialect" value="org.hibernate.dialect.MySQLDialect"/>
                        <property name="hibernate.cache.provider_class"
value="org.hibernate.cache.NoCacheProvider"/>
                        <property name="hibernate.connection.username" value="admin"/>
                        <property name="hibernate.connection.password" value="admin"/>
                        <property name="hibernate.show_sql" value="true"/>
                        <property name="hibernate.hbm2ddl.auto" value="update"/>
                </properties>
        </persistence-unit>
</persistence>
```

- ✓ AbstractDao.java

```
package br.com.sysfarma.dao;

import java.io.Serializable;
import java.util.List;

import javax.persistence.EntityManager;
import javax.persistence.EntityTransaction;
import javax.persistence.Query;
```

```
import br.com.sysfarma.entity.PersistentEntity;

public class AbstractDao<T extends PersistentEntity<ID>, ID extends Serializable>
		implements Dao<T, ID> {

	private final Class<T> typeClazz;

	private final String findAllEjbQL;

	public AbstractDao(Class<T> typeClazz) {
		if (typeClazz == null)
			throw new IllegalArgumentException("TypeClass is null");
		this.typeClazz = typeClazz;
		this.findAllEjbQL = "select t from " + typeClazz.getName() + " t";
	}

	public void delete(ID id) {
		if (id == null)
			throw new IllegalArgumentException("Id is null");
		EntityManager em = Manager.getEntityManager();
		EntityTransaction transaction = em.getTransaction();
		try {
			transaction.begin();
			T o = findByPk(id);
			if (o == null)
				throw new IllegalStateException("Object with id " + id
						+ " not found");
			o = em.merge(o);
			em.remove(o);
			transaction.commit();
		} catch (Exception e) {
			if (transaction != null && transaction.isActive())
				transaction.rollback();
			throw new RuntimeException(e);
		} finally {
			if (em.isOpen())
				em.close();
		}
	}

	@SuppressWarnings("unchecked")
	public List<T> findAll() {
		EntityManager em = Manager.getEntityManager();
		Query query = em.createQuery(findAllEjbQL);
		return query.getResultList();
	}

	public T findByPk(ID id) {
		EntityManager em = Manager.getEntityManager();
		T t = em.find(typeClazz, id);
		return t;
	}

	public void persist(T t) {
		if (t == null)
			throw new IllegalArgumentException("Type is null");
		EntityManager em = Manager.getEntityManager();
		EntityTransaction transaction = em.getTransaction();
		try {
			transaction.begin();
			em.persist(t);
			transaction.commit();
		} catch (Exception e) {
			if (transaction != null && transaction.isActive())
				transaction.rollback();
			throw new RuntimeException(e);
		} finally {
			if (em.isOpen())
				em.close();
		}
	}

	public void update(T t) {
```

```
                if (t == null)
                        throw new IllegalArgumentException("Type is null");
                EntityManager em = Manager.getEntityManager();
                EntityTransaction transaction = em.getTransaction();
                try {
                        transaction.begin();
                        em.merge(t);
                        transaction.commit();
                } catch (Exception e) {
                        if (transaction != null && transaction.isActive())
                                transaction.rollback();
                        throw new RuntimeException(e);
                } finally {
                        if (em.isOpen())
                                em.close();
                }
        }

}
```

- ✓ Dao.java

```
package br.com.sysfarma.dao;

import java.io.Serializable;
import java.util.List;

import br.com.sysfarma.entity.PersistentEntity;

public interface Dao<T extends PersistentEntity<ID>, ID extends Serializable> {

        void persist(T t);

        void delete(ID id);

        void update(T t);

        List<T> findAll();

        T findByPk(ID id);

}
```

- ✓ DaoFactory.java

```
package br.com.sysfarma.dao;

public class DaoFactory {

        public static UsuarioDao getUsuarioDao() {
                return new UsuarioDaoImpl();
        }

        public static MedicamentoDao getMedicamentoDao(){
                return new MedicamentoDaoImpl();
        }

}
```

- ✓ Manager.java

```
package br.com.sysfarma.dao;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
```

```
public class Manager {

private static ThreadLocal<EntityManager> tl = new ThreadLocal<EntityManager>();

public static EntityManager getEntityManager() {
EntityManager em;
em = tl.get();
if (em == null || !em.isOpen()) {
EntityManagerFactory emf = Persistence
.createEntityManagerFactory("sysfarma");
em = emf.createEntityManager();
tl.set(em);
}
return em;
}

}
```

- ✓ MedicamentoDao.java

```
package br.com.sysfarma.dao;

import br.com.sysfarma.entity.Medicamento;

public interface MedicamentoDao  extends Dao<Medicamento, Long>{


}
```

- ✓ MedicamentoDaoImpl

```
package br.com.sysfarma.dao;

import br.com.sysfarma.entity.Medicamento;

public class MedicamentoDaoImpl extends AbstractDao<Medicamento, Long> implements MedicamentoDao {

public MedicamentoDaoImpl() {
super(Medicamento.class);
// TODO Auto-generated constructor stub
}


}
```

- ✓ UsuarioDao.java

```
package br.com.sysfarma.dao;

import br.com.sysfarma.entity.Usuario;

public interface UsuarioDao extends Dao<Usuario, Long> {

Usuario findUsuarioByLogin(String usuario, String senha);
Boolean findUsuarioByUsuario(String usuario);


}
```

- ✓ UsuarioDaoImpl.java

```
package br.com.sysfarma.dao;
```

```
import javax.persistence.EntityManager;
import javax.persistence.Query;

import br.com.sysfarma.entity.Usuario;

public class UsuarioDaoImpl extends AbstractDao<Usuario, Long> implements
		UsuarioDao {

	public UsuarioDaoImpl() {
		super(Usuario.class);
	}

	public Usuario findUsuarioByLogin(String usuario, String senha) {
		EntityManager em = Manager.getEntityManager();
		Query q = em.createQuery("select u from usuario u where usuario = :usuario and senha = :senha and
situacao = :situacao");
		q.setParameter("usuario", usuario);
		q.setParameter("senha", senha);
		q.setParameter("situacao", "A");
		return (Usuario) q.getSingleResult();
	}

	public Boolean findUsuarioByUsuario(String usuario) {
		EntityManager em = Manager.getEntityManager();
		Query q = em.createQuery("select u from usuario u where usuario = :usuario");
		q.setParameter("usuario", usuario);
		return (q.getResultList().size() > 0);
	}


}
```

- ✓ Medicamento.java

```
package br.com.sysfarma.entity;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity(name = "medicamento")
public class Medicamento implements PersistentEntity<Long> {

	@Id
	@Column(name = "id_medicamento", nullable = false)
	@GeneratedValue(strategy = GenerationType.AUTO)
	private Long id;

	@Column(name = "codigo", nullable = false)
	private String codigo;
	@Column(name = "descricao", nullable = false)
	private String descricao;
	@Column(name = "fabricante", nullable = false)
	private String fabricante;
	@Column(name = "principio_ativo", nullable = false)
	private String principioAtivo;
	@Column(name = "apresentacao", nullable = false)
	private String apresentacao;

	public Long getId() {
		return id;
	}
	public void setId(Long id) {
		this.id = id;
	}
	public String getCodigo() {
		return codigo;
	}
```

```
	public void setCodigo(String codigo) {
		this.codigo = codigo;
	}
	public String getDescricao() {
		return descricao;
	}
	public void setDescricao(String descricao) {
		this.descricao = descricao;
	}
	public String getFabricante() {
		return fabricante;
	}
	public void setFabricante(String fabricante) {
		this.fabricante = fabricante;
	}
	public String getPrincipioAtivo() {
		return principioAtivo;
	}
	public void setPrincipioAtivo(String principioAtivo) {
		this.principioAtivo = principioAtivo;
	}
	public String getApresentacao() {
		return apresentacao;
	}
	public void setApresentacao(String apresentacao) {
		this.apresentacao = apresentacao;
	}

}
```

✓ PersistentEntity.java

```
package br.com.sysfarma.entity;

import java.io.Serializable;

public interface PersistentEntity<ID extends Serializable> extends Serializable {


}
```

✓ Usuario.java

```
package br.com.sysfarma.entity;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity(name = "usuario")
public class Usuario implements PersistentEntity<Long> {

	@Id
	@Column(name = "id_usuario", nullable = false)
	@GeneratedValue(strategy = GenerationType.AUTO)
	private Long id;
	@Column(name = "tipo", nullable = false)
	private Character tipo;
	@Column(name = "situacao", nullable = false)
	private Character situacao;
	@Column(name = "nome", nullable = false)
	private String nome;
	@Column(name = "usuario", nullable = false)
	private String usuario;
	@Column(name = "senha", nullable = false)
```

```
	private String senha;

	public Long getId() {
		return id;
	}

	public void setId(Long id) {
		this.id = id;
	}

	public Character getTipo() {
		return tipo;
	}

	public void setTipo(Character tipo) {
		this.tipo = tipo;
	}

	public Character getSituacao() {
		return situacao;
	}

	public void setSituacao(Character situacao) {
		this.situacao = situacao;
	}

	public String getNome() {
		return nome;
	}

	public void setNome(String nome) {
		this.nome = nome;
	}

	public String getUsuario() {
		return usuario;
	}

	public void setUsuario(String usuario) {
		this.usuario = usuario;
	}

	public String getSenha() {
		return senha;
	}

	public void setSenha(String senha) {
		this.senha = senha;
	}

	public String getSituacaoConv() {
		if (situacao == 'A')
			return "Ativo";
		else
			return "Inativo";
	}

	public String getTipoConv() {
		if (tipo == 'A')
			return "Administrador";
		else
			return "Normal";
	}

}
```

- ✓ SysFarmaFilter.java

```
	public void destroy() {
	}

	public void init(FilterConfig arg0) throws ServletException {
	}

	public void doFilter(ServletRequest request, ServletResponse response,
			FilterChain filter) throws IOException, ServletException {
		HttpServletRequest req = ((HttpServletRequest) request);
		HttpSession s = req.getSession();
		Usuario u = (Usuario) s.getAttribute("usuario");
		String uri = req.getRequestURI();
		if (u != null) {
			if ((uri.indexOf("cadastraUsuario.jsf") > 0 || uri
					.indexOf("listaUsuarios.jsf") > 0)
					&& u.getTipo() != 'A')
				((HttpServletResponse) response)
						.sendRedirect("principal.jsf");
			else
				filter.doFilter(request, response);
		} else {
			if (uri.indexOf("login.jsf") > 0)
				filter.doFilter(request, response);
			else
				((HttpServletResponse) response).sendRedirect("login.jsf");
		}
	}
```

- ✓ MedicamentoMb.java

```
package br.com.sysfarma.mb;

import java.util.List;

import javax.faces.application.FacesMessage;
import javax.faces.context.FacesContext;

import org.richfaces.component.html.HtmlDataTable;

import br.com.sysfarma.dao.DaoFactory;
import br.com.sysfarma.entity.Medicamento;

public class MedicamentoMb {

	private Medicamento medicamento;

	private HtmlDataTable datatable;

	private boolean update;

	List<Medicamento> listaMedicamentos;

	public void actionExcluir() {
		try {
			medicamento = (Medicamento) getDatatable().getRowData();
			DaoFactory.getMedicamentoDao().delete(medicamento.getId());
			listaMedicamentos = null;
		} catch (Exception e) {
			FacesContext.getCurrentInstance().addMessage(null,
					new FacesMessage(e.getMessage()));
		}
		medicamento = null;
	}

	public String actionIncluir() {
		try {
			DaoFactory.getMedicamentoDao().persist(medicamento);
			listaMedicamentos = null;
			return "lista_medicamentos";
		} catch (Exception e) {
			FacesContext.getCurrentInstance().addMessage(null,
```

```
                                        new FacesMessage(e.getMessage()));
                        return null;
                }
        }

        public String actionEditar() {
                try {
                        DaoFactory.getMedicamentoDao().update(medicamento);
                        listaMedicamentos = null;
                        return "lista_medicamentos";
                } catch (Exception e) {
                        FacesContext.getCurrentInstance().addMessage(null,
                                        new FacesMessage(e.getMessage()));
                        return null;
                }
        }

        public String goEditar() {
                System.out.println("foii.....");
                medicamento = (Medicamento) getDatatable().getRowData();
                medicamento = DaoFactory.getMedicamentoDao().findByPk(
                                medicamento.getId());
                update = true;
                return "cadastra_medicamento";
        }

        public String goInserir() {
                medicamento = null;
                update = false;
                return "cadastra_medicamento";
        }

        public List<Medicamento> getListaMedicamentos() {
                if (listaMedicamentos == null) {
                        listaMedicamentos = DaoFactory.getMedicamentoDao().findAll();
                }
                return listaMedicamentos;
        }

        public Medicamento getMedicamento() {
                if (medicamento == null) {
                        medicamento = new Medicamento();
                }
                return medicamento;
        }

        public void setMedicamento(Medicamento medicamento) {
                this.medicamento = medicamento;
        }

        public HtmlDataTable getDatatable() {
                return datatable;
        }

        public void setDatatable(HtmlDataTable datatable) {
                this.datatable = datatable;
        }

        public boolean isUpdate() {
                return update;
        }

        public void setUpdate(boolean update) {
                this.update = update;
        }

}
```

- ✓ UsuarioMb.java

```
package br.com.sysfarma.mb;

import java.util.List;

import javax.faces.application.FacesMessage;
import javax.faces.context.FacesContext;
import javax.servlet.http.HttpSession;

import org.richfaces.component.html.HtmlDataGrid;

import br.com.sysfarma.dao.DaoFactory;
import br.com.sysfarma.dao.UsuarioDao;
import br.com.sysfarma.entity.Usuario;

public class UsuarioMb {

	private Usuario usuario;
	private Usuario usuarioLogin;
	private HtmlDataGrid dataGrid;
	boolean update;

	List<Usuario> listaUsuarios;

	public String actionLogout() {
		HttpSession s = (HttpSession) FacesContext.getCurrentInstance()
				.getExternalContext().getSession(false);
		s.removeAttribute("usuario");
		return "login";
	}

	public String actionLogin() {
		try {
			usuarioLogin = DaoFactory.getUsuarioDao().findUsuarioByLogin(
					usuario.getUsuario(), usuario.getSenha());
			HttpSession s = (HttpSession) FacesContext.getCurrentInstance()
					.getExternalContext().getSession(false);
			s.setAttribute("usuario", usuarioLogin);
			usuario = null;
			return "principal";
		} catch (Exception e) {
			FacesContext
					.getCurrentInstance()
					.addMessage(
							null,
							new FacesMessage(
									"Usuário/Senha incorretos ou
inexistentes, tente novamente!"));
			return null;
		}
	}

	public String actionIncluir() {
		try {
			Boolean existe = DaoFactory.getUsuarioDao().findUsuarioByUsuario(
					usuario.getUsuario());
			if (!existe) {
				DaoFactory.getUsuarioDao().persist(usuario);
				listaUsuarios = null;
				usuario = null;
				return "lista_usuarios";
			} else
				return null;
		} catch (Exception e) {
			FacesContext.getCurrentInstance().addMessage(null,
					new FacesMessage(e.getMessage()));
			return null;
		}
	}

	public String actionEditar() {
		try {
			UsuarioDao dao = DaoFactory.getUsuarioDao();
			if (usuario.getSenha() == null
```

```
                                        || usuario.getSenha().trim().equals("")) {
                                Usuario old = dao.findByPk(usuario.getId());
                                usuario.setSenha(old.getSenha());
                        }
                        dao.update(usuario);
                        listaUsuarios = null;
                        return "lista_usuarios";
                } catch (Exception e) {
                        FacesContext.getCurrentInstance().addMessage(null,
                                        new FacesMessage(e.getMessage()));
                        return null;
                }
        }

        public String goEditar() {
                usuario = (Usuario) getDataGrid().getRowData();
                usuario = DaoFactory.getUsuarioDao().findByPk(usuario.getId());
                update = true;
                return "cadastra_usuario";
        }

        public String goInserir() {
                usuario = null;
                update = false;
                return "cadastra_usuario";
        }

        public List<Usuario> getListaUsuarios() {
                if (listaUsuarios == null) {
                        listaUsuarios = DaoFactory.getUsuarioDao().findAll();
                }
                return listaUsuarios;
        }

        public Usuario getUsuario() {
                if (usuario == null) {
                        usuario = new Usuario();
                }
                return usuario;
        }

        public void setUsuario(Usuario usuario) {
                this.usuario = usuario;
        }

        public Usuario getUsuarioLogin() {
                return usuarioLogin;
        }

        public void setUsuarioLogin(Usuario usuarioLogin) {
                this.usuarioLogin = usuarioLogin;
        }

        public HtmlDataGrid getDataGrid() {
                return dataGrid;
        }

        public void setDataGrid(HtmlDataGrid dataGrid) {
                this.dataGrid = dataGrid;
        }

        public boolean isUpdate() {
                return update;
        }

        public void setUpdate(boolean update) {
                this.update = update;
        }

}
```

✓ CadastraMedicamento.jsp

```
<%@ taglib uri="http://richfaces.ajax4jsf.org/rich" prefix="rich"%>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="https://ajax4jsf.dev.java.net/ajax" prefix="a4j" %>

<f:view>
	<%@include file="menu.jsp" %>
	<h:form>
		<h:panelGrid columns="2">
		<rich:panel style="border:3px; FONT-WEIGHT: bold; BACKGROUND-COLOR: #F1F1F1; width : 631px;">
		<h:panelGrid columns="3" style="FONT-FAMILY: 'SansSerif';">
			<h:outputLabel value="Código" for="codigo"></h:outputLabel>
			<h:inputText id="codigo" value="#{medicamentoMb.medicamento.codigo}" required="true" style="
BACKGROUND-COLOR: #F1F1F1; width : 241px;" requiredMessage="Código é obrigatório!"/>
			<h:message for="codigo" style="COLOR: #ff0000;" />

			<h:outputLabel value="Descrição" for="descricao" ></h:outputLabel>
			<h:inputText id="descricao" value="#{medicamentoMb.medicamento.descricao}" required="true"
style="BACKGROUND-COLOR: #F1F1F1; width : 241px;" requiredMessage="Descrição é obrigatório!"/>
			<h:message for="descricao" style="COLOR: #ff0000;" />

			<h:outputLabel value="Fabricante" for="fabricante"></h:outputLabel>
			<h:inputText id="fabricante" value="#{medicamentoMb.medicamento.fabricante}" required="true"
style="BACKGROUND-COLOR: #F1F1F1; width : 240px;" requiredMessage="Fabricante é obrigatório!"/>
			<h:message for="fabricante" style="COLOR: #ff0000;" />

			<h:outputLabel value="Princípio Ativo" for="principioAtivo"></h:outputLabel>
			<h:inputText id="principioAtivo" value="#{medicamentoMb.medicamento.principioAtivo}"
required="true" style="BACKGROUND-COLOR: #F1F1F1; width : 240px;" requiredMessage="Princípio Ativo é obrigatório!"/>
			<h:message for="principioAtivo" style="COLOR: #ff0000;" />

			<h:outputLabel value="Apresentação" for="apresentacao"></h:outputLabel>
			<h:inputText id="apresentacao" value="#{medicamentoMb.medicamento.apresentacao}"
required="true" style="BACKGROUND-COLOR: #F1F1F1; width : 238px;" requiredMessage="Apresentação é obrigatório!"/>
			<h:message for="apresentacao" style="COLOR: #ff0000;" />
		</h:panelGrid>

		<h:commandButton action="#{medicamentoMb.actionIncluir}" value="Confirmar"
rendered="#{!medicamentoMb.update}"/>
		<h:commandButton action="#{medicamentoMb.actionEditar}" value="Confirmar"
rendered="#{medicamentoMb.update}"/>
		<h:commandButton action="lista_medicamentos" value="Cancelar" immediate="true"/>
		</rich:panel>
	 <h:graphicImage value="/imagens/med.jpg"/>
	 </h:panelGrid>
	</h:form>

</f:view>
```

✓ CadastraUsuario.jsp

```
<%@ taglib uri="http://richfaces.ajax4jsf.org/rich" prefix="rich"%>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="https://ajax4jsf.dev.java.net/ajax" prefix="a4j" %>

<f:view>
	<%@include file="menu.jsp" %>
	<h:form>
	<h:panelGrid columns="2">
		<rich:panel style="border:3px; FONT-WEIGHT: bold; BACKGROUND-COLOR: #F1F1F1; width : 555px;">

		<h:panelGrid columns="3" style="FONT-FAMILY: 'SansSerif';">
			<h:outputLabel value="Nome" for="nome"></h:outputLabel>
			<h:inputText id="nome" value="#{usuarioMb.usuario.nome}" required="true"
style="BACKGROUND-COLOR: #F1F1F1; width : 241px;" requiredMessage="Nome é obrigatório!"/>
			<h:message for="nome" style="COLOR: #ff0000;"/>
```

```
                                    <h:outputLabel value="Usuario" for="usuario"></h:outputLabel>
                                    <h:inputText id="usuario" value="#{usuarioMb.usuario.usuario}" required="true"
style="BACKGROUND-COLOR: #F1F1F1;  width : 241px;" requiredMessage="Usuário é obrigatório!"/>
                                    <h:message for="usuario" style="COLOR: #ff0000;"/>

                                    <h:outputLabel value="Senha" for="senha"></h:outputLabel>
                                    <h:inputSecret id="senha" value="#{usuarioMb.usuario.senha}" required="#{!usuarioMb.update}"
style="BACKGROUND-COLOR: #F1F1F1; width : 241px;" requiredMessage="Senha é obrigatório!"/>
                                    <h:message for="senha" style="COLOR: #ff0000;"/>

                                    <h:outputLabel value="Tipo" for="tipo"></h:outputLabel>
                                    <h:selectOneRadio value="#{usuarioMb.usuario.tipo}" id="tipo" required="true"
requiredMessage="Tipo é obrigatório!">
                                    <f:selectItem itemLabel="Administrador" itemValue="A"/>
                                    <f:selectItem itemLabel="Normal" itemValue="N"/>
                                    </h:selectOneRadio>
                                    <h:message for="tipo" style="COLOR: #ff0000;"/>

                                    <h:outputLabel value="Situação" for="situacao"></h:outputLabel>
                                    <h:selectOneRadio value="#{usuarioMb.usuario.situacao}" id="situacao" required="true"
requiredMessage="Situação é obrigatório!">
                                    <f:selectItem itemLabel="Ativo" itemValue="A"/>
                                    <f:selectItem itemLabel="Inativo" itemValue="I"/>
                                    </h:selectOneRadio>
                                    <h:message for="situacao" style="COLOR: #ff0000;"/>

                        </h:panelGrid>

                        <h:commandButton action="#{usuarioMb.actionIncluir}" value="Confirmar"
rendered="#{!usuarioMb.update}"/>
                        <h:commandButton action="#{usuarioMb.actionEditar}" value="Confirmar"
rendered="#{usuarioMb.update}"/>
                        <h:commandButton action="lista_usuarios" value="Cancelar" immediate="true"/>
                        </rich:panel>
                        <h:graphicImage value="/imagens/user.jpg"/>
                        </h:panelGrid>
            </h:form>

</f:view>
```

✓ Index.jsp

```
<jsp:forward page="login.jsf"></jsp:forward>
```

✓ ListaMedicamentos.jsp

```
<%@ taglib uri="https://ajax4jsf.dev.java.net/ajax" prefix="a4j" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>

<f:view>
<%@include file="menu.jsp" %>

<a4j:form >
<a4j:status startText="Processando..." stopText=""/>
<rich:dataTable cellpadding="0" cellspacing="0"
binding="#{medicamentoMb.datatable}"
id="listaMed" rows="7" value="#{medicamentoMb.listaMedicamentos}" var="medicamento" style=" width : 225px;"
onRowMouseOver="this.style.backgroundColor='#{a4jSkin.tableBackgroundColor}'"
onRowMouseOut="this.style.backgroundColor='#F1F1F1'">
<f:facet name="header">
<rich:columnGroup>
<rich:column>
<h:outputText value="Código" />
</rich:column>
<rich:column>
<h:outputText value="Descrição" />
```

```
</rich:column>
<rich:column>
<h:outputText value="Fabricante" />
</rich:column>
<rich:column>
<h:outputText value="Princípio Ativo" />
</rich:column>
<rich:column>
<h:outputText value="Apresentação" />
</rich:column>
<rich:column>
<h:outputText value="Excluir" />
</rich:column>
<rich:column>
<h:outputText value="Editar" />
</rich:column>
</rich:columnGroup>
</f:facet>
<rich:column>
<h:outputText value="#{medicamento.codigo}" />
</rich:column>
<rich:column>
<h:outputText value="#{medicamento.descricao}"/>
</rich:column>
<rich:column>
<h:outputText value="#{medicamento.fabricante}" />
</rich:column>
<rich:column>
<h:outputText value="#{medicamento.principioAtivo}"/>
</rich:column>
<rich:column>
<h:outputText value="#{medicamento.apresentacao}" />
</rich:column>
<rich:column>
<h:commandButton action="#{medicamentoMb.goEditar}" value="Editar" />
</rich:column>
<rich:column>
<h:commandButton action="#{medicamentoMb.actionExcluir}" value="Excluir" />
</rich:column>
</rich:dataTable>
</a4j:form>

</f:view>
```

- ✓ listaUsuarios.jsp

```
<%@ taglib uri="http://richfaces.ajax4jsf.org/rich" prefix="rich"%>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="https://ajax4jsf.dev.java.net/ajax" prefix="a4j" %>

<style>
.optionList {
height:22px;
}
.vertical-menu-cell {
padding:0px 4px 0px 4px;
}
.label{
font-weight:bold;
}
</style>

<f:view>
<%@include file="menu.jsp" %>
<rich:panel style="border:3px; FONT-WEIGHT: bold; BACKGROUND-COLOR: #F1F1F1; width: 955px;">
<f:facet name="header">
<h:outputText value="Usuários"></h:outputText>
</f:facet>
<h:form>
```

```
<rich:dataGrid binding="#{usuarioMb.dataGrid}" value="#{usuarioMb.listaUsuarios}" var="usuario" id="listaUsuarios"
columns="4">
<rich:panel>
<f:facet name="header">
<h:outputText value="#{usuario.nome}"></h:outputText>
</f:facet>
<h:panelGrid columns="2">
<h:outputText value="Usuário:" styleClass="label"></h:outputText>
<h:outputText value="#{usuario.usuario}" />
<h:outputText value="Nome:" styleClass="label"></h:outputText>
<h:outputText value="#{usuario.nome}" />
<h:outputText value="Tipo:" styleClass="label"></h:outputText>
<h:outputText  value="#{usuario.tipoConv}" />
<h:outputText value="Situação:" styleClass="label"></h:outputText>
<h:outputText value="#{usuario.situacaoConv}" />
<h:outputText value="" styleClass="label"></h:outputText>
<h:commandButton style="align=left" value="Editar" action="#{usuarioMb.goEditar}"/>
</h:panelGrid>
</rich:panel>
</rich:dataGrid>
</h:form>
</rich:panel>

</f:view>
```

- ✓ Login.jsp

```
<%@ taglib uri="https://ajax4jsf.dev.java.net/ajax" prefix="a4j" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://richfaces.ajax4jsf.org/rich" prefix="rich"%>
<style>
   .box {
      BACKGROUND-COLOR: #F1F1F1;
      height:100px;
      width: 200px;
                  FONT-SIZE: small; FONT-WEIGHT: bold
      text-align:center;
   }
</style>

<f:view>
<h:graphicImage value="/imagens/inee.jpg"/>
<h:form>
<br>
         <rich:panel id="login" styleClass="box" style="border:3px;height : 179px; width : 501px;">
                  <h:outputText value="Acesse o SysFarma com o seu login" style="FONT-SIZE: small; FONT-WEIGHT:
bold;"/><br>
                  <br><h:messages style="COLOR: #ff0000;"/>
                  <h:panelGrid columns="2">
                           <h:outputLabel value="Usuário" for="usuario" style="FONT-SIZE: small;"></h:outputLabel>
                           <h:inputText id="usuario" value="#{usuarioMb.usuario.usuario}" required="true"
style="BACKGROUND-COLOR: #F1F1F1; COLOR: #000000; width : 241px;" requiredMessage="Usuário é obrigatório!"/>
                           <h:outputLabel value="Senha" for="nome" style="FONT-SIZE: small; COLOR:
#000000;"></h:outputLabel>
                           <h:inputSecret id="senha" value="#{usuarioMb.usuario.senha}" required="true"
style="BACKGROUND-COLOR: #F1F1F1; COLOR: #000000; width : 241px;" requiredMessage="Senha é obrigatório!"/>
                           <h:commandButton id="btn" action="#{usuarioMb.actionLogin}" value="Acessar"/>
                  </h:panelGrid>
   </rich:panel>
</h:form>

</f:view>
```

- ✓ Menu.jsp

```
<%@ taglib uri="https://ajax4jsf.dev.java.net/ajax" prefix="a4j" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
```

```
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://richfaces.ajax4jsf.org/rich" prefix="rich"%>

<style>
            .optionList {
            height:22px;
            }
            .vertical-menu-cell {
            padding:0px 4px 0px 4px;
            }
             .box {
      background-color: #F1F1F1;
      height:100px;
      width: 200px;
      text-align:center;
   }
</style>

<f:subview id="menu">
<h:graphicImage value="/imagens/inee.jpg"/>

            <h:form>
                        <h:panelGrid columns="1" columnClasses="optionList" cellspacing="0" cellpadding="0">
                                    <rich:dropDownMenu style="border:1px solid #{a4jSkin.panelBorderColor}" value="Usuários"
rendered="#{usuarioMb.usuarioLogin.tipo == 'A'}" direction="bottom-right" jointPoint="tr">
                                                <rich:menuItem value="Listar" action="lista_usuarios"/>
                                                <rich:menuItem value="Cadastrar" action="#{usuarioMb.goInserir}" />
                                    </rich:dropDownMenu>
                                    <rich:dropDownMenu style="border:1px solid #{a4jSkin.panelBorderColor}"
value="Medicamentos"  direction="bottom-right"  jointPoint="tr">
                                                <rich:menuItem value="Listar" action="lista_medicamentos"/>
                                                <rich:menuItem value="Cadastrar" action="#{medicamentoMb.goInserir}"/>
                                    </rich:dropDownMenu>
                                    <rich:dropDownMenu style="border:1px solid #{a4jSkin.panelBorderColor}" value="Opções"
direction="bottom-right"  jointPoint="tr">
                                                <rich:menuItem value="Principal" action="principal"/>
                                                <rich:menuItem value="Logout" action="#{usuarioMb.actionLogout}"/>
                                    </rich:dropDownMenu>
                        </h:panelGrid>
            </h:form>

</f:subview>
```

- ✓ Principal.jsp

```
<%@ taglib uri="http://richfaces.ajax4jsf.org/rich" prefix="rich"%>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="https://ajax4jsf.dev.java.net/ajax" prefix="a4j" %>

<style>
   .box {
      background-color:#F1F1F1;
      text-align:center;
      FONT-WEIGHT: bold;
   }
   .cell {
      vertical-align:top;
      FONT-WEIGHT: bold;
            }
            .optionList {
            height:22px;
            }
            .vertical-menu-cell {
            padding:0px 4px 0px 4px;
            }
</style>

<f:view>
            <%@include file="menu.jsp" %>
```

```
            <h:outputLabel value="#{usuarioMb.usuarioLogin.nome}" style="COLOR: ##F1F1F1; FONT-WEIGHT: bold;"/>
            <h:outputLabel value="Seja bem vindo!" style="COLOR: ##F1F1F1;"/>
            <br>
            <br>
            <rich:panel  id="switchbox" style="border:3px; FONT-WEIGHT: bold; BACKGROUND-COLOR: #F1F1F1; width :
753px; height : 194px;">
                        <br>
                <h:outputText value="SysFarma - Sistema para Farmácias de Postos de Saúde" style="FONT-SIZE: medium;
FONT-WEIGHT: bold;"/><br>
                <br><br>
      <h:outputText value="Universidade Federal de Santa Catarina." style="FONT-SIZE: small;"/><br>
      <h:outputText value="Centro Tecnológico - Informática e Estatística" style="FONT-SIZE: small;"/><br>
      <h:outputText value="Sistemas de Informação" style="FONT-SIZE: small;"/><br>
      <h:outputText value="Sistemas desenvolvido durante realização do Trabalho de Conclusão de Curso - 2007/2"
style="FONT-SIZE: small;"/><br>
      <h:outputText value="'Estudo do uso de Metodologias Ágeis no Desenvolvimento de uma Aplicação de Governo
Eletrônico'" style="FONT-SIZE: small;"/><br>
      <h:outputText value="Autores: Diogo Ludvig e Jonatas Reinert." style="FONT-SIZE: small;"/><br>
   </rich:panel>

</f:view>
```

11 ANEXO D – TELAS DO SISTEMA

Figura 27 – Tela de login.

Figura 28 – Tela principal logado como Administrador.

Figura 29 – Tela de cadastro/edição de usuários.

Figura 30 – Tela de Consulta de usuários.

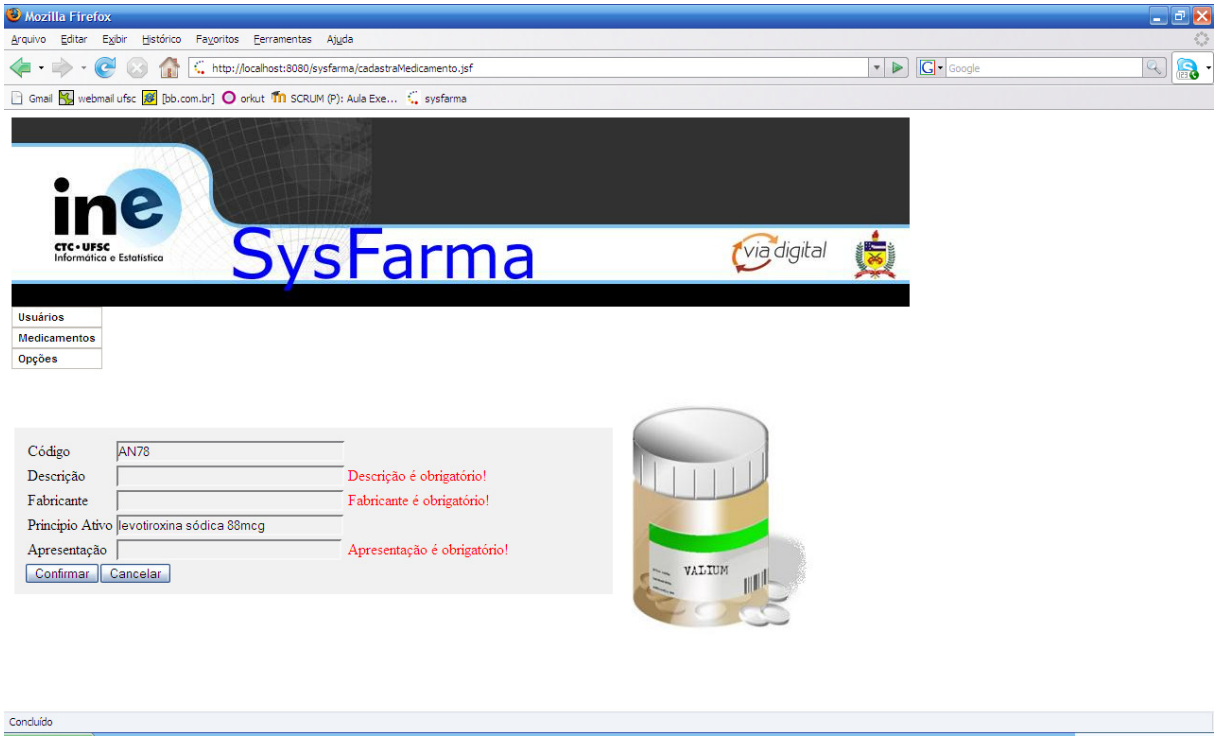


Figura 31 – Tela de cadastro/edição de medicamento.

Figura 32 – Tela de listagem de medicamentos.

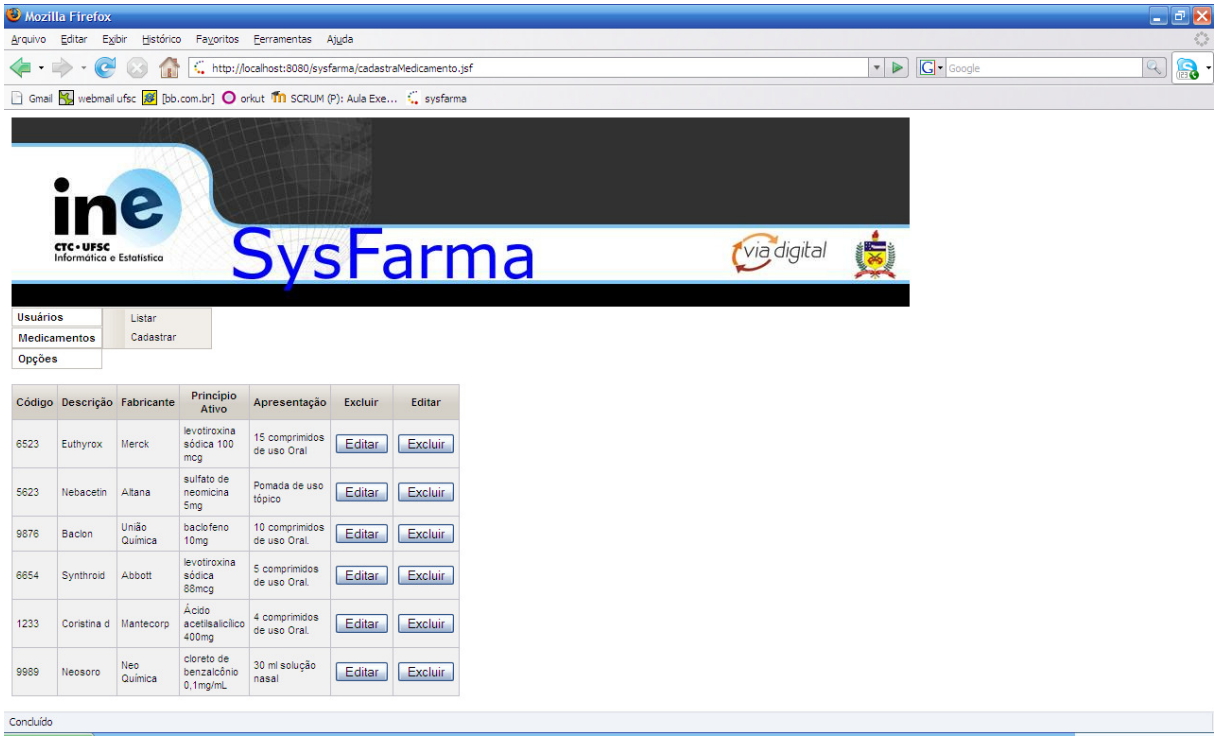


Figura 33 – Menu de Usuários.

Figura 34 – Menu de medicamentos.

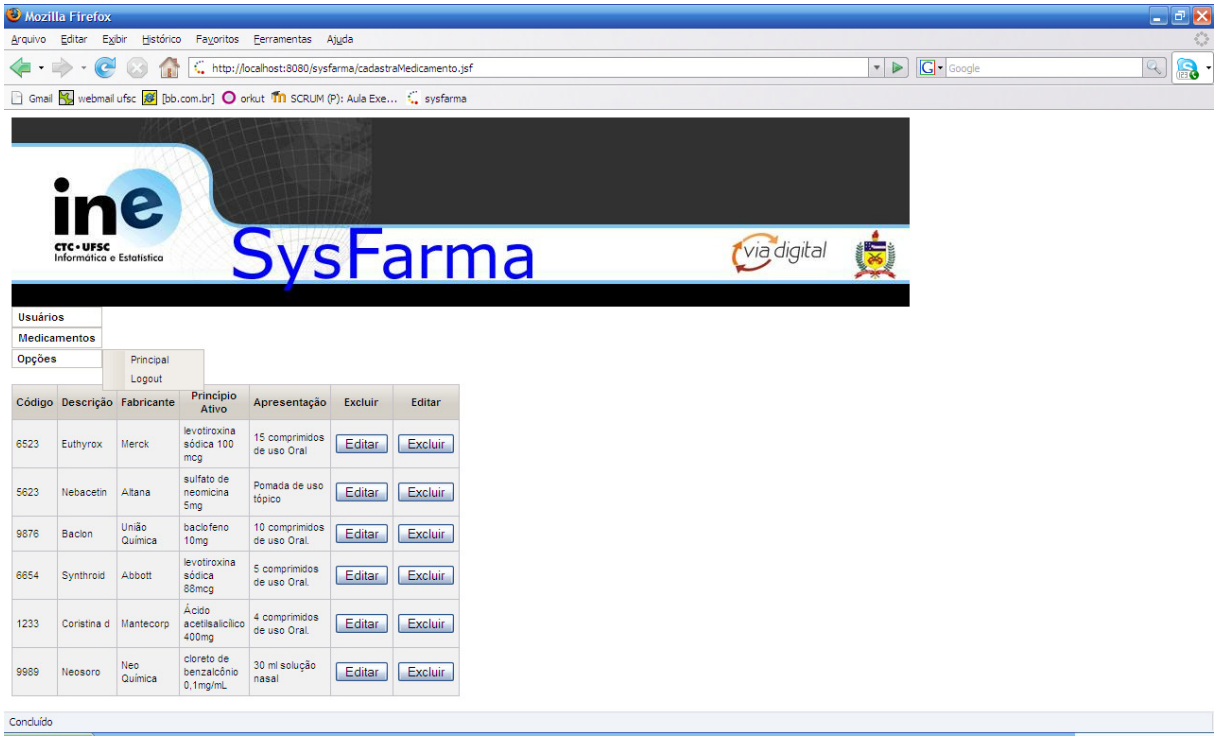


Figura 35 – Menu de Opções.

12 ANEXO E – ARTIGO

Estudo do uso de Metodologias Ágeis no Desenvolvimento de uma Aplicação de Governo Eletrônico

Diogo Ludvig[1], Jonatas Davson Reinert[1]

[1]Departamento de Informática e Estatística – Universidade Federal de Santa Catarina (UFSC) – Florianópolis – SC – Brasil

{ludvig,jonatas}@inf.ufrgs.br

Abstract. *This meta-paper describes the comparative analysis between easYProcess and Scrum agile methodologies of software development. To achieve the objective of the study, both metodologies were applied to a partial construction of a software component.*

Resumo. *Este artigo apresenta um estudo comparado entre as metodologias ágeis de desenvolvimento de software easYProcess e Scrum. Para isto foi feito a análise com cada uma das metodologias e o desenvolvimento parcial de um componente de software.*

1. Introdução

A sociedade está cada vez mais dependente da indústria de desenvolvimento de software. Porém, devido aos diversos problemas que esta área possui, como alto custo de implementação, alta complexidade e falta de manutenção, não conseguem atender a demanda do mercado.

Com isso, o interesse no estudo a aplicação de metodologias de desenvolvimento de software tem aumentado. Sommerville (2003, p.147) demonstra que a melhoria no processo de desenvolvimento de software aperfeiçoa a qualidade do produto final; e como destaca Côrtes (2001, p.20) a "preocupação com a qualidade deixou de ser um diferencial competitivo e passou a ser um pré-requisito básico para participação no mercado".

Dentre as metodologias temos duas linhas distintas, as tradicionais e as ágeis. Enquanto as tradicionais prezam por uma quantidade excessiva de documentação as ágeis prezam por ter o software funcionando com o mínimo de documentação necessária. Portanto, adotar processos mais simplificados, como as metodologias tem despertado um grande interesse entre as comunidades de desenvolvimento de software. Enquanto que as metodologias tradicionais têm seus processos baseados na produção de grande quantidade de documentação e de modelos para guiar o processo de desenvolvimento (FOWLER, 2001).

Um exemplo entre as metodologias ágeis pode-se citar YP – easYProcess (easYProcess, 2006), que é uma metodologia desenvolvida através de um projeto da Universidade Federal de Campina Grande (UFCG). Por ser um método recente, com pouca pesquisa realizada bem como sua comprovação acadêmica, surgiu a idéia da realização de um estudo mais profundo a respeito da metodologia ágil easYProcess. Por outro lado, outra metodologia que será

abordada neste trabalho é a Scrum (Scrum Alliance, 2007), onde pesquisas mostram sua grande aceitação pela comunidade de desenvolvimento de software internacional.

2. Metodologias de Desenvolvimento de Software

Metodologias de desenvolvimento de Software (ou Processo de Software) é o conjunto de atividades, ordenadas ou não, com a finalidade de produzir um software de qualidade. O resultado do processo é um produto que reflete a forma como o processo foi realizado. Embora existam vários processos de desenvolvimento de software, existem atividades comuns a todos eles [Sommerville (2004)]:

- **Especificação do software:** definição do que o sistema deve fazer; definição dos requisitos e das restrições do software;
- **Desenvolvimento do software:** projeto e implementação do software, o sistema é desenvolvido conforme sua especificação;
- **Validação do software:** o software é validado para garantir que as funcionalidades implementadas estejam de acordo com o que especificado;
- **Evolução do software:** evolução do software conforme a necessidade de cliente.

O desafio mais comum dentro do Desenvolvimento de Software é entregar um produto que atenda as reais necessidades do cliente, dentro do prazo e orçamento previsto. Porém, muitas organizações desenvolvem software sem usar nenhum Processo de Desenvolvimento. Geralmente porque os Processos Tradicionais não são adequados às realidades das organizações. O resultado da falta na utilização de Processos

2.1. Metodologias Tradicionais

Na década de 70 a atividade de "desenvolvimento de software" era executada de forma desorganizada, desestruturada e sem planejamento, gerando um produto final de má qualidade, que não correspondia com as reais necessidades do cliente (PRESSMAN, 2001). A partir deste cenário surgiu a necessidade de tornar o desenvolvimento de software como um processo estruturado, planejado e padronizado (NETO, 2004).

As metodologias consideradas tradicionais também são conhecidas como "pesadas" ou orientadas a documentação. Têm como característica marcante dividir o processo de desenvolvimento em etapas e/ou fases bem definidas. Segundo SOARES (2004), essas metodologias surgiram em um contexto de desenvolvimento de software muito diferente do atual, baseado apenas em um mainframe e terminais burros. Na época, o custo de fazer alterações era muito alto, uma vez que o acesso aos computadores era limitado e não existiam modernas ferramentas de apoio ao desenvolvimento do software, como depuradores e analisadores de código. Por isso, o software era todo planejado e documentado antes de ser implementado.

Muitas metodologias pesadas são desenvolvidas em cima do Modelo em Cascata (SOMMERVILLE, 2004). É um modelo em que as fases definidas são seguidas de maneira linear (LARMAN, 2002). Cada fase tem associado ao seu término uma documentação padrão que deve ser aprovada para que se inicie a etapa imediatamente posterior. Cada fase concluída

gera um marco, que geralmente é algum documento, protótipo do software ou mesmo uma versão do sistema (NETO, 2004). De uma forma geral fazem deste modelo as etapas de análise de requisitos, projeto do software, geração de código e testes.

2.2. Metodologias Ágeis

A partir da década de 90 começaram a surgir novos métodos sugerindo uma abordagem de desenvolvimento ágil, em que os processos adotados tentam se adaptar às mudanças, apoiando a equipe de desenvolvimento em seu trabalho (FAGUNDES, 2005). As metodologias ágeis surgiram como uma reação às metodologias pesadas (FOWLER, 2005) e tiveram como principal motivação criar alternativas para o Modelo em Cascata (HILMAN, 2004).

O termo "Metodologia Ágil" tornou-se popular em fevereiro de 2001, quando um grupo de 17 especialistas (referências mundiais em desenvolvimento de software) criou a Aliança Ágil e estabeleceram o Manifesto Ágil para o desenvolvimento de software (HIGHSMITH, 2002). Conforme Highsmith (2002), os valores do Manifesto Ágil são:

- Indivíduos e interações valem mais que processos e ferramentas;
- Um software funcionando vale mais que documentação extensa;
- A colaboração do cliente vale mais que a negociação de contrato;
- Responder a mudanças vale mais que seguir um plano.

De acordo com Fagundes (2005), para auxiliar as pessoas a compreender o enfoque do desenvolvimento ágil, os membros da Aliança Ágil refinaram as filosofias contidas em seu manifesto em uma coleção de doze princípios, aos quais os métodos ágeis de desenvolvimento de software devem se adequar. Estes princípios são (COCKBURN, 2000):

1. A prioridade é satisfazer ao cliente através de entregas de software de valor contínuas e freqüentes;
2. Entregar softwares em funcionamento com freqüência de algumas semanas ou meses, sempre na menor escala de tempo;
3. Ter o software funcionando é a melhor medida de progresso;
4. Receber bem as mudanças de requisitos, mesmo em uma fase avançada, dando aos clientes vantagens competitivas;
5. As equipes de negócio e de desenvolvimento devem trabalhar juntas diariamente durante todo o projeto;
6. Manter uma equipe motivada fornecendo ambiente, apoio e confiança necessário para a realização do trabalho;
7. A maneira mais eficiente da informação circular dentro da equipe é através de uma conversa face a face;
8. As melhores arquiteturas, requisitos e projetos provêm de equipes organizadas;

9. Atenção contínua a excelência técnica e um bom projeto aumentam a agilidade;
10. Processos ágeis promovem o desenvolvimento sustentável. Todos envolvidos devem ser capazes de manter um ritmo de desenvolvimento constante;
11. Simplicidade é essencial;
12. Em intervalos regulares, a equipe deve refletir sobre como se tornarem mais eficazes e então se ajustar e adaptar seu comportamento.

3. easYProcess

Assim como na construção de uma casa ou edifício, um software para estar pronto de acordo com as necessidades do cliente, envolve vários aspectos teóricos e práticos, a utilização de um processo de desenvolvimento bem como embasamentos da engenharia de software.

Conforme Hazzan e Dubinsky (2003) "processos de Engenharia de Software são difíceis de serem implantados, até mesmo na academia". E, segundo comprovação feita pela Universidade Federal de Campina Grande (UFCG) pela disciplina de Engenharia de Software do curso de Ciência da Computação, perceberam que a utilização de processos como Rational Unified Process (RUP) e eXtreme Programming (XP) não renderam projetos a níveis aceitáveis, ou seja, projetos mal sucedidos e desenvolvedores mal capacitados.

Diante destas dificuldades, o Departamento de Sistemas e Computação da UFCG apresenta desde Maio de 2003, o easYProcess, um processo de software mais simplificado que se apóia em práticas do XP, RUP e Agile Modeling (AM). Sendo assim, o YP tem como objetivo auxiliar tanto a gerência do desenvolvimento de aplicações em disciplinas de engenharia de software quanto à aprendizagem dos conceitos desta disciplina na graduação, podendo também ser utilizado em projetos de pequeno e médio porte (GARCIA; LIMA; FERREIRA; JÚNIOR; ROCHA; MENDES; PONTES; ROCHA; DANTAS, 2007).

4. Scrum

Scrum é uma abordagem simples aplicada ao gerenciamento de tarefas complexas (CRUZ, 2006). Trata-se mais especificamente de uma metodologia ágil de desenvolvimento de software, que tem como principais características ser um processo empírico, iterativo e incremental.

Em suas premissas temos que o desenvolvimento de software é muito complexo e imprevisível para ser planejado totalmente no início do projeto. Ao invés disso, o processo deve ser controlado empiricamente para garantir a visibilidade, inspeção e adaptação (MARÇAL; FREITAS; SOARES; MACIEL; BELCHIOR, 2007).

Foi desenvolvido por Ken Schwaber e Mike Beedle na década de 90, possui papéis e fases bem definidas, que veremos com maiores detalhes a seguir.

- **Conceitos e Artefatos**. A seguir veremos alguns dos mais importantes conceitos e artefatos utilizados dentro do processo de desenvolvimento com Scrum:
    - Product Backlog: Temos no product backlog uma lista priorizada dos requisitos, tanto funcional como não-funcional. Em cada item desta lista temos

um valor de negócio associado, onde podemos medir o retorno do projeto e a priorização dos itens.

- Sprint: Cada iteração do processo de desenvolvimento é denominada de Sprint. A duração de cada Sprint é recomendando que fique entre dois e quatro semanas.
- Sprint Backlog: É uma lista de tarefas, onde temos o trabalho da equipe em cada Sprint do processo. A lista nasce durante o planejamento do Sprint. As tarefas do Sprint Backlog são o que a equipe definiu como sendo necessário para a fluência da realização dos itens do Product Backlog nas funcionalidades do sistema. Cada tarefa é identificada pelo seu responsável e a sua quantidade estimada de trabalho restante (SCRUM FOR TEAM SYSTEM, 2007).

- **Papéis e responsabilidades**. A seguir veremos três dos principais papéis implementados pelo Scrum:
  - Product Owner: É o "dono do produto", identificando o interesse de todos no projeto. Além de priorizar os requisitos do projeto é o responsável pelo seu ROI (return of investment). Trocando em miúdos: é o dono do produto que sabe o que vai gerar retorno ao projeto (CRUZ, 2006).
  - ScrumMaster: É a pessoa responsável por fazer o Scrum funcionar. Deve ensinar a metodologia a todos os envolvidos no processo, assim como assegurar que todos sigam suas regras e práticas. Trabalha juntamente com o Product Owner na organização dos requisitos.
  - Team Members: Desenvolvem as funcionalidades do produto, são responsáveis coletivamente pelo sucesso da iteração e conseqüentemente pelo projeto como um todo (MARÇAL; FREITAS; SOARES; MACIEL; BELCHIOR, 2007).
- **Fluxo de Desenvolvimento**. No Scrum, um projeto se inicia com a visão do produto que será desenvolvido. Em seu desenvolvimento, são realizados vários Sprints. A cada Sprint são estabelecidos novos itens a serem desenvolvidos, e durante sua realização temos diariamente reuniões de aproximadamente 15 minutos com os membros da equipe. No final de cada Sprint, é feita uma retrospectiva para avaliar como esta o andamento do processo, posteriormente são levantados os próximos itens a serem desenvolvidos para a realização de um novo Sprint.

5. Comparação dos Métodos Ágeis YP e Scrum

Agora, será realizada uma análise comparativa entre as metodologias estudas neste trabalho. São elas: *easYProcess* e *Scrum*. A realização da análise tem como base a satisfação dos princípios ágeis classificando cada prática em NS – Não Satisfeito, PS – Parcialmente Satisfeito e S – Satisfeito.

N.	Princípio Ágil	easYProcess	Scrum
1	A prioridade é satisfazer ao cliente através de entregas de software de valor contínuo e freqüentes	S	S
2	Receber bem as mudanças de requisito, mesmo em uma fase avançada, dando aos clientes vantagens competitivas.	PS	S
3	Entregar software em funcionamento com freqüência de algumas semanas ou meses, sempre na menor escala de tempo.	S	S
4	Equipes de negócio e de desenvolvimento devem trabalhar juntas diariamente durante todo o projeto.	PS	S
5	Manter uma equipe motivada fornecendo ambiente, apoio e confiança necessária para a realização do trabalho.	S	S
6	A maneira mais eficiente da informação circular dentro da equipe é a através de uma conversa face a face.	S	S
7	Ter o software funcionando é a melhor medida de progresso.	S	S
8	Processos ágeis promovem o desenvolvimento sustentável. Todos envolvidos devem ser capazes de manter um ritmo de desenvolvimento constante.	S	S
9	Atenção contínua a excelência técnica e um bom projeto aumentam a agilidade.	S	PS
10	Simplicidade é essencial.	S	S
11	As melhores arquiteturas, requisitos e projetos provêm de equipes organizadas.	S	S
12	Em intervalos regulares, as equipes devem refletir sobre como se tornarem mais eficazes e então se ajustar e adaptar seu comportamento.	PS	S

Fonte: (COCKBURN, 2000)

Tabela 37 – Comparação das metodologias de acordo com os princípios ágeis

Como podemos perceber na tabela 1, apenas 1 princípio foi parcialmente satisfatório pelo Scrum, enquanto os demais foram satisfatórios. Enquanto no YP obtemos 3 princípios como parcialmente satisfatórios.

A metodologia *YP* se destaca com suas práticas sistemáticas dando um grande valor a engenharia de processos, como podemos perceber na análise comparativa do princípio 9. Já a metodologia Scrum da um maior destaque na gerencia do projeto onde reuniões diárias acontecem durante o *Sprint* favorecendo o controle do projeto pelo *ScrumMaster*.

Na prática as duas modelagens atendem suas necessidades. O *YP* por ser uma metodologia voltada para projetos acadêmicos é mais detalhado em documentos e artefatos. Já o *Scrum* que é mais voltado para a gerência do processo de desenvolvimento se destaca mais em aspectos de gerência e organização da equipe durante o processo.

Notamos que em muitos aspectos as duas metodologias se equivalem. Como por exemplo: A definição dos requisitos em *YP* se dá na construção do documento de visão onde serão listadas e geradas as *Users Stories*, No *Scrum*, a definição dos requisitos acontece com a criação do *Product Backlog* a partir da fase de preparação.

Outro aspecto que as duas metodologias se equivalem é a preparação para o processo de desenvolvimento onde no *YP* as *User Stories* são divididas em planos de releases e iteração. Já no Scrum, os requisitos listados no Product Backlog são divididos em *Sprint Backlog* onde serão dividido em *Sprints* para o desenvolvimento. Após cada release e *Sprint*, uma nova versão do produto é lançada para apresentação ao cliente.
Como as duas metodologias focam no contato constante com o cliente e na comunicação pessoal periódica entre a equipe, em projetos de software pode-se utilizar da tecnologia de comunicação atual e realizar estas reuniões através de sistemas de mensagens instantâneas, onde se pode até realizar teleconferência. Como exemplos destes sistemas podemos citar: *Microsoft MSN*, *Skype* e *Google Talk*.

6. Conclusão

Foi verificada ao longo da pesquisa que a metodologia YP é uma ótima alternativa, porém sendo específica em situações como, por exemplo, em projetos acadêmicos e tem como aspecto positivo a possibilidade de execução de todas as suas tarefas entre a equipe de desenvolvimento e o cliente.

Já a metodologia Scrum descreve melhor a organização de como o projeto deve ser executado, não definindo muitos documentos de especificação técnica do componente. Por exemplo, o Scrum não define que temos que criar um modelo lógico de dados, porém a metodologia não impede que este artefato seja criado para facilitar a implementação do sistema sendo que este artefato pode ser definido como um item do Product Backlog.

Um dos pontos negativos durante a modelagem YP é a definição de artefatos que julgamos desnecessários ou improdutivos como o modelo de tarefas. Já na modelagem Scrum, percebemos que é uma metodologia voltada para pessoas que já tenham um grau de conhecimento em projetos de software e que queiram otimizá-lo.

References

COCKBURN, Alistair. **Agile Software Development.** Adisson-Wesley, 2001.

CÔRTES, Mario Lúcio; CHIOSSI, Thelma C. dos Santos. **Modelos de Qualidade de Software.** Campinas: Unicamp, Instituto de Computação, 2001.

EASYPROCESS**.** Disponível em: <http://www.dsc.ufcg.edu.br/~yp/>. Acessado em 11 dez 2006.

FAGUNDES, Priscila Bastos. **Framework para Comparação e Análise de Métodos Ágeis.** Dissertação de Mestrado. Universidade Federal de Santa Catarina. Florianópolis: 2005.

FOWLER, Martin. **The New Methodology.** 2005. Disponível em: <http://www.martinfowler.com/articles/newMethodology.html>. Acessado em 15 abril 2007.

GARCIA, Francile Procópio; LIMA, Aliandro Higino Guedes; FERREIRA, Danilo de Sousa; JÚNIOR, Fábio Luiz Leite; ROCHA, Giselle Regina Chaves da; MENDES, Gustavo Wagner

Diniz; PONTES, Renata França de; ROCHA, Verlaynne Kelley da Hora; DANTAS, Vinicius Farias. ***easYProcess* – Um Processo de Desenvolvimento de Software.** Universidade Federal de Campina Grande. Campina Grande: 2007.

HAZZAN, O.; DUBINSKY, Y. **eXtreme Programming as a Framework for Student-Project Coaching in Computer Science Capstone Courses,** 2003.

HIGHSMITH, Jim. **Agile Software Development Ecosystems.** Adisson-Wesley, 2002.

HILMAN. **Metodologias Ágeis.** 2004. Disponível em: <http://www.redes.unb.br/material/ESOO/Metodologias%20%c1geis.pdf>. Acessado em 20 abril 2007.

LARMAN, Craig. **Applying UML And Patterns.** 2nd Edition, 2002.

MARÇAL, Ana Sofia Cysneiros; FREITAS, Bruno Celso Cunha; SOARES, Felipe Santana Furtado; MACIEL, Teresa Maria Medeiros; BELCHIOR, Arnaldo Dias. **Estendendo o SCRUM segundo as Áreas de Processo de Gerenciamento de Projetos do CMMI.** CLEI2007: XXXIII Conferencia Latinoamericana de Informática, San Jose, Costa Rica. Disponível em: <http://www.cesar.org.br/files/file/SCRUMxCMMI-CLEI-2007.pdf>. Acesso27 de agosto de 2007.

NETO, Oscar Nogueira de Souza. **Análise Comparativa das Metodologias de Desenvolvimento de Softwares Tradicionais e Ágeis.** Trabalho de Conclusão de Curso. Universidade da Amazônia. Belém: 2004.

PRESSMAN, Roger. **Software Engineering – A Pratitictioner's Approach.** McGraw-Hill, 5th Edition, 2001.

SCRUM ALLIANCE. Disponível em: <http://www.scrumalliance.org> Acesso em Setembro de 2007.

SCRUM FOR TEAM SYSTEM. Disponível em: <http://scrumforteamsystem.com/en/default.aspx>. Acesso em 28 de agosto de 2007.

SOARES, Michel dos Santos. **Comparação entre Metodologias Ágeis e Tradicionais para o Desenvolvimento de Software.** 2004. Disponível em: <http://www.dcc.ufla.br/infocomp/artigos/v3.2/art02.pdf> . Acessado em 11 janeiro 2007.
SOMMERVILLE, Ian**. Engenharia de Software**. São Paulo: Addison-Wesley, 2003.

SOMMERVILLE, Ian. **Software Engineering.** 7th Edition, 2004.