UNIVERSIDADE ESTADUAL DA PARAÍBA
CAMPUS GOVERNADOR ANTONIO MARIZ
CENTRO DE CIÊNCIAS EXATAS E SOCIAIS APLICADAS - CCEA
LICENCIATURA PLENA EM COMPUTAÇÃO

ELDER GONÇALVES PEREIRA

Implementando um Componente de Software com o Auxílio da Metodologia Ágil easYProcess.

PATOS – PB
2011

ELDER GONÇALVES PEREIRA

Implementando um Componente de Software com o Auxílio da Metodologia Ágil easYProcess.

Monografia apresentada ao Curso de Graduação **Licenciatura Plena em Computação** da Universidade Estadual da Paraíba, em cumprimento à exigência para obtenção do grau de Licenciado em Computação.

Orientador: Prof. Esp. Vitor Abílio Sobral Dias Afonso

PATOS– PB

2011

P436i PEREIRA, Elder Gonçalves

Implementando um componente de software com auxílio da metodologia ágil easYProcess
/Elder Gonçalves Pereira_-Patos: UEPB, 2011.
75 f.

Monografia (TRABALHO de Conclusão de Curso - (TCC) - Universidade Estadual da Paraíba.
Orientador: Prof. Esp. Vitor Abílio Sobral Dias Afonso

1. Engenharia de software 2. Desenvolvimento de software I. Titulo II. Afonso, Vitor Abílio Sobral Dias

CDD 005.1

ELDER GONÇALVES PEREIRA

Implementando um Componente de Software com o Auxílio da Metodologia Ágil easYProcess.

Monografia apresentada ao Curso de Graduação **Licenciatura Plena em Computação** da Universidade Estadual da Paraíba, em cumprimento à exigência para obtenção do grau de Licenciado em Computação.

Aprovado em: 16/11/2011

Prof. Esp. Vitor Abílio Sobral Dias Afonso / UEPB
Orientador

Prof. Msc. Wellington Candeia de Araújo / UEPB
Examinador

Prof. Msc. Vilmar Vaz da Silva / UEPB
Examinador

PATOS – PB

2011

AGRADECIMENTOS

Primeiramente a Deus, pois sem ele nada é possível, e em segundo a minha mãe, Maria de Fátima Gonçalves Pereira e todos os meus irmãos, pela dedicação, companheirismo e amizade. Ao professor Vitor Abílio Sobral Dias Afonso pelas leituras sugeridas e correções ao longo dessa orientação, a todos os professores que fizeram isso se tornar realidade. E a uma pessoa que me ajudou muito no processo gramatical desse trabalho, Luana Araújo.

RESUMO

O presente Trabalho de Conclusão de Curso objetiva apresentar a metodologia ágil de desenvolvimento easYProcess (YP) aplicada à implementação do Software para Controle de Vendas (SysVenda), mostrando e especificando na prática a quantidade de artefatos que esta metodologia de desenvolvimento produz durante a elaboração do sistema de software. Partindo do fato de que easYProcess (YP) encontra-se em sua versão original, datada do ano de 2007, sem atualizações, e competindo com outras metodologias ágeis presentes atualmente no mercado de trabalho. Para essa realização todos os artefatos necessários para a esquematização do problema são definidos e especificados segundo a metodologia em questão.

Palavras chaves: Engenharia de Software, desenvolvimento ágil, easYProcess.

A B S T R A C T

This Course's conclusion project is intended to present the agile methodology of development easYProcess (YP) applied to software implemtation: Sales Management (SysVendas) , show up and specifying on practice the amount of artifacts that this development's methodology produces for software's system preparation. Keeping in mind that the easYProcess (YP) is in your original version, from year 2007, without update and competing with others current agile methodology existent on current market. To realize this project all artifacts necessary to builindg the problem are defined and specified follow the methodology used in the study

keyword: Software engineering, agile development, easYProcess.

LISTAS DE FIGURAS

Figura 01: Modelo em Cascata ---------- 20

Figura 02: Modelo em Espiral ---------- 21

Figura 03: Síntese do Fluxo do Processo YP ---------- 29

Figura 04: Cadastro de Produto ---------- 47

Figura 05: Venda de Produto ---------- 48

Figura 06: Lista de Produto ---------- 48

Figura 07: Confirmação de Venda ---------- 49

Figura 08: Modelo Arquitetural ---------- 49

Figura 09: Modelo Relacional ---------- 50

Figura 10: Interface de Venda ---------- 74

LISTAS DE QUADROS

Quadro 01: Definição de papéis ---------- 41

Quadro 02: Objetivo de usabilidade ---------- 43

Quadro 03: Uses Stories ---------- 45

Quadro 04: Matriz de Competência ---------- 52

Quadro 05: Plano de Release 01 ---------- 53

Quadro 06: Plano de Release 02 ---------- 53

Quadro 07: Plano de Release 03 ---------- 53

Quadro 08: Plano de Iteração 02 ---------- 54

Quadro 09: Big Chat ---------- 59

Quadro 10: Plano de Iteração 02 ---------- 60

Quadro 11: Risco ---------- 61

SÚMARIO

1. INTRODUÇÃO ---------- 12

1.1. PROBLEMÁTICA ---------- ---------- 13

1.2 JUSTIFICATIVA ---------- ---------- 14

1.3 DELIMITAÇÃO ---------- ---------- 15

1.4 OBJETIVOS ---------- ---------- 16

1.4.1 Geral ---------- ---------- 16

1.4.2 Específicos ---------- ---------- 16

1.5 ESTRUTURA DO TRABALHO ---------- 16

2 ENGENHARIA DE SOFTWARE ---------- ---------- 17

2.1 PROCESSOS DE DESENVOLVIMENTO DE SOFTWARE ---- --------- 18

2.2 METODOLOGIAS DE DESENVOLVIMENTO TRADICIONAL ------- 20

2.3 METODOLOGIA DE DESENVOLVIMENTO ÁGIL ----------22

2.4 MODELOS DE DESENVOLVIMENTO APOIADORES DO YP -------- 23

2.4.1 Extreme Programming - XP (programação extrema) ---------- 23

2.4.1.1 Definição 23

2.4.1.2 Princípio 23

2.4.2 Rational Unified Process - RUP (processo unificado racional) 25

2.4.2.1 Definição 25

2.4.2.2 Principio 25

2.4.3 Agile Modeling (AM) 26

2.4.3.1 Definição 26

2.4.3.2 Princípios 27

2.4.3.3 Práticas 27

3 EASY PROCESS 29

3.1 DEFINIÇÃO DE PAPÉIS 30
3.2 CONVERSA COM O CLIENTE 32
3.2.1 Descrição 32
3.2.2 Requisitos 32
3.2.3 Perfil do Usuário 33
3.2.4 Objetivos de Usabilidade 33
3.3 INICIALIZAÇÃO 34
3.3.1 Modelo de Tarefa 34
3.3.2 User Stories e Teste de Aceitação 34
3.3.3 Protótipo da Interface 34
3.3.4 Projeto Arquitetural 35
3.3.5 Modelo Lógico de Dados 35
3.4 PLANEJAMENTO 36
3.5 IMPLEMENTAÇÃO 37
3.5.1 Integração continua 37
3.5.2 Boas Práticas de Codificação 37
3.5.3 Propriedade Coletiva de Código 38
3.5.4 Testes 38
3.5.5 Pequenos Releases 39
3.6 REUNIÃO DE ACOMPANHAMENTO 40

4 SYS VENDA E YP 41
4.1 DEFINIÇÃO DE PAPÉIS 41
4.2 CONVERSA COM O CLIENTE 42
4.3 INICIALIZAÇÃO 45
4.4 PLANEJAMENTO 52
4.5 IMPLEMENTAÇÃO 56
4.6 REUNIÃO DE ACOMPANHAMENTO 59

5 CONSIDERAÇÕES FINAIS 63

5.1 TRABALHOS FUTUROS 64

REFERÊNCIAS 65

Apêndice A: Estrutura Base do Sistema 68

Apêndice B: Modulo de Venda 70

Apêndice C: Teste de Usabilidade 74

1 INTRODUÇÃO

Neste momento, o mundo encontra-se em uma sociedade informatizada que passa por sucessivas modificações, e em conseqüência disso, o comércio internacional e nacional precisa de soluções computacionais disponibilizadas ao negócio do cliente o mais rápido possível, que possa auxiliar no ambiente de negócio de forma mais fácil e produtiva.

De acordo com Sommerville (2007), os softwares têm que ser desenvolvidos rapidamente para que possam atender as necessidades de negócio do mercado, a entrega rápida do software às vezes é considerada o requisito mais crítico, isso devido as constantes variações no ambiente de negócio, os requisitos mudam rapidamente por que é praticamente quase que impossível prever como o sistema se comportará, entretanto somente ao término do sistema que os requisitos tornam-se claros.

Visando uma maneira de produzir software mais rápido, de qualidade e que atenda a exigência do cliente, a Engenharia de Software propõe e estuda novas formas de desenvolvimento baseadas unicamente em processos ágeis, conhecidos como Modelos de Desenvolvimento Ágeis. Este trabalho tem como fundamento apresentar easYProcess (YP) como uma alternativa para o desenvolvimento ágil de aplicações. Para efetivação desse trabalho, torna-se necessário a modelagem e implementação de um componente de software baseado no modelo de desenvolvimento em questão, onde os passos do processo de desenvolvimento são relatados e analisados.

Normalmente uma metodologia de desenvolvimento requer uma quantidade mínima de pessoas para o processo de construção do software, ao se tratar de um TCC (trabalho de conclusão de curso) os outros membros da equipe podem ser parcialmente omitidos. Sabendo que o YP é uma metodologia de desenvolvimento simples onde a mesma pessoa pode desempenhar mais de um papel no processo de desenvolvimento.

Segundo GARCIA (2007), "Um papel não corresponde necessariamente a uma pessoa da equipe, ou seja, uma mesma pessoa pode desempenhar vários papéis simultaneamente."

1.1 PROBLEMÁTICA

Em algumas equipes de desenvolvimento de software um dos problemas encontrados no processo está relacionado à conclusão do projeto em tempo programado, isso devido à falta de uma forma eficiente de gerenciamento que, fazem as entidades de uma equipe trabalhar descoordenadamente.

Pensando neste tipo de problemática, surgiu uma nova metodologia de desenvolvimento baseada em agilidade. Como afirma Mansur (2007), o mercado de TI responde que a metodologia ágil permite reduzir o tempo de entrega de software para introduzi-lo no ambiente de trabalho, como também um simples gerenciamento para o projeto, atribuindo aos membros da equipe maior facilidade para o processo de desenvolvimento de software.

Afirma Caetano (2009), que as metodologias de desenvolvimento ágeis oferecem aos membros da equipe de software maior flexibilidade para o desenvolvimento, como também uma maior participação e aproximação do usuário final. Com esse tipo de metodologia, a efetivação do projeto é realizada em etapas, resultando em tempos de entrega de software menores em torno de três a seis semanas.

Produtos de software com qualidade reprovada, orçamentos estourados, gerenciamento descoordenado, atraso na entrega do produto final do software, são alguns dos fatores que levam vários pesquisadores a definirem novas metodologias de desenvolvimento de software que, auxilie um processo de desenvolvimento simplicista e eficiente. Onde a curva de aprendizado do processo seja baixa, para os membros da equipe não desperdiçarem muito tempo ao analisarem a nova abordagem de desenvolvimento.

1.2 JUSTIFICATIVA

Apesar do curso de Licenciatura em Computação da Universidade Estadual da Paraíba (UEPB) oferecer uma disciplina na área de engenharia de software, o conhecimento apresentado é considerado insuficiente para um mercado de trabalho tão exigente quando o assunto é desenvolvimento de sistemas.

Sob essa ótica, torna-se essencial a aquisição de mais conhecimento na área de Engenharia de Software por parte do formando, e em especial, abordando uma temática específica que são os Processos de Desenvolvimento de Software, mais intrinsecamente a Metodologia Ágil, fazendo com que o discente tenha mais atribuições na área de interesse, e torne-se mais qualificado para atuação e absorção do mercado tecnológico, comprometendo-se com o setor desenvolvimentista.

A priori, o easYProcess (YP) foi escolhido para a modelagem e implementação do desenvolvimento de um sistema, por ser considerado uma Metodologia de Desenvolvimento de Software Ágil simples e eficiente, e apresentar uma curva de aprendizagem baixa. Surgindo num cenário acadêmico da Universidade Federal de Campina Grande (UFCG), o YP vem auxiliando os alunos nas disciplinas que envolvem desenvolvimentos de sistemas de softwares desde o ano de 2003, com mais de 90% dos projetos chegando à fase final.

O YP é uma metodologia alternativa para ser adotada não só no meio acadêmico, mas como também no ambiente comercial, seja na categoria de Free-lance ou empresarial, os usuários da mesma não precisam ter um conhecimento específico na área de Engenharia de software, uma vez que apresenta uma abordagem dos processos de forma bem intuitiva.

1.3 DELIMITAÇÃO

A presente pesquisa aborda o processo de implementação de um componente de software com o auxílio de uma metodologia ágil de desenvolvimento, a saber, o easYProcess.

Apesar de o componente desenvolvido ser produzido em virtude de um trabalho acadêmico, o mesmo poderá ajudar no auxílio de venda de produtos do comércio em geral, realizando todo o controle de estoque, o cenário cadastral, dentre outras funcionalidades.

1.4 OBJETIVOS

1.4.1 Geral

Realizar, na prática, um estudo referente a uma metodologia de desenvolvimento ágil, no caso, o easYProcess (YP); para tal será modelado e implementado um componente de software com auxílio desta.

1.4.2 Específicos

I. Apresentar as características das metodologias de desenvolvimento ágeis e tradicionais;
II. Analisar as principais características das metodologias de desenvolvimento apoiadoras do easYProcess;
III. Especificar detalhadamente a metodologia de desenvolvimento ágil easYProcess;
IV. Modelar e implementar um componente de software com auxílio da metodologia de desenvolvimento easYProcess;
V. Avaliar a aderência do YP no desenvolvimento de um Sistema.

1.5 ESTRUTURA DO TRABALHO

O restante do trabalho está dividido em quatro capítulos:

- A seção 2 (dois) apresenta a Revisão de Literatura acerca de Engenharia de Software;
- A seção 3 (três) aponta a Metodologia Ágil de Desenvolvimento com ênfase em easYProcess;
- A seção 4 (quatro) aborda o Desenvolvimento de Software com o processo YP;
- A seção 5 (cinco) apresenta as Considerações Finais sobre Estudo em questão.

2 ENGENHARIA DE SOFTWARE

Engenharia é a aplicação de conhecimentos científicos, empíricos e certas habilitações especificas que tem como objetivo converter recursos naturais para atender as necessidades humanas, já o software é o conjunto de componentes informacionais que não fazem parte do equipamento físico, podendo incluir dados e programas à ele associado (AURÉLIO, 2004).

Para Pressman (2001), a Engenharia de Software é um rebento da Engenharia de Sistemas e Hardware, abrangendo três fundamentações, a saber: métodos (técnicas usadas para simplificar o processo de desenvolvimento), ferramentas (softwares CASE[1] que auxiliam no processo de desenvolvimento) e procedimentos (uma ligação entre os métodos e as ferramentas usadas no processo de desenvolvimento). Dando ao gerente o controle para o processo de desenvolvimento de software de alta qualidade.

Segundo Somerville (2007), a Engenharia de software é uma disciplina que aborda todos os aspectos na construção do software, desde a fase de especificação de requisitos (processo no desenvolvimento que tem como função coletar dados a respeito do sistema de software a ser construído), até a fase de manutenção (onde o sistema de software produzido entra em operação e periodicamente é submetido a novas especificações e alterações).

Sendo uma área de conhecimento da computação voltada para construção de softwares, aplicando tecnologias e práticas que englobam Linguagens de Programação, Banco de Dados, Ferramentas CASE, Plataformas, Bibliotecas de Código, Padrões de Projeto e a questão da Qualidade de Software (MOLINARI, 2007).

[1] Manoel Silva e Thaissa Rocha (1998): "Computer-Aided Software Engineering é uma classificação que abrange todas ferramentas baseadas em computadores que auxiliam atividades de engenharia de software, desde análise de requisitos e modelagem até programação e testes".

2.1 PROCESSOS DE DESENVOLVIMENTO DE SOFTWARE

Processos de Desenvolvimento de Software são um conjunto de atividades requeridas para a produção de um software com alta qualidade, onde todo controle do processo é administrado pelo gerente (PRESSMAN, 2001). Embora existam vários tipos de processos de desenvolvimentos de software, as atividades são consideradas genéricas em relação aos modelos até então existentes, relacionadas unicamente as seguintes atividades (SOMMERVILLE, 2007):

- **Requisitos:** Especifica todas as funções e características do sistema que será produzido, é dividido em duas categorias: requisitos funcionais (são as funções que sistema deverá suportar), e não funcionais (são características que o sistema deverá apresentar) do sistema de software em questão;
- **Projeto:** É responsável de especificar o problema do cliente em diagramas, deixando mais específicos para os membros da equipe, nesta fase são propostos vários modelos arquiteturais em formas de diagramas, e apenas um sendo selecionado e direcionado para o processo de desenvolvimento;
- **Desenvolvimento:** Neste momento começa a codificação propriamente dita do sistema de software, nesta fase é aconselhável que os desenvolvedores tenham em mente alguns padrões de codificação de terceiros que apresente como objetivo auxiliar os mesmos em uma codificação coerente, simples e de fácil entendimento entre os membros da equipe;
- **Validação e Verificação:** Nesta etapa o sistema de software então já produzido é submetido à prova, diversos aspectos relacionados ao funcionamento do produto de software devem ser verificados e analisados, com um único objetivo, encontrar falhas que comprometam o bom desempenho do sistema de software, e então direcioná-las aos desenvolvedores para correção;
- **Gerenciamento:** A equipe de desenvolvimento deve ser gerenciada por um ou grupo de pessoas que tenham um bom conhecimento do sistema de software a ser produzido, e apresente qualificação para gerenciar, tais como, estimativa de custo, gerenciamento de qualidade de software, aprimoramento do processo de desenvolvimento e que

apresente grande capacidade de estabelecer uma comunicação eficiente entre os membros da equipe.

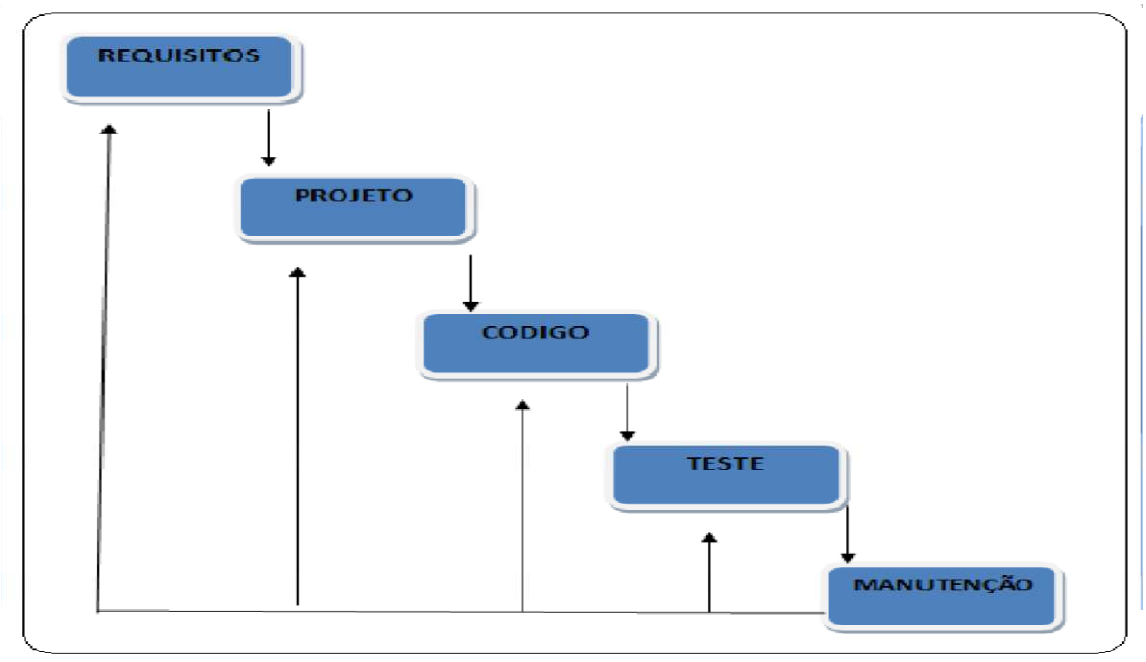
O desenvolvimento de software não é uma tarefa simplificada, pois o mesmo problema pode apresentar inúmeras soluções. Além da efetivação do desenvolvimento do software depender unicamente da competência de um bom relacionamento entre os membros da equipe (AVILA, 2008).

2.2 METODOLOGIAS DE DESENVOLVIMENTO TRADICIONAL

A atividade de "desenvolvimento de software" na década de 70 era executada de forma desorganizada, e sem planejamento, gerando um produto final de software de má qualidade, que não correspondia com as reais necessidades abordadas inicialmente pelo cliente, a partir desta problemática torna-se indispensável efetivar o desenvolvimento de software de forma estruturada, planejada e padronizada (PRESSMAN, 2001).

O Modelo em Cascata (ver **Figura 01**) surgido na década de 1970, tem como foco de desenvolvimento realizar uma sequência de atividades uma única vez, estruturadas como uma cascata, onde a saída de uma atividade é um produto para a entrada da atividade subseqüente (WILEY, 2002).

Figura 01: Modelo em Cascata

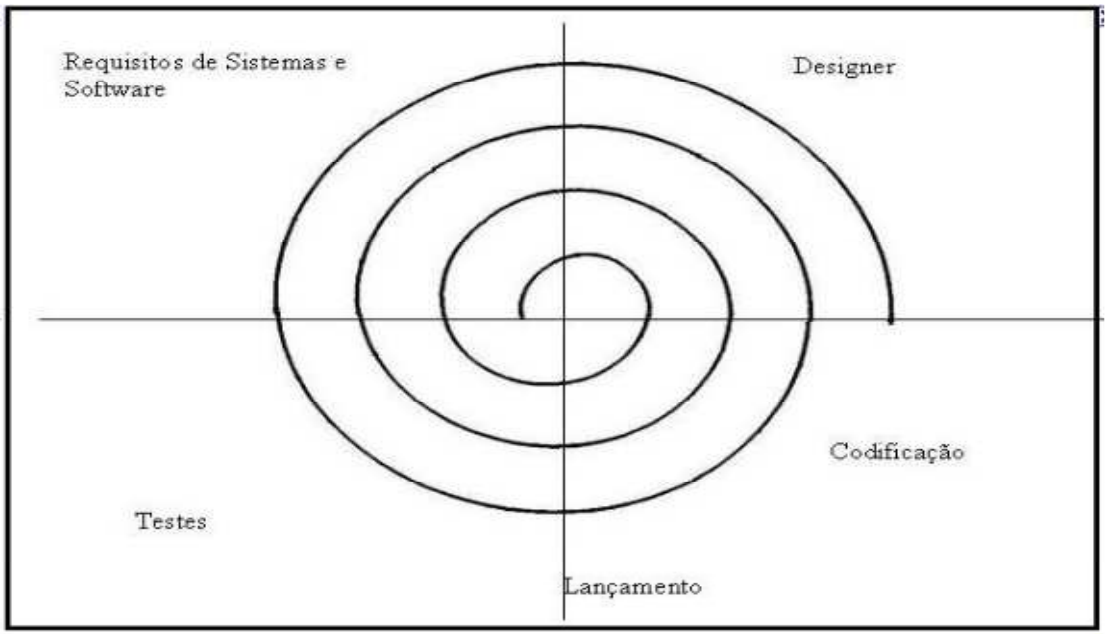


Fonte: Adaptado (WILEY, 2002).

O Modelo em Espiral (**ver Figura 02**) apresenta um conjunto de atividades de forma não seqüencial, onde cada loop na espiral cria uma nova versão de software. E em cada volta,

todas as atividades de análises, teste, projeto, planejamento e construção são repetidas e entregues ao cliente como nova versão, a maior vantagem desse modelo em relação a outros modelos tradicionais, é a sua capacidade de análise de risco no início do desenvolvimento (SOMMERVILLE, 2007).

Figura 02: Modelo em Espiral



Fonte: Adaptado (SOMMERVILLE, 2007).

Quando nos referimos a projetos de pequenas e médias empresas, e principalmente em um mercado altamente competitivo, ou em crise econômica, esses tipos de modelos são considerados desnecessários, o problema é que, dependendo do projeto, as metodologias tradicionais podem deixar os desenvolvedores amarrados a requisitos desatualizados, fazendo com que os sistemas em desenvolvimento sejam entregues após o tempo especificado pelo o cliente, ou até mesmo não concluídos (CAETANO, 2009).

2.3 METODOLOGIA DE DESENVOLVIMENTO ÁGIL

Esse modelo de desenvolvimento foi criado especificamente para apoiar a efetivação de aplicações de negócio, onde os requisitos de sistema são encontrados em uma constante modificação.

> (...) eles são mais adequados para o desenvolvimento de sistemas de pequenas e médias empresas e produtos para computadores pessoas. Eles não são tão adequados para o desenvolvimento de sistema de larga escala com as equipes de desenvolvimento em lugares diferente e onde possa haver interações complexas com outros sistemas de hardware e software (SOMMERVILLE, 2007).

Existe uma quantidade significativa de metodologias ágeis baseadas unicamente em desenvolvimento incremental e interativo, contudo apresentam e compartilham um mesmo conjunto de princípios (SOMMERVILLE, 2007):

- **Envolvimento do Cliente**: O cliente tem uma participação ativa no processo de desenvolvimento, tendo como papel fornecer e priorizar novos requisitos para implementação do sistema;
- **Entrega Incremental**: O software é desenvolvido em incrementos, cada incremento é resultado de um conjunto de requisitos atribuídos e priorizados pelo cliente;
- **Pessoas, não processos**: Cada membro da equipe pode apresentar habilidades diferentes que devem ser reconhecidas e exploradas de forma saudável, fazendo os mesmos trabalharem produtivamente e de forma confortável;
- **Aceite as Mudanças**: Projetar o sistema de software para possíveis mudanças, já que requisitos de sistema de software e principalmente de negócio estão numa constante variação;
- **Manter Simplicidade**: Procurar manter a concentração no desenvolvimento de um sistema de software simples, e trabalhar proativamente para eliminar às complexidades que podem aparecer no processo de desenvolvimento.

2.4 MODELOS DE DESENVOLVIMENTO APOIADORES DO YP (easyProcess)

2.4.1 **Extreme Programming (XP)**

2.4.1.1 Definição

O Extreme Programming (XP) é um modelo de desenvolvimento ágil de software desenvolvido por Kent Back (Engenheiro de Software americano formado em M. S. Licenciatura em Ciência da computação pela Universidade de Oregon) (GERVAZONI, 2005). Essa forma de construir software permite um grupo entre 2 a 10 programadores, projetos de 1 à 36 meses de duração, de 1000 à 250 000 linhas de código (FREIRE, 2003).

2.4.1.2 Princípio

O XP aborda os requisitos do sistema em "cartões de história", que por sua vez serão subdivididos em tarefas menores para facilitação do desenvolvimento (MONTEIRO, 2009). O desenvolvimento com XP leva em consideração alguns princípios básicos ou práticas que devem ser seguidos, a saber (KUHN e PALOMA, 2009):

- **Planejamento:** O desenvolvimento é realizado por semana o cliente juntamente com os desenvolvedores se reúnem semanalmente para que sejam definidos os requisitos de maior prioridade;
- **Pequenas Liberações:** As funcionalidades de maior prioridades são implementadas, fazendo essas prioridades serem atribuídas ao sistema como incrementos, sendo assim, pequenas releases são atribuídas constantemente ao sistema;

- **Metáfora:** Uma problemática encontrada na aquisição de requisitos é unicamente entender o problema do cliente para então se atribuir uma solução, pensando nisso é apoiado o uso de metáforas para que os membros da equipe possam compartilhar a mesma linguagem técnica;
- **Projeto Simples:** Fazer a equipe de desenvolvimento manter o foto no desenvolvimento de um sistema simples, por exemplo, se é dado a um programador para codificar um meio de acesso ao sistema com a senha "123456", é desnecessário o programador elaborar um meio sofisticado, como acesso biométrico;
- **Testes de Aceitação:** São particularmente definidos pelo cliente juntamente com os testadores, são essenciais para comprovar as funcionalidades de uma determinada parte do sistema;
- **Ritmo Sustentável:** A equipe tem que procurar uma forma saudável de trabalho, sem exceder à hora normal, o XP recomenda uma quantidade máxima de 40 horas semanais, por exemplo, de segunda a sexta-feira com oito (8) horas por dia, voltando o funcionário na próxima segunda-feira, cheio de novas idéias e muita motivação;
- **Posse Coletiva:** O sistema é visto como propriedade coletiva de código, ou seja, todos os membros da equipe podem fazer possíveis mudanças ou incrementos no código produzido pelo um membro da equipe sem muitas dificuldades;
- **Programação em Pares:** A programação de um código é realizado por duas pessoas no mesmo computador enquanto uma pessoa digita o código a outra vai instruindo, essa técnica ajuda a produzir código sem ou com o mínimo de erros possíveis, por causa do refatoramento constante;
- **Padrões de Codificação:** Os códigos devem ser redigidos seguindo a mesma padronização, isso permite aos programadores entender bem o que eles produzem;
- **Refatoração**: Realizar um refinamento no sistema, eliminar códigos e fontes repetidas, deixando mais claro, enxutos, coesos e com a mesma funcionalidade;
- **Integração Contínua:** Como a programação é realizada de forma incremental, torna-se essencial adicionar uma nova funcionalidade a cada versão atribuída de forma contínua até a finalização do projeto.

2.4.2 Rational Unified Process (RUP)

2.4.2.1 Definição

O RUP (processo unificado racional) é um modelo proprietário de desenvolvimento de software desenvolvido pela Rational Software Corporation e adquirido em fevereiro de 2003 pela atual pertencente IBM, apesar de ser considerada uma maneira de produzir software pesada por algumas organizações, é uma das formas mais disciplinadas e usadas por grandes equipes de desenvolvimento, considerada um modelo customizável por causa de sua adaptabilidade à projetos de qualquer escala (MARTINESES, 2010).

2.4.2.2 Principio

A filosofia RUP pode ser melhor esclarecida ao analisarmos alguns de seus princípios clássicos, a saber (TANAKA e BANKI, 2008):

- **Desenvolvimento iterativo**: Intuitivamente não se dá para desenvolver um software em um único passo, pois o sistema pode passar por constantes alterações, questões relacionadas a arquitetura, usuário, e até mesmo um maior entendimento do problema, tudo isso amarrado a uma ou a conjunto de iterações, que tem como principal foco fazer refinamentos no projeto;
- **Gerenciamento de requisitos**: Na prática todo modelo de desenvolvimento de software passa pelo levantamento de requisitos, no RUP não é muito diferente, entretanto, ele faz todo um gerenciamento de requisitos, tais como: análise do problema, definição problema, mudança de requisitos, escopo do problema, entre outros;
- **Arquitetura baseada em componentes**: Componentes são basicamente bibliotecas de software já prontas, onde os desenvolvedores têm a única função de usá-las e retirar o

maior aproveitamento possível, promovendo de forma contínua a reusabilidade de software;

- **Modelagem visual de software**: O uso de UML (linguagem de modelagem unificada) é apoiada pelo RUP, definir toda a lógica de um problema em diagramas facilita muito o seu entendimento, é uma forma intuitiva de analisar o funcionamento do sistema, abstraindo de forma simplificada o entendimento de um projeto complexo;
- **Qualidade de software:** Foco na qualidade é uma tarefa indispensável para qualquer modelo de desenvolvimento, no RUP não seria diferente, entretanto ele garante qualidade em todo processo de desenvolvimento, fazendo com que as fases do ciclo de vida sejam todas observadas por cada membro da equipe;
- **Alterações no software**: Em qualquer projeto mudanças são imprevisíveis, vendo esse tipo de problemática o RUP apóia o processo de planejamento direcionado a mudanças, como: espaço de trabalho seguro (forma de garantir a confiabilidade de um sistema).

2.4.3 Agile Modeling (AM)

2.4.3.1 Definição

A princípio o Agile Modeling não é uma metodologia de desenvolvimento como XP, RUP, mas uma forma de modelagem que pode ser usada paralelamente a uma metodologia de desenvolvimento tanto em caráter ágil como prescritiva. Caracteristicamente não é definida uma forma de modelagem específica, apresenta como função orientações aos membros da equipe para que sejam mais efetivos ao projeto.

> "(...) é um conjunto de valores, princípios e práticas de modelagem de software que pode ser aplicado em um projeto de desenvolvimento de software em uma forma leve de peso eficaz (...)" (AMBLER, 2009).

2.4.3.2 Princípios

Agile Modeling define uma coleção de princípios que devem ser levados em consideração durante o processo desenvolvimento de software, logo abaixo alguns princípios: (AMBLER, 2009):

- **Maximizar Stakeholder:** É uma abordagem que deve ser usada para manter os investidores do projeto informados sobre o processo de construção do software, pois os interessados no desenvolvimento investem muitos recursos, como: tempo, dinheiro, entre outras questões relacionadas;
- **Comentários rápidos**: Cada ação atribuída a modelagem deve ser constituída unicamente por intervalo de tempo mínimo destinados a possíveis comentários que ajudam a produzir um feedback necessário entre os membros da equipe;
- **Suponha simplicidade:** Não apresentar termos adicionais em seu sistema é uma questão importantíssima, manter sempre simplicidade é essencial no processo de modelagem para uma maior coerência do sistema;
- **Trabalho de Qualidade:** As pessoas quando gostam do que fazem tendem a produzir o melhor trabalho possível, os membros da equipe têm que produzir modelos com foco na qualidade que possam ser facilmente analisados e modificados por outras pessoas.

2.4.3.3 Práticas

Agile Modeling define uma coleção de práticas que devem ser adotadas no processo de desenvolvimento de software, a seguir algumas práticas (AMBLER, 2009):

- **Participação ativa dos interessados:** Nesta categoria a construção de um site onde os membros da equipe possam ver todo o andamento do projeto, como também apresentar uma participação bem mais significativa;
- **Uso de ferramentas simples:** Evitar o uso de ferramentas complexas é o objetivo principal desta categoria, boa parte dos modelos podem ser projetados até mesmo em

um pedaço de papel, então para quê desperdiçar tempo em uma ferramenta complicada;

- **Modelo em pequenos incrementos:** Todo o problema de modelagem não é necessariamente resolvido de uma só vez, então em um determinado período toda a modelagem pode ser resolvida com a definição de incrementos;
- **Criar conteúdo simples:** Torna-se essencial para os membros da equipe criarem seus conteúdos o mais simples possível, evitar incrementos desnecessários, é uma virtude que a equipe deve seguir.

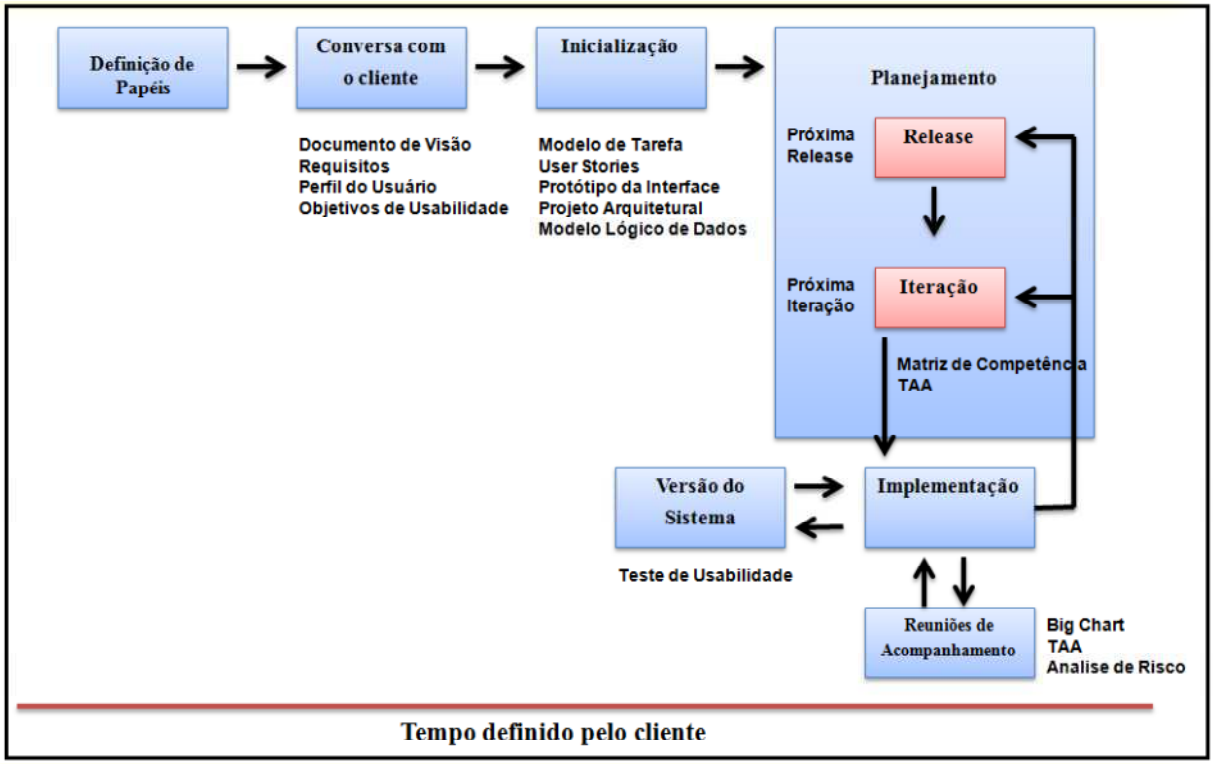
Sendo o easYProcess foco deste trabalho de conclusão de curso (TCC), o mesmo será apresentado na próxima seção, onde é abordado todos os processos de forma simples e intuitiva, e que são consideradas indispensáveis para efetivação de um componente de software de qualidade.

3 EASY PROCESS (YP)

O easYProcess é uma metodologia de desenvolvimento de software ágil criada pelo grupo PET[2] da Universidade Federal de Campina Grande (UFCG), sendo idealizada pela Professora Drª Francilene Procópio Garcia (SILVA, 2010). Está metodologia foi criada com o intuito de auxiliar os alunos do curso de Ciência da Computação no desenvolvimento e efetivação de seus projetos de software ofertados pelas disciplinas no decorre do ano letivo (GARCIA, 2007).

O fluxo de trabalho do easYProcess, segundo Garcia (2007), são descritas logo abaixo, **(ver Figura 03).**

Figura 03: Síntese do Fluxo do Processo YP



Fonte: Adaptado (GARCIA, 2007).

[2] O Grupo PET (Programa de Educação Tutorial) do Curso de Graduação em Ciência da Computação da Universidade Federal de Campina Grande (UFCG) foi criado em 1992. [Disponível em: <http://www.dsc.ufcg.edu.br/~pet/>. Acesso : 13/05/11]

3.1 DEFINIÇÃO DE PAPÉIS

O processo é iniciado com a definição de papéis (função de uma pessoa no processo de desenvolvimento), neste momento os participantes da equipe de desenvolvimento são listados em uma tabela juntamente com suas habilidades que ajudaram no desenvolvimento do projeto.

Segundo Garcia (2007) dependendo da quantidade de pessoas na equipe, ou habilidades pertencentes a uma mesma pessoa, um único membro poderá desempenhar mais de um papel paralelamente, e desta forma estruturando a melhor equipe possível para se obter uma maior produtividade no processo.

Segue a baixo os papéis adotados pelo YP:

- Gerente: Membro da equipe que gerencia o processo de desenvolvimento, analisa e toma decisões referentes ao andamento do projeto. Segundo Garcia (2007) as competências do Gerente são: conduzir os planejamentos e as ações dos desenvolvedores, elaborar o plano de desenvolvimento, avaliar sistematicamente os riscos descobertos, coletar e analisar métricas, alocar testadores, presidir as reuniões de acompanhamento, resolver conflitos internos, tornar a documentação do projeto sempre acessível e atualizada;
- Desenvolvedor: Membro da equipe responsável pela efetivação dos requisitos em códigos executáveis. De acordo com Garcia (2007) as competências do Desenvolvedor são: levantar os requisitos funcionais e não-funcionais, auxiliar o gerente na elaboração de um plano de desenvolvimento, analisar e modelar a tarefa, gerar um protótipo de interface, elaborar o projeto arquitetural, construir um modelo lógico de dados;
- Testador: Membro da equipe que efetiva a maioria dos testes do projeto, além de defini-los juntamente com o cliente. Afirma Garcia (2007), competências do Testador são: revisar o código de outras pessoas, elaborar testes sobre o código de outros desenvolvedores, refatoramento de código, gerar teste de aceitação e elaborar o material para realização dos testes de usabilidade;

- Cliente: É considerado membro da equipe por causa de sua participação efetiva no projeto, é quem adquire o sistema de software para solucionar um determinado problema. Constata Garcia (2007) as competências do Cliente são: definir os requisitos do sistema, priorizar as funcionalidades, ajudar no plano de release, identificar o perfil do usuário, identificar os objetivos de usabilidade, validar o protótipo da interface, validar o projeto arquitetural, ser ativo no processo de desenvolvimento;
- Usuário(s): Têm participações significativas no processo de desenvolvimento, é basicamente a pessoa para quem se destinará o produto de software. Segundo Garcia (2007), competências do Usuário são: ajudar a definir os testes de aceitação, ajudar a identificar os objetivos de usabilidade, validar o protótipo da interface, avaliar a interface do sistema.

3.2 CONVERSA COM O CLIENTE

Em seguida vem a conversa com o cliente, neste momento os desenvolvedores devem extrair o máximo de informações do cliente para que sejam produzidos todos os artefatos necessários e indispensáveis a efetivação do projeto.

Afirma Garcia (2007), antes da conversa os desenvolvedores devem ter um entendimento breve do problema para que possam elaborar um roteiro de perguntas eficientes ao cliente. Durante a conversa deve-se usar termos simples para evitar complicações, usar o máximo de analogias quando necessário, de forma a exemplificar o problema, deve-se também deixar bem claro ao cliente a sua função no processo.

3.2.1 Documento de visão

De acordo com Garcia (2007), os desenvolvedores elaboram um documento especificando a descrição do sistema que será produzido, esse documento deve apresentar de forma geral e objetiva, o que o sistema se propõe a fazer para solucionar o problema do cliente.

3.2.2 Requisitos

Os requisitos é um conjunto de especificações referentes ao projeto a ser desenvolvido, ou seja, são todas as funções e características que o sistema deverá apresentar depois de finalizado. Destacando basicamente duas categorias de requisitos, os funcionais que são responsáveis por todas as funções que sistema deverá apresentar para efetivação das atividades administrativas, e não funcionais que são responsáveis por todas as características que o sistema deverá apresentar (GARCIA, 2007).

3.2.3 Perfil do Usuário

Trata-se de uma lista de informações provenientes das características do usuário, essas informações são consideradas indispensáveis para elaboração do projeto. Segundo Garcia (2007), o YP afirma as seguintes características:

- Sexo;
- Se canhoto, destro ou ambidestro;
- Faixa etária;
- Experiência prévia no uso de sistemas computacionais;
- Etc.

3.2.4 Objetivos de Usabilidade

De acordo com Garcia (2007), o sistema depois de finalizado deve apresentar as especificações, a saber:

- Eficácia;
- Eficiência;
- Segurança;
- Memorização.

3.3 INICIALIZAÇÃO

Este processo permite que os membros da equipe possam ver o futuro sistema em diversos aspectos, a saber: Modelo de tarefa, Protótipo de interface, User Stories e Testes de aceitação, Projeto arquitetural, Modelo lógico de dados. De acordo com Garcia (2007): "investir tempo aqui é muito importante, pois quanto maior for o entendimento do sistema, menos problema se terá na etapa de implementação".

3.3.1 Modelo de Tarefa

É uma representação gráfica de como as tarefa serão realizadas, a análise da tarefa permite aos desenvolvedores terem maior compreensão de como o usuário irá interagir com a interface do sistema, ajudará também na elaboração do protótipo da interface. Afirma Garcia (2007): "O modelo de tarefa pode ser construído de acordo com vários formalismos, no entanto, o YP sugere o uso do TAOS[3]".

3.3.2 User Stories e Teste de Aceitação

São os requisitos do cliente definidos de forma mais especifica, a cada User Story é alocado pelo menos um Teste de Aceitação (condição mínima da funcionalidade definida pelo cliente), são listadas pelo cliente com orientação dos desenvolvedores, e estruturadas na tabela em ordem de priorização, quando uma User Story é muito grande é dividida em partes menores para facilitar o processo de implementação (GARCIA, 2007).

3.3.3 Protótipo da Interface

[3]Task and Action Oriented System é uma ferramenta CASE para ajudar na arquitetura do sistema.

É uma representação gráfica da interface do sistema que permite aos membros da equipe se comunicarem em uma linguagem comum, nesta etapa a interface não precisa ser funcional. Especifica Garcia (2007): "a construção de um esboço de interfaces simples que possam ser produzidas com baixo custo de investimento, facilita maior entendimento entre a interação de sistema e usuário."

3.3.4 Projeto Arquitetural

De acordo com Garcia (2007), é a forma de visualizar o sistema em um alto nível de abstração sendo útil quando se deseja explicar o funcionamento entre as partes do sistema, é essencial que esse artefato seja gerado em forma de diagrama para facilitar o entendimento dos membros da equipe, ao se concluir o artefato ele deve mostrar a estrutura do sistema e relacionamento existente entre seus módulos.

3.3.5 Modelo Lógico de Dados

Trata-se unicamente da modelagem de entidades referentes ao problema específico, caso o sistema a ser desenvolvido necessite de uma forma de guardar e manipular informações.

3.4 PLANEJAMENTO

É o cronograma de todas as atividades que serão realizadas no processo de implementação, sabendo-se da estimativa de tempo necessária para a conclusão do projeto, os números de Releases e Iterações devem ser especificados pelos membros da equipe. Segundo Garcia (2007): "A fase de planejamento é composta por dois planejamentos, o de Release e o de Iteração, no qual existem 3 releases, cada qual contendo 2 iterações de 2 semanas cada."

No planejamento, o plano de release é formado basicamente por iterações, onde cada iteração é constituída por um conjunto de User Stories priorizadas pelo cliente na fase de inicialização; no plano de iteração as User Stories se necessária são divididas em atividades menores para facilitar o processo de implementação, e alocadas a cada membro da equipe de acordo com suas habilidades para efetivação da mesma (GARCIA, 2007).

3.5 IMPLEMENTAÇÃO

É a codificação das atividades presentes na iteração, para se ter um bom processo de implementação os desenvolvedores devem seguir algumas práticas, a saber: Integração contínua, Boas práticas de codificação, Propriedade coletiva de código, Testes e Pequenos releases (GARCIA, 2007).

3.5.1 Integração continua

Tem como objetivo facilitar o gerenciamento do projeto e o trabalho dos desenvolvedores, em relação ao gerenciamento do projeto essa prática auxilia na coleta de métricas, como também na elaboração do Big Chart (tabela de informações sobre o processo de desenvolvimento, em relação aos desenvolvedores essa prática é bem observada quando os mesmos não possuem horários de trabalho em comum.

Cada código produzido e testado deve ser integrado continuamente ao sistema como parte de um todo, o easYProcess recomenda o uso de algumas ferramentas no auxilio da integração contínua, a saber: CruiseControl[4], Apache Gump[5].

3.5.2 Boas Práticas de Codificação

De acordo com Garcia (2007), os membros da equipe de desenvolvimento devem ter em mente a idéia de código limpo e principalmente de fácil entendimento, para tal o YP adota o seguinte conjunto de práticas, a saber:

[4] É tanto uma ferramenta de integração contínua e um framework extensível para a criação de um contínuo processo de compilação personalizada. [CruiseControl]

[5] Constrói e compila software contra as últimas versões de desenvolvimento desses projetos. [Apache Gump]

- Design Simples: O código deve apresentar um fácil entendimento, ser alto-explicativo, com uso de comentários apenas essenciais;
- Padrões de Codificação: Ajudam na estruturação do código de forma visual possibilitando um maior entendimento por parte dos desenvolvedores. Por exemplo: a forma de posicionamento dos parênteses, o meio de nomear variáveis e métodos, formam um padrão de codificação aceitável entre os membros da equipe;
- Padrões de Projeto: Consiste no uso de soluções computacionais previamente elaboradas e testadas por grande projetistas, o uso de padrões faz com que os membros da equipe ganhem tempo no desenvolvimento, e consequentemente eficiência e qualidade no projeto;
- Refatoramento: É basicamente uma modificação em determinada parte do código do sistema, essa modificação não pode alterar o comportamento funcional, entretanto são apresentadas modificações significativas em termos não funcionais, a saber: simplicidade, flexibilidade, clareza do código.

3.5.3 Propriedade Coletiva de Código

Essa prática é uma forma de melhoramento contínuo de código, que é visto como uma propriedade coletiva, ou seja, de todos os membros da equipe, isso significa que um membro da equipe de desenvolvimento pode alterar e acrescentar o código produzido por outro membro sem muitas dificuldades.

3.5.4 Testes

De acordo com Garcia (2007), o YP recomenda o uso de três tipos de testes, a saber:

- Testes de Unidade: É uma categoria de teste indispensável para um bom funcionamento do código interno, esse tipo de teste analisa a estrutura interna do código, ou seja, tanto a parte lógica e o fluxo de dados;

- Testes de Aceitação: São indispensáveis no processo de elaboração das User Stories, o YP recomenda pelo menos um teste de aceitação para cada User Story, a principal característica desse tipo de teste é ele ser atribuído pelo cliente, que por sua vez é orientado pela equipe de desenvolvimento;
- Teste de Usabilidade: Essa categoria de teste é executada em duas etapas, na primeira o usuário é submetido ao sistema produzido com um questionário de atividades referentes às funções que o sistema deve cumprir, na segunda etapa é aplicado outro questionário, sendo este referente à satisfação subjetiva do usuário.

3.5.5 Pequenos Releases

O YP recomenda a alocação dos requisitos em pequenos releases para que se tenha uma maior facilidade no processo de implementação, o ideal é que cada release seja constituída de duas iterações e estas com conjunto de User Stories (GARCIA, 2007).

3.6 REUNIÃO DE ACOMPANHAMENTO

Sendo organizadas uma vez por semana pelo gerente as reuniões de acompanhamento tem como objetivo analisar sistematicamente o andamento do projeto. Um gerente comprometido com o projeto nessas reuniões pode identificar previamente falhas no processo de desenvolvimento e promover junto à equipe possíveis soluções. Segundo o Garcia (2007), toda equipe de desenvolvimento deve participar das reuniões semanais, pois a mesma se dará em torno do material produzido no processo de desenvolvimento, assegurando as atividades a seguir:

- Big Chart: Constitui em uma tabela com informações para análise quantitativa do projeto;
- Tabela de Alocação de Atividade (TAA): Constitui em uma tabela com informações referentes às atividades realizadas pelos membros da equipe;
- Análise de Risco: Consiste em todos os empecilhos encontrados no processo de desenvolvimento que podem prejudicar significativamente o produto final de software;
- Mudança: São possíveis alterações, existe vários tipos: as que exigem modificações de implementações, no modelo de tarefa, no projeto arquitetural, no modelo lógico de dados, nas User Stories, na Interface, na modificação de membros da equipe, até mesmo no documento de visão.

Esta seção apresentou uma abordagem metodologia referente ao modelo de desenvolvimento ágil em questão, visando ao leitor conhecimento especifico para próxima seção, onde é usado na prática o uso dessa metodologia.

4 SYS VENDA E YP (easYProcess)

Nesta seção é abordado o desenvolvimento de um sistema para controle de vendas de produtos (SysVenda), com auxilio de um modelo de desenvolvimento ágil (easYProcess).

4.1 DEFINIÇÃO DE PAPÉIS

Lembrando que este projeto trata-se de um trabalho de conclusão de curso (TCC), um único membro desempenha os papéis referentes a uma equipe de desenvolvimento**, (ver Quadro 01).**

Quadro 01: Definição de papéis

Equipe	Papéis
Elder G. Pereira	Gerente, desenvolvedor, testador, cliente e usuário.

Fonte: Autor da Pesquisa, 2011

O próprio YP afirma que um mesmo integrante da equipe pode desempenhar mais de um papel (função) simultaneamente, mas que também não abre mão de uma boa equipe no processo de desenvolvimento.

4.2 CONVERSA COM O CLIENTE

Depois da primeira conversa com o cliente os desenvolvedores elaboram uma série de artefatos essenciais para o andamento do projeto, a saber: Documento de visão (descrição do sistema a ser produzido); Requisitos funcionais e não funcionais; Perfil do usuário e Objetivos de usabilidade.

4.2.1 Documento de visão

A proposta é o desenvolvimento de um sistema para a manipulação de informações de uma determinada empresa de vendas de produtos, o SysVenda será um sistema de informação desenvolvido para funções especificas e indispensáveis na manipulação de funcionários, clientes, fornecedores, vendas, produtos, entre outros recursos auxiliares no processo administrativo.

Para a efetivação do funcionamento do SysVenda, torna-se necessário que o mesmo seja projetado em arquitetura WEB, onde funcionários da empresa poderão desempenhar funções administrativas sem necessariamente ter o sistema instalado nas suas maquinas.

4.2.2 Requisitos

Depois da conversa com o cliente os desenvolvedores já devem ter em mente uma idéia inicial sobre o sistema que será produzido, neste momento é gerado os requisitos funcionais e não funcionais do sistema.

Requisitos Funcionais:

- Controle de venda;
- Cadastro de cliente, fornecedor, funcionário e produto;

- Editar cliente, fornecedor, funcionário e produto;
- Listar cliente, fornecedor, funcionário e produto;
- Confirmação de venda;
- Gráfico para controle de venda.

Requisitos não-funcionais

- Interface WEB (JSP + JSF + HTML);
- Tratando-se de sistema de informação torna-se indispensável, segurança com LOGIN;
- Confiabilidade;
- Integridade;
- Componentes RichFaces;
- Componente para persistência de dados usando o HIBERNATE;
- Padrões de projeto (DAO e MVC).

4.2.3 Perfil do Usuário

Usuários de baixo nível de conhecimento em informática, mas que podem demonstrar muita capacidade de aprendizagem no uso do sistema, visto que a interface (interação homem maquina) será projetada de forma intuitiva, analisando essa perspectiva à disponibilização de um treinamento torna-se indispensável para os usuários se familiarizarem com as funções do SysVenda.

4.2.4 Objetivo de Usabilidade

A seguir são apresentados os objetivos usuais que o sistema deve alcançar, (**ver Quadro 02**).

Quadro 02: Objetivo de usabilidade

OBJETIVO	**MENSURAÇÃO**
Aumentar nível de segurança.	Tornar as informações do processo

	administrativo longe de pessoas não autorizadas.
Eficiência para controle de informações.	Processo administrativo amigável, para o cadastro, listagem e edição de informações.
Aumento de produtividade das vendas.	Realização e confirmação de venda, de forma intuitiva.
Possuir uma Interface simples.	Assegurar ao usuário maior compreensão na identificação de componentes na interface.
Garantir a validação dos campos da Interface.	Fazer as informações na interface ter um valor significativo.
Manter a estrutura do sistema de forma clara.	Usar termos referentes ao tipo de processo administrativo da equipe de usuários.

Fonte: Autor da Pesquisa, 2011

Este item é importante para todos os membros da equipe de desenvolvimento tentarem manter a fidelidade na construção do sistema, e para que não percam o foco no que o sistema deve apresentar na sua fase final.

4.3 INICIALIZAÇÃO

4.3.1 USER STORIES (histórias de usuário) e TESTE DE ACEITAÇÃO

A seguir será mostrado todos os requisitos funcionais de forma mais especifica a serem implementados, são listados e especificados com uma estimativa de tempo inicial para cada User Store (história de usuário), e estruturados no quadro em ordem de priorização de implementação, é indispensável frisar que para cada história de usuário é definido pelo menos um teste de aceitação que devem ser todos indicados, analisados e aprovados pelo cliente, com orientação dos desenvolvedores e testadores (**ver Quadro 03**).

Quadro 03: User Stories

US01	Ampliar conhecimento nas seguintes tecnologias, a saber: JSF, Rich Faces, MySql, JFreeChart, HIBERNATE, MVC, DAO Genérico. **Estimativa Inicial: 20 h**
TA1.1	Elaborar exemplos referentes às tecnologias citadas, efetivando maior conhecimento prático.
US02	Implementar os modelos de classes referentes as entidades do sistema, e a classe DAOGenerica responsável para operações CRUD do sistema. **Estimativa Inicial: 10 h**
TA2.1	Configuração da plataforma de desenvolvimento, com todos os plugins indispensáveis para a efetivação do sistema.
TA2.2	Confirmar as classes necessárias na camada de modelo.
TA2.3	Confirma classe responsável pelas operações CRUD.
TA2.4	Verificação de todas as bibliotecas necessárias, para uso das tecnologias.
US03	Implementar modulo de venda. **Estimativa Inicial: 10 h**
TA3.1	Verificar se o cliente está sendo disponibilizado como opção.
TA3.2	Verificar se funcionário estar sendo disponibilizado como opção.
TA3.3	Verificar se o produto está sendo disponibilizado como opção.
TA3.4	Realizar venda com todos os campos (operação com sucesso).
TA3.5	Realizar venda sem todos os campos (operação não realizada).
TA3.6	Testar o campo Unidade para que liste a quantidade de produto em estoque.
TA3.7	Verificar se o produto selecionado está disponibilizando a quantidade certa em estoque, e realizando o cálculo correto.
US04	Implementar listas de todas as vendas não pagas, com ações para confirmar , desfazer venda.

	Estimativa Inicial: 6 h
TA4.1	Verificar se está listando todas as vendas não pagas.
TA4.2	Testar opção de confirmação de vendas (confirmação em nova janela).
TA4.3	Testar o desfazer de venda (ação realizada com sucesso)
TA4.4	Verificar se o gráfico gerado está de acordo com as vendas pagas e pendentes.
US05	Implementar o cadastro de fornecedor. **Estimativa Inicial: 8 h**
TA5.1	Cadastro de um fornecedor com todos os campos (operação com sucesso).
TA5.2	Cadastro de fornecedor apenas com os campos obrigatórios (operação com sucesso).
TA5.3	Cadastro de fornecedor faltando um campo obrigatório (operação não realizada).
TA5.4	Cadastro de fornecedor faltando todos os campos obrigatórios (operação não realizada).
US06	Implementar o cadastro de produto. **Estimativa Inicial: 8 h**
TA6.1	Verificar se o fornecedor está sendo disponibilizado como opção.
TA6.2	Cadastro de um produto com todos os campos (operação com sucesso).
TA6.3	Cadastro de um produto com campos obrigatórios (operação com sucesso).
TA6.4	Cadastro de um produto sem os campos obrigatórios (operação não realizada).
US07	Implementar o cadastro de cliente. **Estimativa Inicial: 8 h**
TA7.1	Cadastro de um cliente com todos os campos (operação com sucesso).
TA7.2	Cadastro de cliente apenas com os campos obrigatórios (operação com sucesso).
TA7.3	Cadastro de cliente faltando um campo obrigatório (operação não realizada).
TA7.4	Cadastro de cliente faltando todos os campos obrigatórios (operação não realizada).
US08	Implementar o cadastro de funcionário. **Estimativa Inicial: 8 h**
TA8.1	Cadastro de um funcionário com todos os campos (operação com sucesso).
TA8.2	Cadastro de funcionário apenas com os campos obrigatórios (operação com sucesso).
TA8.3	Cadastro de funcionário faltando um campo obrigatório (operação não realizada).
TA8.4	Cadastro de funcionário faltando todos os campos obrigatórios (operação não realizada).
US09	Implementar funcionalidade para listas de: cliente, funcionário, fornecedor e produto. **Estimativa Inicial: 22 h**
TA9.1	No módulo busca de cliente por nome (cliente retornado com sucesso).
TA9.2	No módulo busca de funcionário por nome (funcionário retornado com sucesso).
TA9.3	No módulo busca de fornecedor por nome (fornecedor retornado com sucesso).
US10	Implementar funcionalidade de alteração, nos tipos de lista. **Estimativa Inicial: 10 h**

TA10.1	Editar cadastro com todos os campos (cadastro realizado com sucesso).
TA10.2	Editar cadastro faltando os campos obrigatórios (cadastro não realizado).
TA10.3	Editar cadastro faltando qualquer campo obrigatório (cadastro não realizado).
TA10.4	Verificar modificação do botão confirmar para o botão voltar.
US11	Implementar funcionalidade para autenticação de usuários. **Estimativa Inicial: 5 h**
TA11.1	Entrar no sistema a partir de login valido (autenticação realizada).
TA11.2	Entrar no sistema a partir de login invalido (autenticação não realizada).

Fonte: Autor da Pesquisa, 2011

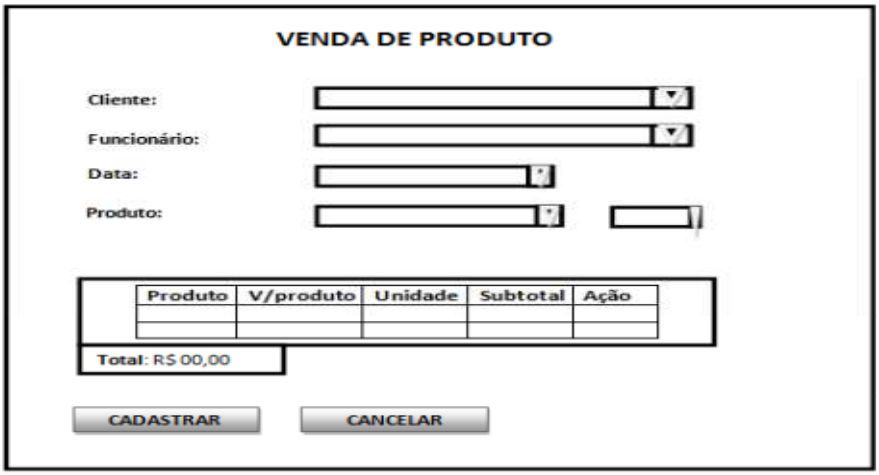
4.3.2 PROTÓTIPO DA INTERFACE

O easYProcess recomenda um protótipo inicial das interfaces do sistema a ser desenvolvido para que os membros da equipe tenha um maior entendimento de como se dará a interação entre usuário e sistema, e também abordar umas idéias de design antes de gerar as interfaces executáveis, logo abaixo são apresentadas as telas iniciais do sistema a ser desenvolvido (**ver Figura 04, Figura 05, ver Figura 06 e Figura 07**).

Figura 04: Cadastro de Produto

Fonte: Autor da Pesquisa, 2011

Figura 05: Venda de Produto

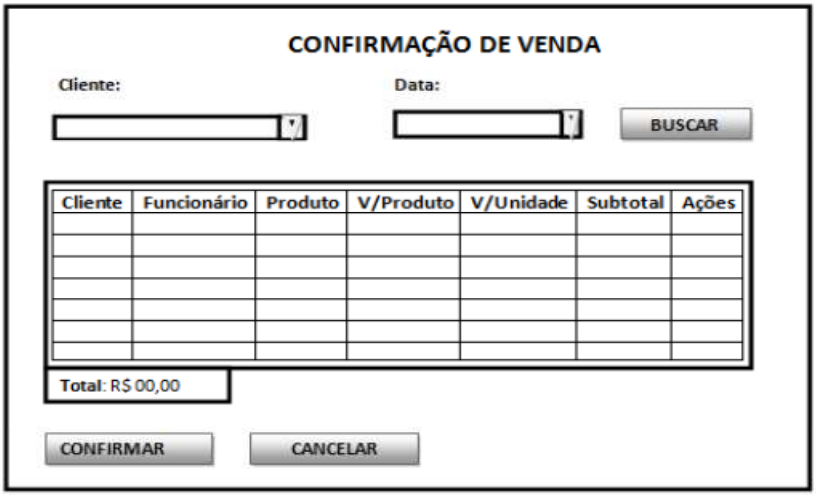


Fonte: Autor da Pesquisa, 2011

Figura 06: Lista de Produto

Fonte: Autor da Pesquisa, 2011

Figura 07: Confirmação de Venda

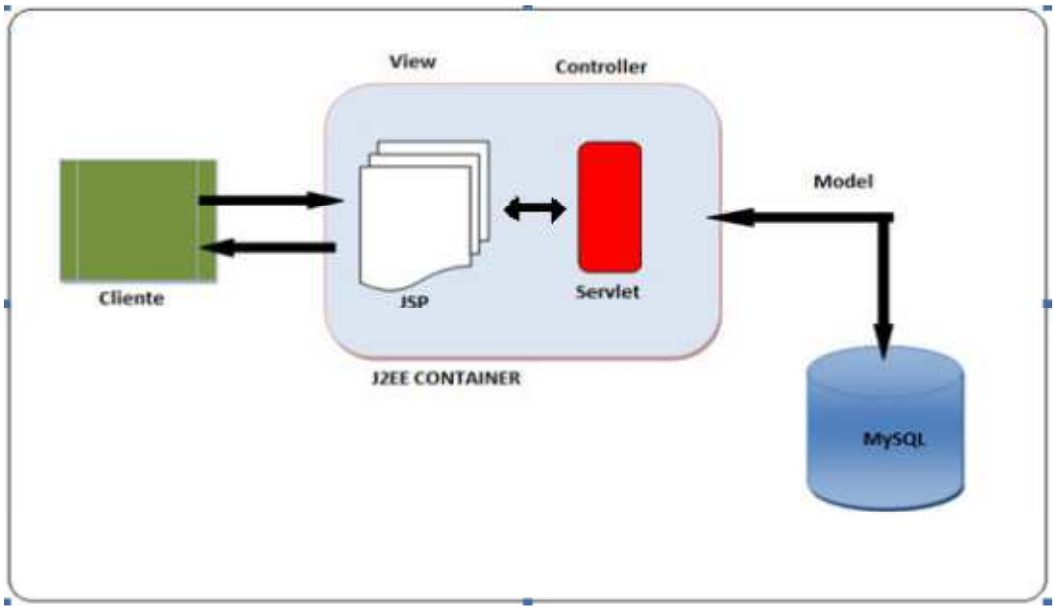


Fonte: Autor da Pesquisa, 2011

O YP que as interfaces devem ser geradas da forma mais simples possível, ou seja, sem uso de ferramentas complexas.

4.3.3 PROJETO ARQUITETURAL

Figura 08: Modelo Arquitetural



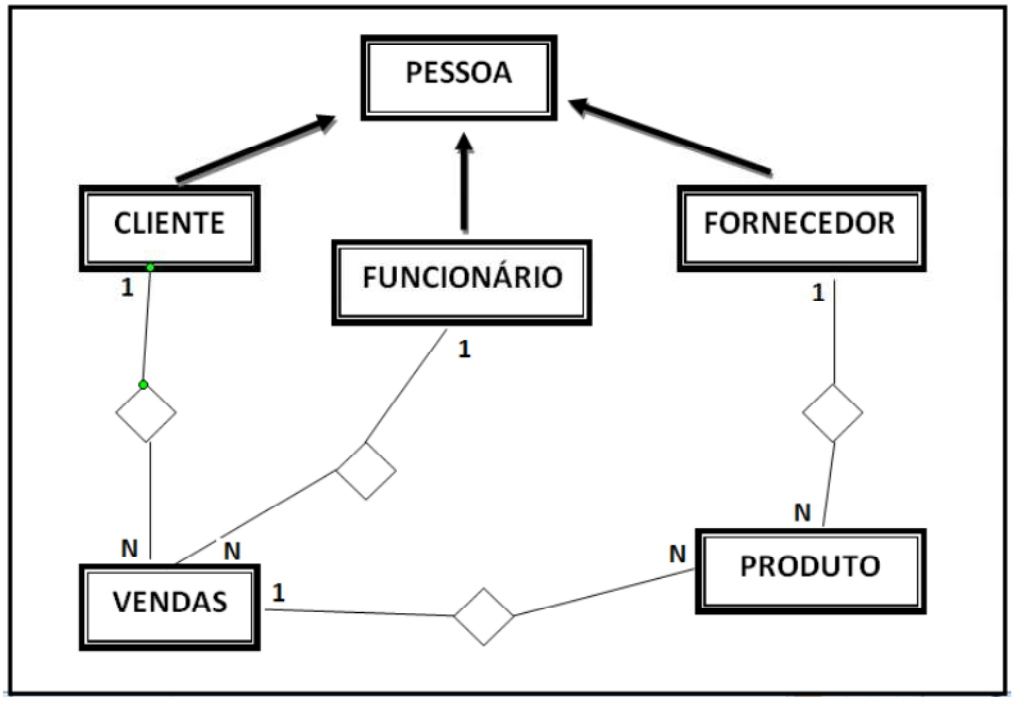
Fonte: Autor da Pesquisa, 2011

O software será produzido a partir de tecnologias WEB Abertas (Open Source), onde os desenvolvedores podem consegui lá sem nenhum custo. A arquitetura é baseada com componentes JSF que por padronização apóia o padrão de projeto MVC (model, view and controller), o cliente ao fazer suas requisições ao servidor, induz a camada de controle redirecionar uma chamada para a camada de modelo que, por sua vez envia uma resposta ao cliente na camada de apresentação com intermédio da camada de controle **(ver Figura 08).**

Para SGBD (sistema de gerenciamento de banco de dados) foi escolhido o MySQL, um dos mais usados banco de dados na WEB, isso devido: desempenho, portabilidade, facilidade de uso, exige pouco recurso de hardware, segurança, entre outros requisitos necessários para um padrão de usabilidade.

4.3.4 MODELO RELACIONAL

Figura 09: Modelo Relacional



Fonte: Autor da Pesquisa, 2011

Ao se tratar de manipulação de registros referentes a uma boa administração da empresa do cliente, torna-se indispensável um modelo relacional, então para o problema do cliente é definido uma base de dados com seis (6) entidades: temos a tabela Pessoa que é extensível para as tabelas Cliente, Funcionário e Fornecedor; tem a tabela Produto atribuída pelos fornecedores; tem tabela Vendas que faz uma venda para um Cliente de um conjunto de produto por um determinado Funcionário do plano de negocio do cliente (**ver Figura 09**).

4.4 PLANEJAMENTO

Depois do artefato de Inicialização ter sido finalizado e aprovado pelo cliente com as especificações necessárias para o andamento do projeto, parte-se para as fases seguintes, o Plano de Releases e Iteração. É relevante salientar que os períodos definidos nestes são de tempo fixo, ou seja, o projeto pode passar por grandes modificações, mas os tempos definidos nos mesmos devem permanecer constantes, isso ajuda a manter credibilidade com cliente.

4.4.1 MATRIZ DE COMPETÊNCIA

Na elaboração das Iterações é recomendado uso da Matriz de Competência (quadro para especificar as capacidades profissionais dos membros da equipe), para alocar os membros da equipe nas atividades que requerem especificações profissionais adequadas em uma determinada atividade (**ver Quadro 04**).

Quadro 04: Matriz de Competência

Equipe	Competência
Elder G. Pereira	**Java, JSP, JSF, PHP, HTML, MySQL, JavaScript, Hibernate Annotation.**

Fonte: Autor da Pesquisa, 2011

4.4.2 PLANO DE RELEASES

As Releases são preenchidas a partir de uma subseção do artefato de Inicialização (User Story), um conjunto destas são alocadas dentro das Iterações, e estas dentro de uma Release, o YP recomenda nas Releases duas Iterações de duas semanas cada, isso levando em consideração uma disciplina de período letivo de um curso, mas que não precisa

necessariamente seguir essa recomendação quando adotado no ramo comercial, a seguir são apresentadas as Releases (**ver Quadro 05, Quadro 06, Quadro 07**).

Quadro 05: Plano de Release 01

Release 01: 01/03/11 – 20/03/11	**Gerente – Elder Gonçalves Pereira**	
Iteração	**User Story**	**Período**
Iteração 01	US01	01/03/11 – 15/03/11
Iteração 02	US02, US03	16/03/11 – 30/03/11

Fonte: Autor da Pesquisa, 2011

Quadro 06: Plano de Release 02

Release 02: 21/03/11 – 12/04/11	**Gerente – Elder Gonçalves Pereira**	
Iteração	**User Story**	**Período**
Iteração 03	US04, US05	01/04/11 – 15/04/11
Iteração 04	US06, US07	16/04/11 – 30/04/11

Fonte: Autor da Pesquisa, 2011

Quadro 07: Plano de Release 03

Release 03: 13/04/11 – 20/05/11	**Gerente – Elder Gonçalves Pereira**	
Iteração	**User Story**	**Período**
Iteração 05	US08, US09	01/05/11 – 15/05/11
Iteração 06	US10, US11	16/05/11 – 30/05/11

Fonte: Autor da Pesquisa, 2011

4.4.3 PLANO DE ITERAÇÃO

Cada User Story alocada em uma Iteração apresenta uma definição de problema a ser solucionada, são listadas em um plano de Iteração e divididas em um conjunto de atividades,

estas são direcionadas a cada desenvolvedor para solucionar o problema, é especificado uma Estimativa de Tempo para o termino da atividade, ao termino da atividade o desenvolvedor conclui o Tempo Real gasto, e o STATUS (se a atividade foi ou não concluída).

Quadro 08: Plano de Iteração 02

Iteração 02 - 16/03/11 – 30/03/11					
US02 - Implementar os modelos de classe referente as entidades do sistema, e a classe DAOGenerica responsável para operações CRUD do sistema.					
Testes de Aceitação					**STATUS**
TA2.1	Configuração da plataforma de desenvolvimento, com todos os plugins indispensáveis para a efetivação do sistema.				
TA2.2	Confirmar as classes necessárias na camada de modelo.				
TA2.3	Confirma classe responsável pelas operações CRUD.				
TA2.4	Verificação de todas as bibliotecas necessárias, para uso das tecnologias.				
TAA 01	**Descrição**	**Responsável**	**Estimativa de Tempo (Horas)**	**Tempo Real (Horas)**	**STATUS**
A2.1	Configuração das bibliotecas necessárias no Eclipse.	ELDER	1		
A2.2	Gerar as classes entidades com tecnologia Hibernate, indispensáveis para mapeamento com o banco.	ELDER	3		
A2.3	Criar classe geradora de tabela.	ELDER	2		
A2.4	Criação de DAO Genérico para operações CRUD, com tecnologia Hibernate.	ELDER	3		
A2.5	Criar classe teste para por em prova o DAO Genério.	ELDER	1		
US03 - Implementar modulo de venda.					
Teste de Aceitação					**STATUS**
TA3.1	Verificar se o cliente está sendo disponibilizado como opção.				
TA3.2	Verificar se funcionário estar sendo disponibilizado como opção.				
TA3.3	Verificar se o produto está sendo disponibilizado como opção.				
TA3.4	Realizar venda com todos os campos (operação com sucesso).				
TA3.5	Realizar venda sem todos os campos (operação não realizada).				
TA3.6	Testar o campo Unidade para que liste a quantidade de produto em estoque.				

TA3.7	Verificar se o produto selecionado está disponibilizando a quantidade certa em estoque, e realizando o cálculo correto.				
Atividade	**Descrição**	**Responsável**	**Estimativa de Tempo (Horas)**	**Tempo Real (Horas)**	**STATUS**
A3.1	Criação da interface do módulo de venda.	ELDER	1		
A3.2	Criar os "Beans" responsáveis pelo mapeamento das informações.	ELDER	1		
A3.3	Criar método para listar os clientes na interface.	ELDER	2		
A3.4	Criar método para listar os funcionários na interface.	ELDER	2		
A3.5	Criar método para listar os produtos na interface.	ELDER	2		
A3.6	Criar método para fazer o calculo do valor da venda, após selecionar o produto e unidades.	ELDER	3		
A3.7	Método para realizar a efetivação do cadastro, e o cancelamento do mesmo.	ELDER	2		

Fonte: Autor da Pesquisa, 2011

Percebe-se que nesse momento o Tempo Real gasto no desenvolvimento e STUTUS (se atividade foi ou não concluída), ainda permanecem sem preenchimento, isso devido não ter ocorrido implementação de código **(ver Quadro 08).**

4.5 IMPLEMENTAÇÃO

Depois da Iteração ter sido definida parte-se para o processo de implementação, neste momento a codificação é iniciada, lembrando que para tal o YP recomenda o uso de Integração Contínua, Boas Práticas de Codificação, Propriedade coletiva de Código (se existe mais de um membro na equipe).

Segue-se o Plano de Iteração 02, as atividades referentes à US02 geram a estrutura base do sistema (**ver Apêndice A**), depois as atividades referentes à US03 geram o modulo de venda do sistema (**ver Apêndice B).** Com base no código produzido, o mesmo é submetido a uma maratona de testes, primeiro Testes de Unidade (referentes à estrutura interna do código), depois Testes de Aceitado (definidos pelo cliente), e Testes de Usabilidade (**ver Apêndice C).**

4.5.1 FERRAMENTAS E TECNOLOGIAS

4.5.1.1 Java

Java é uma linguagem de programação orientada a objetos criada pela Sun Microsystems que atualmente foi incorporada pela Oracle Corporation. Sendo uma linguagem de programação de referencia no mundo, devido sua: versatilidade, eficiência, segurança, robusta e, o mais importante dos fatores, a portabilidade entre plataformas. Muito diferentemente das linguagens convencionais, que são compiladas para código nativo, enquanto a linguagem Java é compilada para um bytecode e executada por uma máquina virtual.

4.5.1.2 Apache Tomcat

Apache Tomcat é um servidor web desenvolvido pela Apache Software Foundation (ASF), essa tecnologia implementa cmo JavaServer Pages (JSP) e Java Servlet que atualmente são especificações da Oracle Corporation, tendo como finalidade fornecer um servidor web para aplicações Java.

4.5.1.3 JSF

JavaServer Faces (JSF) é uma tecnologia para desenvolvimento web baseada em linguagem de programação Java, sendo um padrão de desenvolvimento para fornecedores de ferramentas criarem produtos que valorizem a produtividade no desenvolvimento de interfaces visuais. JSF é baseado no padrão de projeto MVC (modelo, visão e controle), com separação bem definida entre as camadas e regras de negócio.

4.5.1.4 Framework RichFaces

RichFaces é uma biblioteca de componentes JSF open source (gratuita), que disponibiliza uma variedade de componentes visuais e, sendo também um framework capaz de tornar aplicações web capazes de trabalhar com AJAX (Asynchronous Java Script and XML) técnica essa usada para criar aplicações web mais interativas.

4.5.1.5 MySQL

O MySQL atualmente um dos SGBD (sistema de gerenciamento de banco de dados) mais populares na WEB, apresenta algumas características importantíssimas, como: portabilidade, compatibilidade com drivers externos (JDBC é uma API JAVA responsável pela comunicação entre MySQL e linguagem JAVA), suporte a controle transacionais, dentre outras funcionalidades.

4.5.1.6 Hirbernate

O Hibernate é um framework para o mapeamento objeto-relacional, esta tecnologia facilita o mapeamento dos atributos de uma Classe Java em tabelas numa base de dados, a partir de arquivos XML.

4.5.1.7 DAO

O DAO (data acess objecto) foi projetado para persistência de dados em aplicações que utilizem banco de dados relacional, no desenvolvimento do SysVenda foi implementado esse padrão, sendo o principal papel do DAO no software consultar, alterar, inserir e excluir dados, operações estas indispensáveis para manipulação dos registros.

4.5.1.8 MVC

Model, view and controller (modelo, visão e controle) é um padrão arquitetural para criação de software que tem como objetivo a separação das camadas de controle, apresentação e modelo. Por exemplo: um cliente faz uma requisição na camada de apresentação, que conseqüentemente faz a camada de controle direcionar essa requisição a camada de modelo, e depois o acontecimento inverso deste processo até apresentação dos dados ao cliente.

4.6 REUNIÃO DE ACOMPANHAMENTO

As reuniões segundo o YP devem acontecer uma vez a cada semana, neste momento se a finalização de uma iteração coincidir junto a reunião os módulos do sistema produzidos devem ser integrados, deve ocorrer também a coleta de métricas por parte do Gerente para atualização do Big Chart, a finalização do preenchimento da TAA (tabela de alocação de atividades) e a indicação de possíveis riscos no projeto se existir.

4.6.1 BIG CHART

A seguir o gerente faz a coleta de métricas para possibilitar uma maior compreensão no andamento do projeto, tais como: Classes Criadas, Teste de Aceitação realizados, Teste de Unidade, entre outras, (ver Quadro 09).

Quadro 09: Big Chat

Período de Iteração	CC	TA	TU	PJ	US	Observação
01/03/11 à 15/03/11	0	0	0	0	1	Sem implementação de código, devido estudo das tecnologias.
16/03/11 à 30/03/11	3	10	24	0	2	
Próxima Iteração Aqui						

Fonte: Autor da Pesquisa, 2011

CC => Classes Criadas; TA => Teste de Aceitação; TU=> Teste de Unidade
PJ => Páginas JSF; US => User Story

4.6.2 TAA (tabela de alocação de atividades)

Ao fazer a coleta de métricas (informações) o campo de STATUS referentes as atividades são preenchidas, nesta Iteração todas as atividades foram finalizadas como concluídas, mas em outros casos elas poderiam ter sido finalizadas como: em desenvolvimento (neste caso as atividades seriam alocadas para próxima iteração) ou abortadas. O Tempo Real do desenvolvimento das atividades é atualizado, e por último o STATUS do Teste de Aceitação (**ver Quadro 10**).

Quadro 10: Plano de Iteração 02

Iteração 02 - 16/03/11 – 30/03/11					
US02 - Implementar os modelos de classe referente as entidades do sistema, e a classe DAOGenerica responssável para operações CRUD do sistema.					
Testes de Aceitação					**STATUS**
TA2.1	Configuração da plataforma de desenvolvimento, com todos os plugins indispensáveis para a efetivação do sistema.				C
TA2.2	Confirmar as classes necessárias na camada de modelo.				C
TA2.3	Confirma classe responsável pelas operações CRUD.				C
TA2.4	Verificação de todas as bibliotecas necessárias, para uso das tecnologias.				C
TAA 01	**Descrição**	**Responsável**	**Estimativa de Tempo (Horas)**	**Tempo Real (Horas)**	**STATUS**
A2.1	Configuração das bibliotecas necessárias no Eclipse.	ELDER	1	1	C
A2.2	Gerar as classes entidades com tecnologia Hibernate, indispensáveis para mapeamento com o banco.	ELDER	3	4	C
A2.3	Criar classe geradora de tabela.	ELDER	2	2	C
A2.4	Criação de DAO Genérico para operações CRUD, com tecnologia Hibernate.	ELDER	3	4,5	C
A2.5	Criar classe teste para por em prova o DAO Genério.	ELDER	1	1	C
US03 - Implementação do modulo de venda.					
Testes de Aceitação					**STATUS**
TA3.1	Verificar se o cliente está sendo disponibilizado como opção.				C
TA3.2	Verificar se funcionário estar sendo disponibilizado como opção.				C
TA3.3	Verificar se o produto está sendo disponibilizado como opção.				C
TA3.4	Realizar venda com todos os campos (operação com sucesso).				C
TA3.5	Realizar venda sem todos os campos (operação não realizada).				C
TA3.6	Testar o campo Unidade para que liste a quantidade de produto em estoque.				C

TA3.7	Verificar se o produto selecionado está disponibilizando a quantidade certa em estoque, e realizando o cálculo correto.				C
Atividade	**Descrição**	**Responsável**	**Estimativa de Tempo (Horas)**	**Tempo Real (Horas)**	**STATUS**
A3.1	Criação da interface do módulo de venda.	ELDER	1	1	C
A3.2	Criar os "Beans" responsável pelo mapeamento das informações.	ELDER	1	1	C
A3.3	Criar método para listar os cliente na interface.	ELDER	2	1,5	C
A3.4	Criar método para listar os funcionários na interface.	ELDER	2	1,5	C
A3.5	Criar método para listar os produtos na interface.	ELDER	2	1,5	C
A3.6	Criar método para fazer o calculo do valor da venda ao selecionar o produto e unidades.	ELDER	3	4	C
A3.7	Implementar forma de efetivar o cadastro, e cancelar o cadastro.	ELDER	2	2	C

Fonte: Autor da Pesquisa, 2011

Status: C = concluído, D = desenvolvimento, A = abortado.

É indispensável à presença do cliente na reunião e demais membros, pois o primeiro é responsável pela aprovação das atividades finalizadas pelos desenvolvedores, e segundo pelas informações referentes sobre suas produções durante a semana.

4.6.3 ANALISE DE RISCO

Lembrando-se que riscos são empecilhos na efetivação do desenvolvimento do projeto, e que estes devem ser identificados, analisados e superados o mais rápido possível.

Quadro 11: Risco

Data	Risco	Prioridade	Responsável	Status	Providencia/Solução
16/03/11 à 30/03/11	Tecnologia desconhecida	Alta	Elder	Superado	Adquirir conhecimento básico

	RichFaces				no assunto.
16/03/11 à 30/03/11	Tecnologia desconhecida **Hibernate**	Alta	Elder	**Vigente**	Adquirir conhecimento mais abrangente.
16/03/11 à 30/03/11	Tecnologia desconhecida **JFreeChar**	Baixa	Elder	**Abortado**	Adquirir conhecimento básico no assunto, se o projeto requerer gráficos.

Fonte: Autor da Pesquisa, 2011

Prioridade: Alta, Média e Baixa.
Status: **Vigente**, **Superado** e **Abortado**.

Ao inicio do projeto foram previamente identificados alguns riscos, o primeiro foi superado, no segundo o desenvolvedor conseguiu abstrair conhecimento necessário para finalizar o projeto, e o terceiro foi considerado aquisição de conhecimento desnecessário, sendo então abortado. Mas que ainda pode ser levando em consideração dependendo da necessidade do cliente (**ver Quadro 11**).

A partir desse ponto os processos referentes às seções 4.4 à 4.6 são repetidas até a finalização de todas as Iterações alocadas nas Releases. Dependendo do encaminhamento do projeto se o mesmo necessitar de modificações outras sessões poderão ser alteradas, o projeto será dito finalizado depois da aprovação e satisfação do cliente.

5 CONSIDERAÇÕES FINAIS

Nesta seção, será apresentado o processo conclusivo desta pesquisa, identificando todos os passos do trabalho, análise, relacionamento e prova de finalização dos objetivos.

O TCC em questão propiciou uma visão geral da Engenharia de Software segundo alguns autores renomados no assunto, e também uma análise significativa dos processos de um modelo de desenvolvimento de software genericamente que podem ser direcionados tanto a modelos tradicionais quanto ágeis e, que essa significância se direciona a escolha da metodologia mais adequada ao problema.

Aqui, também foram mostradas algumas definições e princípios de XP, RUP e Agile Modeling, modelos estes usados como base para a especificação e definição do easYProcess, atribuindo a este às melhores práticas e valores dos modelos apoiadores, integrando à produção, rapidez com qualidade.

Quanto ao desenvolvimento do software anteriormente mencionado, por meio das especificações do YP, trouxe como experiência pessoal e profissional a definição de papéis, o diálogo com o cliente, e a cultura do planejamento, implementação e reuniões de acompanhamento. Sem contar, a elaboração de artefatos específicos.

Com a modelagem do sistema de software, pôde facilmente ser observado que YP gera muitos artefatos antes de implementar o software, isso devido ao fato de o modelo de desenvolvimento estar disponível em versão original, considerando artefatos essenciais para um bom andamento do projeto.

Portanto, uma boa atualização e refinamento se fazem necessários para que a metodologia de desenvolvimento easYProcess produza menos artefatos, e desta forma, possa também atuar mais e competir com outras metodologias de desenvolvimento ágeis no mercado atual, garantindo sucesso do binômio problema-solução.

5.1 TRABALHOS FUTUROS

Este trabalho serve como base para mostrar que o easYProcess apesar de ser uma metodologia de desenvolvimento ágil, deve passar por um processo de atualização, não pelo fato de ainda está disponível na versão original (2007), mas que também tenha finalidade de produzir menos artefatos, já que o mercado é bem competitivo e tempo é um bem precioso, e desta forma competir com outras metodologias de desenvolvimento ágeis atualmente no mercado.

Em relação ao sistema produzido (SysVenda), é previsto que o mesmo em trabalhos futuros seja submetido a um processo de atualização (upgrade) para possíveis melhoramentos de suas funcionalidades e incrementos de outras pendências, como a opção para visualização de gráficos onde os administradores do SysVenda poderão ter melhores análises das informações na tela do sistema.

REFERÊNCIAS

AVILA, Ana Luiza**. Introdução à Engenharia de Requisitos.** 2008**.** Disponível em:<http://www.devmedia.com. br/articles/ viewcomp.asp? comp=8034>. Acesso: 03/06/2011.

AURÉLIO, Holanda B. Ferreira. **Mini Aurélio – O minidicionário da língua portuguesa.** 5º Ed. Rio de Janeiro : FNDE, 2004.

AMBLER, Scott W. **Uma Introdução à Modelagem Ágil.** 2009. Disponível em: <http://translate.googleusercontent.com/translate_c?hl=ptBR&prev=/search%3Fq%3Dwww.agilemodeling.com%26hl%3DptR%26biw%3D836%26bih%3D478%26prmd%3Divns&rurl=translate.google.com.br&sl=en&twu=1&u=http://www.agilemodeling.com/essays/introductionToAM.htm&usg=ALkJrhhq6njJCXCJbNICGODJSqxp94z87Q>. Acesso: 22 de Abril de 2011

CAETANO, Rodrigo. **Metodologias de desenvolvimento: qual a mais adequada?**. Computerwordl, 2009. Disponível em: < http://computerworld.uol.com.br/gestao/2009/08/05/metodologias-de-desenvolvimento-qual-a-mais-adequada/>. Acesso: 01 de Julho de 2011

FREIRE, Alexandre. **Programação eXtrema Desenvolvendo Software com Qualidade e Agilidade**. 2003.

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.

GERVAZONI, Thiago Pastorello. XP – Extreme Programming. 2005.Disponível em: <http://www.linhadecodigo.com.br/artigo/764/XP-%E2%80%93-Extreme-Programming-%E2%80%93-Parte-1.aspx>. Acesso: 18 de Abril de 2011.

MOLINARI, Leonardo. **Gerência de Configuração - Técnicas e Práticas no Desenvolvimento do Software.** Florianópolis: Visual Books, 2007.

MONTEIRO, Guilherme **Alexandre** Reinaldo. **Extreme Programming (XP).** 2009. Disponivel em: <http://www.cin.ufpe.br/~gamr/FAFICA/ Desenvo lvimento % 20de%20sistemas/XP.pdf>. Acesso: 15 de Abril de 2011.

MARTINES, Marina. **RUP**. 2010. Disponível em: <http://www.infoescola.com/engenharia-de-software/rup/>. Acesso: 19 de Abril de 2011.

KUHN, Giovane Roslindo**. PAMPLONA,** Vitor Fernando. Apresentando XP. Encante seus clientes com Extreme Programming. 2009. Disponível em: <http:// javafree.uol.com.br/artigo/871447/#equipe>. Acesso: 18 de Abril de 2011.

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

MANSUR, Ricardo. **Governança de TI: Metodologias, Frameworks e Melhores Práticas.** Rio de Janeiro: BRASPORT, 2007, pg. 167. Disponível em: < http://www.livrariaresposta.com.br /v2/produto.php?id=30722&sp=0>. Acesso: 29 de Abril de 2011.

SILVA, F. A. M; LIMA, F. A. **Analisando e Utilizando o easYProcess**. 2010

SOMMERVILLE, Ian. **Engenharia de Software**. 8. ed. São Paulo: Addison Wesley, 2007.

TANAKA, Sergio Akio. BANKI, André. **ALINHAMENTO DO RUP 7.0 AOS VALORES DO MOVIMENTO ÁGIL**. 2008. Disponível em: < http://revista.ctai.senai.br/index.php/edicao01/article/download/19/17>. Acesso: 27 de Junho de 2011.

WILEY, John e Sons. **Design de Interação**. São Paulo: Bookman, 2002. P. 207 - 208.

APÊNDICES

Apêndice A: Estrutura Base do Sistema

Classe Entidade Venda

```
package br.systemVenda.entidade;

import java.io.Serializable;
import java.util.Date;
import javax.persistence.*;
import br.systemVenda.util.*;

@Entity
@Table(name="vendas")
public class Vendas implements Serializable{

        private static Vendas instance = new Vendas();
    public static Vendas getInstance(){
      return instance;
    }

        @Id
        @GeneratedValue(strategy=GenerationType.IDENTITY)
        private Integer codigo;

        @ManyToOne
        @JoinColumn(name="fk_cliente_id", nullable=false)
        private Cliente cliente;

        @ManyToOne
        @JoinColumn(name="fk_funcionario_id", nullable=false)
        private Funcionario funcionario;

        @ManyToOne
        @JoinColumn(name="fk_produto_id", nullable=false)
        private Produto produto;

        @Column(scale=10, precision=2, nullable=false)
        private float valorVendaCliente;

        @Column(scale=10, precision=2, nullable=false)
        private int unidadeProduto;

        @Temporal(TemporalType.DATE)
        private Date dataVenda;

        @Temporal(TemporalType.DATE)
        private Date dataConfirmaVenda;
```

```
        @Enumerated(EnumType.STRING)
        @Column(length=10)
        private TipoVenda tipo;

//temos gets e set no fim do codigo
} // fim do código acima
```

Classe para operações CRUD

```
package br.systemVenda.DAOeBusca;

import java.util.*;
import org.hibernate.*;
import org.hibernate.criterion.Restrictions;
import br.systemVenda.entidade.Vendas;
import br.systemVenda.util.*;

public class GenericoDAO<Tipo> implements InterfaceDAO<Tipo>{

        private Session session;
        private Transaction transaction;
        private Tipo tipo;
        private List<Tipo> lista;

        public GenericoDAO(Tipo tipo) {
                super();
                this.session = HibernateUtil.getSession();
                this.transaction = session.beginTransaction();
                this.tipo = tipo; }

        public void daoClose(){
                session.close(); }

        public boolean insert() {
                boolean condicao=true;
                try{
          session.save(tipo);
          transaction.commit();
      }catch(HibernateException hibernateException){
          if(transaction.isActive()){
          transaction.rollback();
          }
          hibernateException.printStackTrace();
          condicao=false;
      }

      return condicao;
          }

        public boolean update(){
                boolean condicao=true;
```

```
            try{
   session.update(tipo);
   transaction.commit();

}catch(HibernateException hibernateException){
   if(transaction.isActive()){
    transaction.rollback();
   }
   hibernateException.printStackTrace();
   condicao=false;
}
return condicao;
    }

    public boolean delete() {
            boolean condicao=true;
            try{
   session.delete(tipo);
   transaction.commit();

}catch(HibernateException hibernateException){
    if(transaction.isActive()){
    transaction.rollback();
   }
   hibernateException.printStackTrace();
   condicao=false;
}
return condicao;
    }

    public List<Tipo> getAll(){
            lista = new LinkedList<Tipo>();
            try{
   lista = session.createCriteria(tipo.getClass()).list();
}catch(HibernateException hibernateException){
                    lista =null;
            }
            return lista;
    }

    public Tipo getId(int id){
            Tipo type;
            try{
        type = (Tipo)session.get(tipo.getClass(), id);
       }catch(HibernateException erro){
            type = null;
       }
       return type;  }
```

Apêndice B: Modulo de venda

Classe vendasBean responsável pela ligação das camadas: controle e visão

```
package br.systemVenda.bean;

//import java.util.*;
import javax.faces.event.*;
import javax.faces.model.SelectItem;

import br.systemVenda.DAOeBusca.*;
import br.systemVenda.entidade.*;
import br.systemVenda.util.*;

public class vendasBean {
	private FactoryObject factory = new FactoryObject(); // fabrica

	private List<ProdutoUnidade> listaProdutos;
	private ProdutoUnidade deleteProduto;

	private List<SelectItem> clientes;
	private List<SelectItem> funcionarios;
	private List<SelectItem> produtos;
	private List<SelectItem> numeros;

	private Vendas selectedVenda;
	private Produto selectProduto;

	private Produto produtoTemporario;
	private float totalSoma=0.f;
	private int unidadeProduto;

	public vendasBean() {
		selectedVenda =  new Vendas();
		produtoTemporario = factory.getProduto();
	}

	public List<SelectItem> getClientes() {
		if(clientes == null) {
			clientes = new LinkedList<SelectItem>();

			Cliente cliente = factory.getCliente(); // usado para referenceia
			cliente.setCodigo(-1);
			cliente.setNome("--selecione--");

			clientes.add(new SelectItem(cliente, cliente.getNome()));

			for(Cliente c : new GenericoDAO<Cliente>(cliente).getAll()){
				clientes.add(new SelectItem(c, c.getNome()));
				//System.out.println("Nome: "+c.getNome());
			}

		}
```

```
            return clientes;
        }

        // gets e sets

}
```

Código JSF para Modulo Venda de Produto(s)

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="f"  uri="http://java.sun.com/jsf/core"%>
<%@ taglib prefix="h"  uri="http://java.sun.com/jsf/html"%>
<%@ taglib prefix="a4j" uri="http://richfaces.org/a4j"%>
<%@ taglib prefix="rich" uri="http://richfaces.org/rich"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Venda Produto</title>
<link rel="stylesheet" type="text/css" href="css/estilo.css" />
</head>
<body>
<f:view>
<h2><h:outputText value="Venda de Produto(s)"/></h2>
<br>
<rich:messages id="message" layout="table" errorClass="msgErro"
infoClass="msgInfo"
                                    showSummary="true" showDetail="true">
      <f:facet name="infoMarker">
            <h:graphicImage value="imagens/sucesso.gif"/>
      </f:facet>
      <f:facet name="errorMarker">
            <h:graphicImage value="imagens/erro.gif"/>
      </f:facet>
</rich:messages>
<br/>
<a4j:form id="frm_venda" >
<rich:panel id="rich_panel_selecionar" style="width: 400px;"
header="Selecione os Dados...">
      <h:panelGrid columns="3" >
            <!-- primeiro pedaço -->
            <h:panelGroup>
                  <h:outputLabel value="Cliente: "/><h:outputLabel
value="*" style="color: red;"/>
            </h:panelGroup>
            <h:selectOneMenu id="id_cliente"
value="#{vendasBean.selectedVenda.cliente}">
                  <a4j:support event="onchange"/>
                  <f:converter converterId="ClienteConverter"/>
                  <f:selectItems value="#{vendasBean.clientes}"/>

            </h:selectOneMenu>
            <h:message for="id_cliente" showDetail="true"
showSummary="false" styleClass="msgErro"/>
```

```
            <!-- segundo pedaço -->
            <h:panelGroup>
                  <h:outputLabel value="Funcionário: "/><h:outputLabel
value="*" style="color: red;"/>
            </h:panelGroup>
            <h:selectOneMenu id="id_funcionario"
value="#{vendasBean.selectedVenda.funcionario}">
                  <a4j:support event="onchange"/>
                  <f:converter converterId="FuncionarioConverter"/>
                  <f:selectItems value="#{vendasBean.funcionarios}"/>
            </h:selectOneMenu>
            <h:message for="id_funcionario" showDetail="true"
showSummary="false" styleClass="msgErro"/>

            <!-- terceiro pedaço -->
            <h:panelGroup>
                  <h:outputLabel value="Data Venda: "/><h:outputLabel
value="*" style="color: red;"/>
            </h:panelGroup>
            <rich:calendar id="id_data_venda"
value="#{vendasBean.selectedVenda.dataVenda}" datePattern="dd/MM/yyyy"
inputSize="12"/>
            <h:message for="data_venda" showDetail="true"
showSummary="false" styleClass="msgErro"/>

            <!-- quarta pedaço -->
            <h:panelGroup>
                  <h:outputLabel value="Produto: "/><h:outputLabel
value="*" style="color: red;"/>
            </h:panelGroup>
            <h:panelGroup>
            <h:selectOneMenu id="id_produto"
value="#{vendasBean.selectProduto}"

valueChangeListener="#{vendasBean.getValueProdutoChange}"
onchange="submit();">
                  <f:converter converterId="ProdutoConverter"/>
                  <f:selectItems value="#{vendasBean.produtos}" />
            </h:selectOneMenu>
            <h:outputText value=" - "/>

            <h:selectOneMenu id="id_unidade"
value="#{vendasBean.unidadeProduto}"

valueChangeListener="#{vendasBean.getValorUnidadeChange}"
onchange="submit();">
                  <f:converter converterId="NumeroConverter" />
                  <f:selectItems value="#{vendasBean.numeros}"/>
            </h:selectOneMenu>
            </h:panelGroup>
            <h:message for="id_produto" showDetail="true"
showSummary="false" styleClass="msgErro"/>

      </h:panelGrid>
</rich:panel>

<rich:panel id="rich_panel_informacao" style="width: 400px;"
header="Produto(s) e subtotal...">
      <rich:dataTable id="tabela" border="1" var="item"
value="#{vendasBean.listaProdutos}"
```

```
                              rows="5" rowClasses="linha1Tabela,
linha2Tabela">
            <rich:column>
                  <f:facet name="header">
                        <h:outputText value="Produto(s)"/>
                  </f:facet>
                        <h:outputText value="#{item.produto.descricao}"/>
            </rich:column>
            <rich:column>
                  <f:facet name="header">
                        <h:outputText value="Valor/Produto"/
```

Interface do Modulo Venda de Produto(s)

Figura 10: Interface de Venda

Fonte: Autor da Pesquisa, 2011

Apêndice C: Teste de Usabilidade

Atividades referentes ao teste de usabilidade

Atividade 01- Realização de Venda de Produto.

Roteiro: Nesta atividade o usuário terá que realizar uma seqüência de vendas, usando a interface de vendas de produtos

Instruções:

a) Abra a interface Venda de Produto(s), e verifique se todos os campos necessários para venda estão disponibilizados;

b) Tente confirmar um venda com pelo menos um campo de informação em branco, o resultado deverá ser uma venda não confirmada;

c) Liste vários produtos na tabela e verifique se o valor total da venda estar resultando corretamente;

d) Depois de selecionado um produto relacione uma quantidade para o produto;

e) Cancele um produto da tabela, e verifique se o valor total de venda está sendo decrementado corretamente;

f) Faça a venda de pelo menos um produto, não esquecer de preencher todos os campos da venda, a venda deve ser realizada com sucesso;

g) Faça uma nova venda com no mínimo 3 produtos listados na tabela, não esquecer de preencher todos os campos da venda, realizada com sucesso.