UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ – UTFPR
DIRETORIA DE PESQUISA E PÓS-GRADUAÇÃO
ESPECIALIZAÇÃO EM ENGENHARIA DE *SOFTWARE*

LUCAS GUILHERME DIEDRICH

INTEGRAÇÃO DA METODOLOGIA ÁGIL *OPENUP* NOS PROCESSOS DE ENGENHARIA DE *SOFTWARE*

MONOGRAFIA DE ESPECIALIZAÇÃO

MEDIANEIRA
2011

LUCAS GUILHERME DIEDRICH

INTEGRAÇÃO DA METODOLOGIA ÁGIL *OPENUP* NOS PROCESSOS DE ENGENHARIA DE *SOFTWARE*

Monografia apresentada como requisito parcial à obtenção do título de Especialista na Pós Graduação em Engenharia de *Software*, da Universidade Tecnológica Federal do Paraná – UTFPR – Câmpus Medianeira.

Orientador: Juliano Rodrigo Lamb, M.Eng

MEDIANEIRA

2011

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Diretoria de Pesquisa e Pós-Graduação
Especialização em Engenharia de *Software*

TERMO DE APROVAÇÃO

INTEGRAÇÃO DE METODOLOGIA ÁGIL *OPENUP* NOS PROCESSOS DE ENGENHARIA DE *SOFTWARE*

Por

Lucas Guilherme Diedrich

Esta monografia foi apresentada às 08:10 h do dia 10 de dezembro de 2011 como requisito parcial para a obtenção do título de Especialista no curso de Especialização em Engenharia de *Software*, da Universidade Tecnológica Federal do Paraná, *Câmpus* Medianeira. Os acadêmicos foram arguidos pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

_____
Prof. M.Eng. Juliano Rodrigo Lamb
UTFPR – Câmpus Medianeira
(orientador)

_____
Prof. Dr. Hermes Del Monego
UTFPR – Câmpus Medianeira

_____
Prof. Me. Pedro Luiz de Paula
UTFPR – Câmpus Medianeira

A folha de aprovação assinada encontra-se na Coordenação de Curso.

AGRADECIMENTOS

À Deus pelo dom da vida, pela fé e perseverança para vencer os obstáculos.

Aos meus pais, pela orientação, dedicação e incentivo nessa fase do curso de pós-graduação e durante toda minha vida.

O meu orientador professor Juliano Rodrigo Lamb, que me orientou, pela sua disponibilidade, interesse e receptividade com que me recebeu e pela prestabilidade com que me ajudou.

Agradeço aos pesquisadores e professores do curso de Especialização em Engenharia de *Software* – Administração e Desenvolvimento, professores da UTFPR, *Câmpus* Medianeira.

Enfim, sou grato a todos que contribuíram de forma direta ou indireta para realização desta monografia.

“Mesmo quando eu estou errado, eu tenho o meu ponto de vista”.

(Notorious BIG)

RESUMO

DIEDRICH, Lucas G. Integração da metodologia ágil *Openup* nos processos de engenharia de *software*. 2011. 80 p. Monografia (Especialização em Engenharia de *Software*). Universidade Tecnológica Federal do Paraná, Medianeira, 2011.

Este trabalho apresenta um estudo sobre a metodologia ágil *Openup*, demonstrando seus processos e funcionalidades focadas na alta produtividade do desenvolvimento de *software*, realizando uma apresentação das principais práticas e sua arquitetura de processos focado em processos ágeis, demonstrando através de um estudo de caso fictício aplicação da metodologia.

Palavras-chave: Integração contínua, Ágil, Alta Qualidade, Metodologia enxuta.

ABSTRACT

DIEDRICH, Lucas G. Integration of agile methodology *Openup* in the processes of *software* engineering. 2011. 80 p. Monografia (Especialização em Engenharia de *Software*). Universidade Tecnológica Federal do Paraná, Medianeira, 2011.

This paper presents a study on the Agile *Openup*, demonstrating their processes and features focused on high productivity of *software* development, making a presentation of significant architecture and its processes focused on agile processes, demonstrating through a fictional case study application the methodology.

Keywords: Continuous Integration, Agile, High Quality, Lean Methodology.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE SIGLAS

EPF	Eclipse Process *Framework*
IBM	International Business Machines
Openup	Open Unified Process
PDV	Ponto de Venda
RUP	Rational Unified Process
TDD	Test-Driven Design
UML	Unified Model Language
XP	Extreme Programming

SUMÁRIO

1 INTRODUÇÃO

Com a crescente evolução das tecnologias e das necessidades individuais na área que envolve a engenharia de *software* inúmeras maneiras de gerenciar os conteúdos gerados a partir do desenvolvimento de um *software* foram criadas.

Devido a esta evolução e a alta concorrência no mercado foi elaborado um manifesto que visa focar o desenvolvimento de um *software* totalmente ao que a *stakeholder*[1] precisa e não ao que está em um contrato, á esse manifesto foi dado o nome de Manifesto Ágil de *Software*. (MANIFESTO ÁGIL, 2001)

Este manifesto serviu de base para criar novas metodologias que ao seguir os fundamentos do manifesto ágil visam desenvolver melhores *software*s para os clientes da maneira mais fácil e prática, porém ainda se fazia necessário adaptar este novo pensamento a um processo de desenvolvimento unificado e conhecido como o RUP (*Rational Unified Process*).

De acordo com WTHREEX (2002), o RUP é um processo de engenharia de *software*. Ele oferece uma abordagem baseada em disciplinas para atribuir tarefas e responsabilidades dentro de uma organização de desenvolvimento. Sua meta é garantir a produção de *software* de alta qualidade que atenda às necessidades dos usuários dentro de um cronograma e de um orçamento previsíveis.

Baseado em metodologias ágeis disponíveis e a fundação Eclipse desenvolveu um novo *framework* de gerenciamento aberto e com a ajuda da comunidade, o EPF (*Eclipse Process Framework*). De acordo com EPF (2011), o *framework* visa produzir processos de engenharia de *software* customizáveis, com conteúdos de exemplos e ferramentas, e suporte com uma grande variedade de tipos de projetos e estilos de desenvolvimento.

Com a base bem estabelecida do EPF foi criado uma nova metodologia, o *Openup* (*Open Unified Process*). O objetivo central com o *Openup* foi criar uma abordagem ágil para o uso do RUP, e ao mesmo tempo aproveitar todas as boas praticas de outros processos ágeis. (KROLL, 2007)

O objetivo deste trabalho é descrever e demonstrar as principais técnicas e artefatos gerados pelo *Openup* utilizando como base um estudo de caso fictício.

---

[1] *Stakeholder* são as pessoas interessados pelo projeto, pode-se dizer que são os clientes que estão arcando com os custos do projeto e aqueles que utilizaram o projeto.

Além disso, será demonstrado diferenças que fazem o *Openup* melhor em relação a metodologia de desenvolvimento em cascata.

1.1 OBJETIVO GERAL

Descrever as práticas, artefatos e conhecimentos empregados pela metodologia ágil *Openup* aplicados a recursos da engenharia de *software* utilizando como demonstração um estudo de caso fictício.

1.2 OBJETIVOS ESPECÍFICOS

Os objetivos específicos são:

- Estudar, descrever e documentar as práticas e os conhecimentos empregados pelo *Openup* no gerenciamento de um projeto de *software*;
- Identificar os principais artefatos provenientes do *Openup*;
- Elaborar um estudo de caso fictício para realizar uma demonstração da criação dos artefatos e dos papéis aplicados na metodologia; e
- Analisar e apontar as principais vantagens sobre a metodologia de desenvolvimento em cascata na adoção do *Openup* para o gerenciamento de projetos.

1.3 JUSTIFICATIVA

Segundo FOWLER (2003), a prática do desenvolvimento de *software* é uma atividade caótica em sua maior parte, normalmente caracterizada pela expressão "codificar e consertar". O *software* é escrito sem um plano definido e o projeto do sistema é repleto de várias decisões de curto prazo.

Ainda de acordo com FOWLER (2003), isso funciona muito bem se o sistema é pequeno - mas à medida que o sistema cresce, torna-se cada vez mais difícil adicionar novos recursos a ele. Um sinal típico de um sistema desse tipo é uma longa fase de testes depois que o sistema está "pronto". Esta longa fase de testes entra em confronto direto com o cronograma, pois testes e depuração são atividades cujos tempos são impossíveis de serem estimados. (FOWLER, 2003)

Além disso, projetos emergentes necessitam certos fatores de mudança formais e ágeis as quais não podem ser supridas utilizando metodologias de desenvolvimento antigas. Os funcionamentos do ciclo de vida útil dos novos projetos estão ficando dinâmicos e esta certa dinamicidade deve ser bem gerenciada e de preferência com agilidade extraordinária.

Desde criação do RUP as empresas vêm adotando a metodologia como um padrão para o seu ciclo de desenvolvimento de projeto, porém, a arquitetura do RUP é extensa e seu tamanho a torna difícil de ser implantada. Já em contrapartida as metodologias ágeis focam em um processo enxuto de fácil compreensão e implantação, porém, elas não possuem a robustez necessária para ser aplicada em grandes projetos.

O *Openup* é uma metodologia que possui um conjunto de atividades e resultados que tem por objetivo auxiliar no processo da engenharia de um *software* com alta qualidade, focando na diminuição de riscos e que atenda a todos requisitos do *stakeholder*. De acordo com GUSTAFSSON (2008), a metodologia comporta guias pequenos e concisos em poucas páginas, as quais estão a apenas alguns cliques e abertas para serem acessadas de qualquer lugar.

Baseado neste modelo e na tendência do mercado em gerenciar projetos de *software* de uma maneira cada vez mais fácil, ágil e limpa esta monografia sustenta-se em discorrer e demonstrar a utilização das principais técnicas disponibilizadas pela metodologia que auxiliam na confecção de *software*.

1.4 ESTRUTURA DO TRABALHO

O trabalho foi subdividido em cinco capítulos, sendo que este primeiro apresenta a introdução e os objetivos do trabalho.

No capítulo dois é abordada os principais conceitos ao redor da engenharia de *software* a qual são empregados diretamente ou indiretamente pela metodologia *Openup*.

O capítulo três detalha a estrutura e funcionamento da metodologia do *Openup*, falando todos os itens e suas respectivas funções durante o processo de gerência do desenvolvimento de um *software*.

Aos capítulos quatro e cinco são destinados respectivamente à implementação de um projeto de *software* utilizando a metodologia ágil empregado neste trabalho e a conclusão final sobre o projeto implementado.

No último capítulo são descritas as considerações finais do trabalho e as possibilidades de trabalhos futuros ou continuações deste mesmo estudo.

2 ENGENHARIA DE *SOFTWARE*

De acordo com BROOKS (1987), a complexidade do *software* é uma propriedade essencial, não acidental. Assim, as descrições de uma entidade de *software* que abstrair a sua complexidade, muitas vezes abstraem sua essência.

Seguindo este pensamento a engenharia de *software* é uma entidade que visa abstrair a complexidade de um *software* em passos metodológicos que diretamente controlam a essência e a qualidade do mesmo.

De acordo com PRESSMAN (2006) a engenharia de *software* é o estabelecimento e uso de sólidos princípios de engenharia para que se possa obter economicamente um *software* que seja confiável e que funcione eficientemente em máquinas reais.

A ciência por trás dos mais diversos ensinos praticados pela engenharia de *software* vem das necessidades ocorridas e supridas através da utilização de um processo de *software*. Segundo PRESSMAN (2006), não existe uma abordagem em particular que seja a melhor para a solução da aflição de um *software*.

Desta maneira os conhecimentos obtidos nem sempre são aplicados a todos os projetos de *software*s, pois cada projeto possui um conceito de existência diferente dentre outros, porém, as metodologias aplicadas na engenharia de *software* visam de uma maneira genérica aplicar as melhores ações a serem realizadas no desenvolvimento e gerenciamento de um *software* sem que estas impactem no ecossistema do mesmo.

A engenharia de *software* como as outras áreas de engenharia envolve inúmeras subáreas e conhecimentos, a seguir estão brevemente descritas as principais subáreas e conhecimentos que são empregados atualmente ou que já foram aplicados durante o processo de estruturação de um *software*.

2.1 PROCESSOS E REQUISITOS DE *SOFTWARE*

Um processo é um conjunto de ações interligadas que visam atingir uma meta em comum para tentar resolver de maneira eficiente uma necessidade. Um processo na engenharia de *software* é o mesmo conjunto de ações disponibilizadas através de

uma metodologia sendo responsável em atingir a meta podendo ser funcionalidade ou um *software* completo.

De acordo FUGGETTA (2000), o desenvolvimento de *software* é um processo coletivo entre os membros da equipe, complexo em níveis de regras de negócios e criativo na solução dos problemas. Sendo assim, a qualidade de um produto de *software* depende fortemente das pessoas, da organização e de procedimentos utilizados para criá-lo.

Desta maneira um processo está ligado diretamente com tudo e todos que estão relacionados com o atual projeto separando em tarefas a serem realizadas e dizendo o que cada um deve realizar e como realizar. Segundo PRESSMAN (2006), o alicerce da engenharia de *software* é a camada do processo, formando uma base para o controle gerencial de projetos de *software* que estabelecem ao contexto como os métodos são aplicados, os produtos de trabalho[2] são produzidos, os marcos são estabelecidos, a qualidade é assegurada e a modificações são adequadamente geridas.

Um processo é definido por um conjunto de atividades que visam realizar a entrega de um artefato com qualidade, artefato este que está ligado diretamente com os requisitos que um *software* deve ter para que o *software* consiga suprir as necessidades de quem o requisitou.

O conjunto dos requisitos[3] é o que faz um projeto de *software* ter sentido em sua existência, cada requisito é uma necessidade levantada pelos interessados na utilização do sistema que deve ser suprida da melhor maneira possível.

Durante este processo de engenharia de requisitos é que deve ser decidido o que deve ser realizado. De acordo com BROOKS (1987), a parte mais árdua na construção de um sistema de *software* é decidir o que construir. Nenhuma outra parte do trabalho compromete mais o sistema se for feito de forma imprópria.

Estes requisitos podem ser separados em duas categorias, são elas:

- Requisitos funcionais: São as funcionalidades que são esperadas do sistema, ou seja, funcionalidades que são diretamente ligadas ao sistema e que algum usuário utilize-a diretamente. Por exemplo: O sistema deverá cadastrar produtos.

[2] Um produto de trabalho é todo item desenvolvido durante o ciclo de vida de um projeto, podem ser documentos ou até os requisitos.
[3] Um requisito é uma descrição do serviço a ser fornecido pelo sistema. (SOMMERVILLE, 2006)

- Requisitos não funcionais: São as funcionalidades que não estão ligadas diretamente com a necessidade de algum usuário, ou seja, não são as funções que um usuário utiliza diretamente e sim que o próprio sistema utiliza. Por Exemplo: O sistema deverá ter servidores de redundância.
- Requisitos suplementares: São as especificações que estão ligadas diretamente com o desempenho, usabilidade e confiabilidade do sistema. Um requisito suplementar também pode ser restrições aplicáveis em nível de *design* ou a regras de negócios.

Todos os requisitos de um *software* são definidos no começo de um projeto na forma de um documento que deve estar de acordo com todas as pessoas relacionadas com o mesmo, entretanto isto não significa que um requisito não possa ser alterado e é neste ponto que alguns modelos de processos divergem visando abstrair da melhor maneira as alterações no ambiente.

2.2 MODELOS DE DESENVOLVIMENTO

A maneira como o *software* é desenvolvido advém de uma metodologia aplicada para gerenciar os recursos necessários para o desenvolvimento do mesmo, a metodologia aplica conhecimentos empregados em projetos anteriores em uma estrutura analítica possibilitando que outros projetos sejam gerenciados corretamente.

Segundo CAETANO (2009), as metodologias de desenvolvimento de *software* servem para não tornar a tarefa, complexa por natureza, um verdadeiro caos. Ainda segundo CAETANO (2009), o problema é que, dependendo do projeto, os métodos tradicionais podem deixar os desenvolvedores amarrados a requisitos desatualizados, que não correspondem às reais necessidades dos usuários do sistema.

Por este motivo é que uma metodologia de desenvolvimento não pode ser utilizada em todos os tipos de desenvolvimento de *software* e é um fator muito importante de ser escolhido antes do início do projeto. Por exemplo, as metodologias ágeis como o *Scrum* geram poucos documentos, deixando a gestão do conhecimento do problema apenas nas pessoas responsáveis pelo projeto, assim

que esta pessoa responsável trocar de empresa o conhecimento não estará mais na empresa atual.

2.2.1 Modelo cascata

O modelo em cascata é considerado um dos mais antigos no ambiente de desenvolvimento de *software*, ele consiste em várias atividades executadas de uma forma sistemática e sequencial (Figura 1).

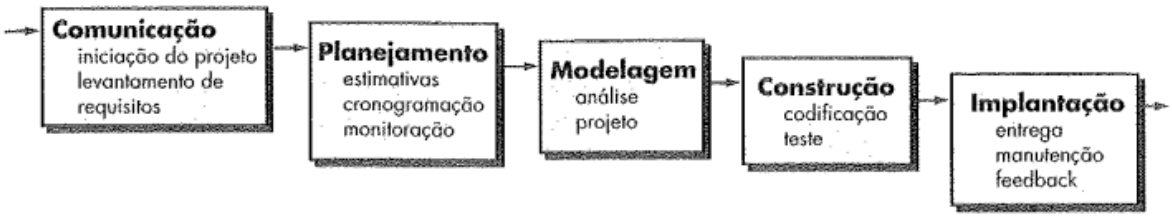


Figura 1 - Tarefas modelo cascata
Fonte: (PRESSMAN, 2006)

Este modelo prevê que a equipe de desenvolvimento possua conhecimento concreto e abrangente em relação à solução do problema, pois ele pressupõe que os requisitos estarão sempre em um ambiente estável o que acaba causando o atraso na redução de riscos.

Seu ciclo sequencial foca em uma abordagem linear que a cada estágio do ciclo é realizada a entrega de um artefato de análise, desta maneira, ao ser encontrado um erro em um estágio posterior é necessário fazer uma reavaliação em todos os estágios anteriores ao atual.

É possível considerar o erro como algo que o próprio analista não conseguiu verificar, porém algo que o cliente não sabia sobre o seu próprio negócio e não soube repassar a informação ao analista, sendo assim, é quase impraticável realizar a refatoração do código fonte na entrega do projeto.

2.2.2 Modelo iterativo incremental

O modelo iterativo incremental é um modelo que se baseia no modelo em cascata, porém utiliza uma separação dos estágios em iterações (Figura 2), ou seja, a cada iteração realizada é passado por todas as etapas do desenvolvimento o que acaba facilitando em detectar erros e corrigi-los rapidamente. De acordo com PRESSMAN (2006), o desenvolvimento incremental é particularmente útil quando não há mão de obra disponível para uma implementação completa, dentro do prazo comercial da entrega do produto.

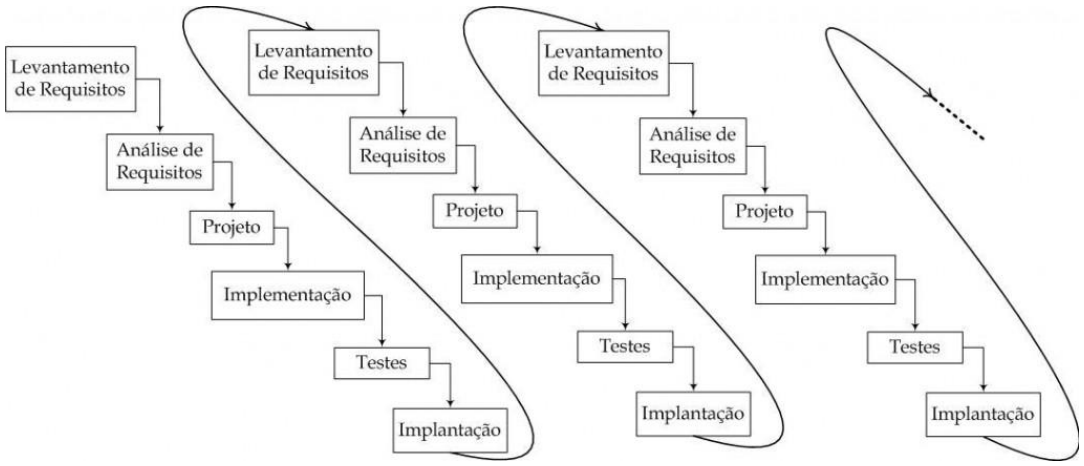


Figura 2 - Ciclo de vida Iterativo e Incremental
Fonte: (DONATO, 2010)

Outro benefício importante do modelo incremental é que se torna possível realizar uma medição mais apurada sobre o estado atual do projeto uma vez que a cada iteração é possível dizer que o projeto evoluiu em um número de porcentagem, e também testes e integração são realizados desde a primeira iteração reduzindo riscos e permitindo um maior *feedback*[4] por parte do cliente e usuário.

[4] O retorno de informação por parte dos interessados(clientes) em relação a um conjunto de informações de uma determinada função.

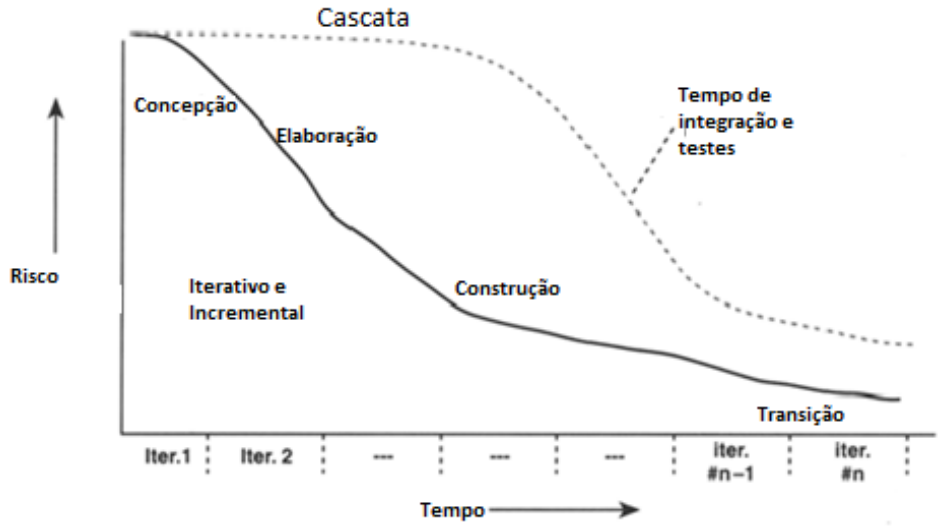


Figura 3 - Comparação entre Cascata e Iterativo Incremental
Fonte: (SOARES, 2010)

Na Figura 3 é possível visualizar a diferença entre os processos em cascata e iterativo incremental, nota-se uma grande diferença no que condiz com a fase de Implantação onde no modelo cascata demora-se mais tempo a ser atingido.

2.2.3 Modelos ágeis

Os modelos ágeis vieram de uma abordagem onde as metodologias atuais como o modelo em cascata ou iterativo incremental não estavam mais suprindo as necessidades ou eram muito burocráticas.

Segundo PRESSMAN (2006), as filosofias dos modelos ágeis encorajam a satisfação do *stakeholder* e a entrega incremental do *software* logo de início; equipes de projeto pequenas e altamente motivadas; métodos informais; produtos de trabalho de engenharia de *software* mínimo e simplicidade global do desenvolvimento.

Ao se utilizar um modelo ágil é capaz de visualizar que o modelo abrange somente o básico, a busca da simplicidade da engenharia de *software* de um projeto simples e até mesmo para projetos com um alto grau de alteração. Estes modelos pregam a maneira simples, entretanto deixam livre para que o arquiteto da equipe gerencie da melhor maneira possível o projeto.

Um dos modelos ágeis é o *SCRUM*, de acordo com SCHWABER e SUTHERLAND (2011), é um *framework* dentro do qual as pessoas podem resolver problemas adaptativos complexos, enquanto, produtivamente e criativamente entregam produtos com o mais alto valor possível.

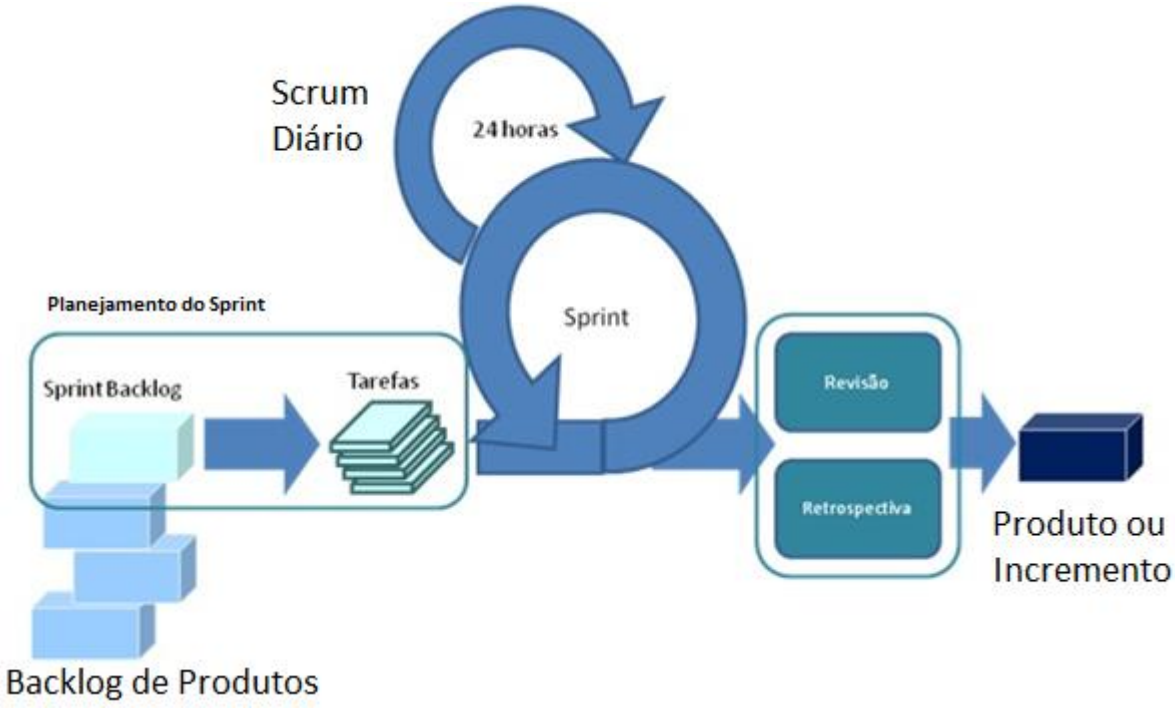


Figura 4 - Ciclo de vida *Scrum*
Fonte: (RODRIGUES, 2011)

As equipes possuem uma cultura auto organizável e funcional, escolhem a melhor forma de realizar seu trabalho; ao contrário de serem dirigidas por outros de fora da equipe; têm todas as competências necessárias para realizar o trabalho sem a dependência de outras que não fazem parte da equipe e é projetada para aperfeiçoar a flexibilidade, criatividade e produtividade. (SCHWABER e SUTHERLAND, 2011)

O ciclo de vida de um projeto utilizando *Scrum* (Figura 4) é focado em antecipar a existência de ocorrências, por isso ele é focado em padrões preestabelecidos e que funcionaram corretamente na empresa, por isso é necessário que a empresa possua uma grande cultura sobre o *software* que está desenvolvendo e que saiba introduzir os conhecimentos aprendidos a todos da equipe.

2.3 GERENCIAMENTO E EVOLUÇÃO DE *SOFTWARE*

Um projeto sem um gerenciamento correto ou até mesmo sem gerenciamento qualquer está predestinado a fracassar, o gerenciamento torna-se então uma parte indispensável na criação de um novo projeto e é através dele que é possível visualizar e levantar os principais pontos de um projeto é:

- Estimativas de custos;
- Estimativas de casos de uso;
- Estimavas de riscos;
- Fundos, componentes e artefatos;
- Monitoramento de desempenho;

Durante a execução do mesmo serão levantados inúmeros artefatos que facilitam o gerenciamento do *software*, e alguns são necessários ao ponto de vista ao realizar um acordo com o *stakeholder*.

O objetivo do gerenciamento além de controlar a evolução do *software* é melhorar os processos de desenvolvimento de *software* e até mesmo da objetividade e qualidade da equipe envolvida. Segundo ROYCE (2000), um melhor gerenciamento é mais importante que uma melhor tecnologia, a melhor tecnologia não compensará um gerenciamento fraco, e um bom gerente pode atingir bons resultados com poucos recursos.

As metodologias ágeis impactaram diretamente em como o *software* é gerenciado atualmente, como o foco da metodologia ágil é conseguir produzir um *software* de qualidade em um tempo mais justo foi necessário adaptar inúmeros dos artefatos gerados por metodologias antigas e em alguns casos nem são mais gerados. Os artefatos devem somente dizer o que realmente importa no desenvolvimento do *software* e não mais criar inúmeras documentações que nunca serão utilizadas pelos desenvolvedores ou até mesmo pelo *stakeholder*.

Em questões de estágios e artefatos gerados conforme ROYCE (2000) demonstra, sempre mantém um número constante de artefatos criados ou alterados em qualquer estágio do ciclo de vida do projeto. (Figura 5)

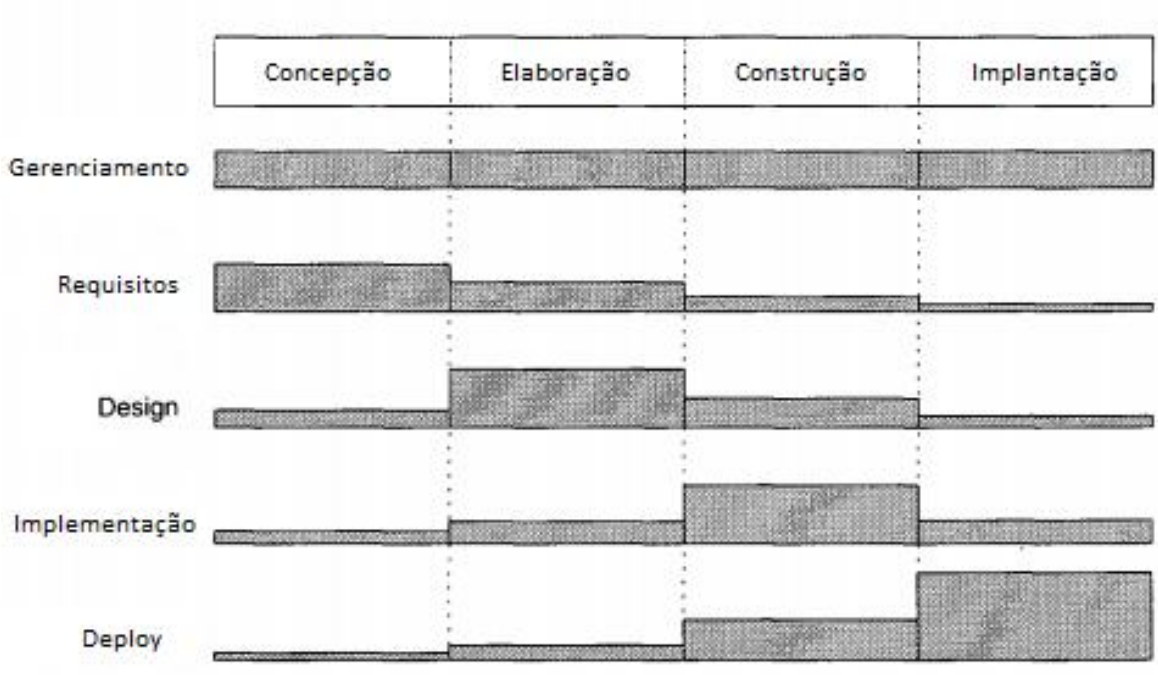


Figura 5 - Artefatos no ciclo de vida do projeto
Fonte: (ROYCE, 2000)

2.4 QUALIDADE DE *SOFTWARE*

É inevitável dizer que todo produto desenvolvido deve sempre ter alta qualidade a fim de trazer um *software* funcional para os usuários, segundo SOMMERVILLE (2006) é necessário reconhecer os problemas com a especificação do possível *software* e através deste criar procedimentos que realizam os testes de qualidade.

Segundo PRESSMAN (2006), a qualidade de *software* é uma somatória na qualidade de vários fatores e não é possível apenas aumentar a qualidade do *software* sem o controle e o gerenciamento de custo dos mesmos. A garantia de qualidade se baseia em um conjunto de funções que servem como padrão para identificar e avaliar pontos específicos do *software*.

O gerenciamento de custos de qualidade é o valor fornecido para atingir a qualidade focando em um nível de qualidade, por isso para garantir uma alta qualidade pode-se atingir um alto custo o que torna a verificação inviável.

É importante que a verificação da qualidade seja realizada constantemente no *software* desenvolvido, se possível a cada ciclo do desenvolvimento, pois a

identificação tardia de um erro também eleva o custo de ser concertado. Na Figura 06 é possível visualizar um gráfico que demonstra em relação à descoberta de uma falha em certo nível de desenvolvimento do projeto, durante o levantamento de requisitos este custo é de 1 vez, e quando o projeto já está em processo de implantação é de 40 a 1000 vezes o custo de corrigir o erro.

A documentação de qualidade é particularmente importante para equipes sistemas grandes e complexos, ajuda as pessoas visualizar que tarefas importantes não foram esquecidas e que partes da equipe não tirem conclusões errôneas sobre outros membros das equipes (SOMMERVILLE, 2006).

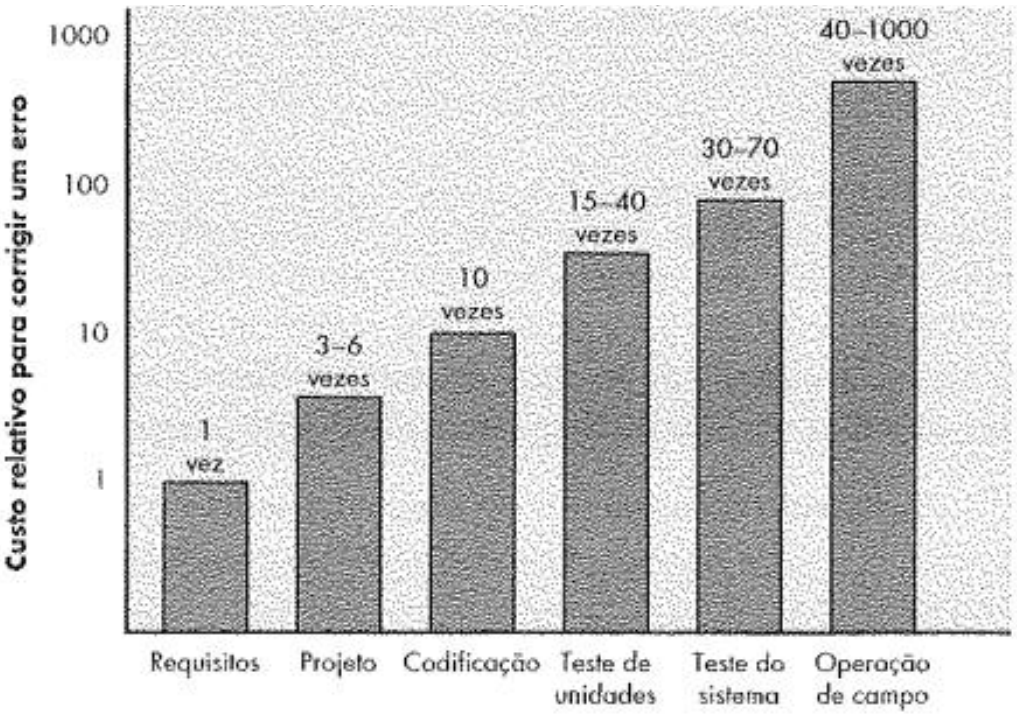


Figura 6 - Controle de custo e qualidade.
Fonte: (PRESSMAN, 2006)

Porém, a qualidade de um *software* não se baseia somente na qualidade do código fonte ou nos testes, a qualidade deve ser sentida pela equipe que está desenvolvendo o *software* e deve fazer parte da cultura interna da equipe.

2.5 PROCESSO UNIFICADO

O processo unificado ou RUP é uma metodologia de desenvolvimento de projetos de *software*, criado pela empresa *Rational Software Corporation* a qual posteriormente foi comprado pela IBM (*International Business Machines*), cujos objetivos são um desenvolvimento iterativo e incremental, baseado em uma arquitetura bem definida de projetos, com o ciclo de desenvolvimento feito por etapas (ou iterações), orientado a objetos, com tarefas e responsabilidades bem definidas, dirigido por casos de uso e com áreas de apoio bem definidas, como gerência de projeto.

A diferença do RUP para outras metodologias é que ele é um *framework* completo de processo de *software*, o *framework* gera um número fixo de artefatos e passos para qualquer tipo de *software*, na é possível visualizar os artefatos gerados em cada estágio do processo do *software*.

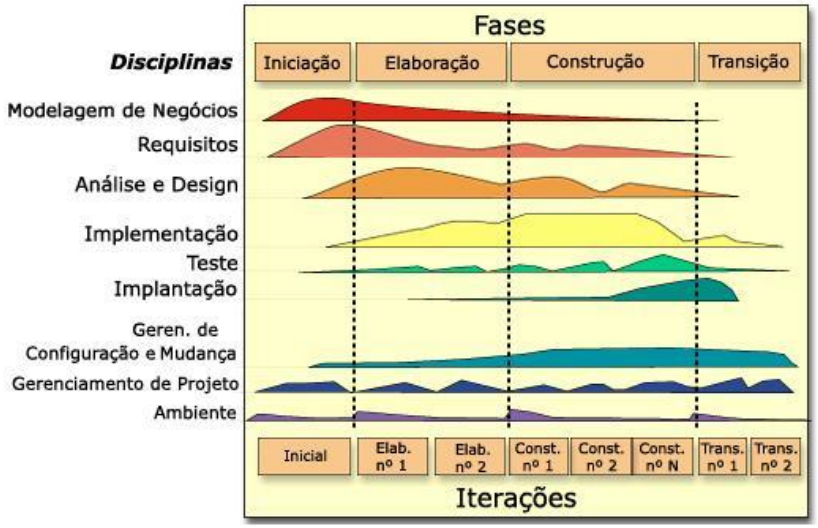


Figura 7 - Fases do RUP
Fonte: (IFSC, 2009)

O processo unificado é uma tentativa de apoiar-se nos melhores recursos e características dos modelos convencionais de processo de *software*, mas caracterizá-los de um modo que implemente muitos dos melhores princípios do desenvolvimento ágil. (PRESSMAN, 2006)

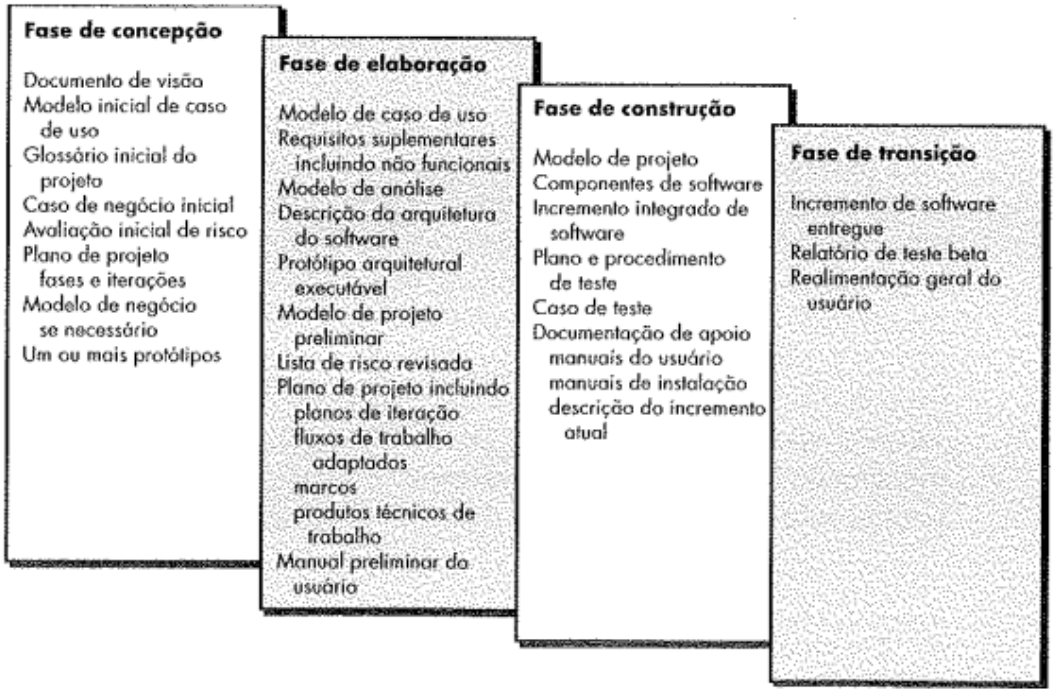


Figura 8 - Artefatos gerados em estágio do RUP
Fonte: (PRESSMAN, 2006)

Os demais itens relacionados ao processo unificado estão detalhadamente descritos na sessão 3 com foco no modelo *Openup*.

2.6 EPF

O EPF é um projeto aberto de um *framework* mantido pela Fundação *Eclipse* que visa abranger os diversos processos de desenvolvimento de *software* dos mais diversos tipos existentes, ele é customizável e vários exemplos para serem utilizados como base. Além disso, o *framework* possui algumas ferramentas que aplicam suas próprias técnicas para auxiliar nos processos de engenharia de *software*.

O *framework* foi criado focando em dois pontos, são eles: (ECLIPSE, 2011)

- Prover um *framework* extensível e ferramentas exemplares para a engenharia de processos de *software* – Criação de métodos e processos, gerenciamento de bibliotecas de processos, configuração e publicação do processo.

- Prover conteúdo de processo exemplar e extensível para uma variedade de tipos de desenvolvimento de *software* e gerenciamento dos processos suportando ciclos iterativos, ágeis e desenvolvimento incrementais aplicáveis a um vasto conjunto de plataformas de desenvolvimento e aplicações.

O núcleo (Figura 9) da ferramenta é extenso e adaptável a todas as necessidades dos diferentes tipos de empresa, é possível visualizar que possuem inúmeros plug-ins de diferentes metodologias e conhecimentos diversos. E, segundo KROLL (2011), praticamente todos podem obter benefícios do *framework*, desde projetos individuais onde pode ser adaptado com diferentes processos e customizável para o próprio projeto até empresas de grande porte que possuem conhecimentos próprios sobre tecnologias especificas aplicando o EPF e distribuindo o conhecimento.

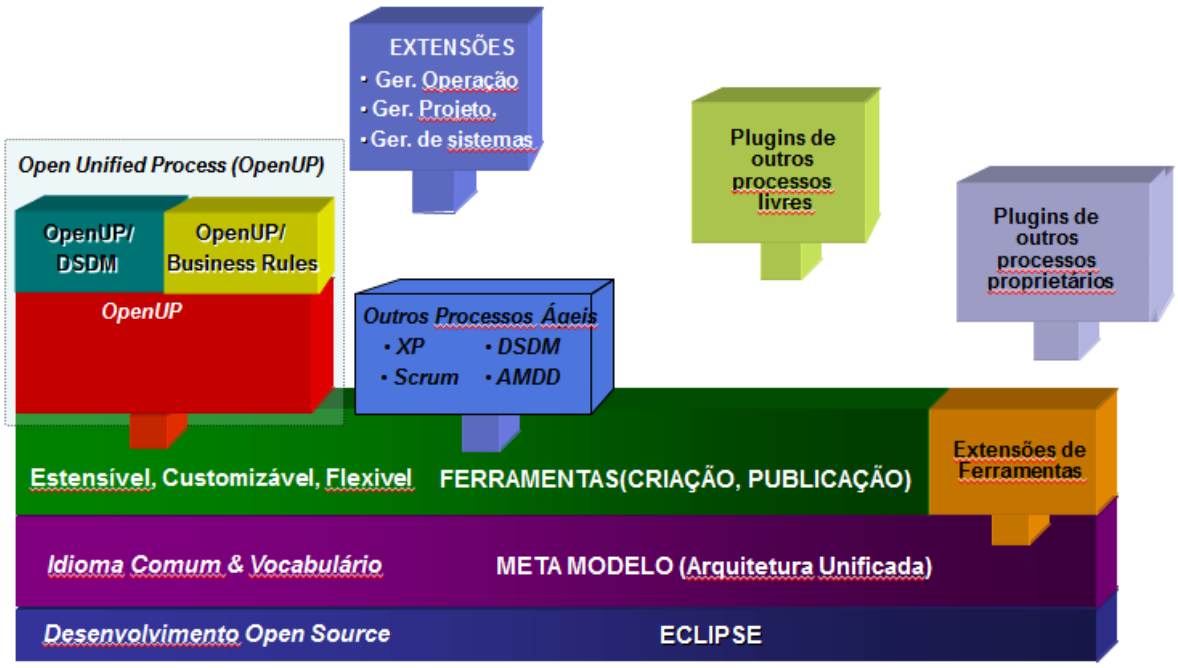


Figura 9 - Núcleo base do *Eclipse Process Framework*
Fonte: (KROLL e LYONS, 2010)

Juntamente com o *framework* é fornecido um produto chamado *Composer*, este produto é um *software* para a engenharia de processos, para líderes de projeto, para o projeto e para as pessoas responsáveis por realizar a manutenção e implementação da organização do desenvolvimento ou projetos individuais. O foco

do *Composer* (Figura 10) é criar um *software* o qual pode ser utilizado na criação e customização de processos de uma maneira extensível.

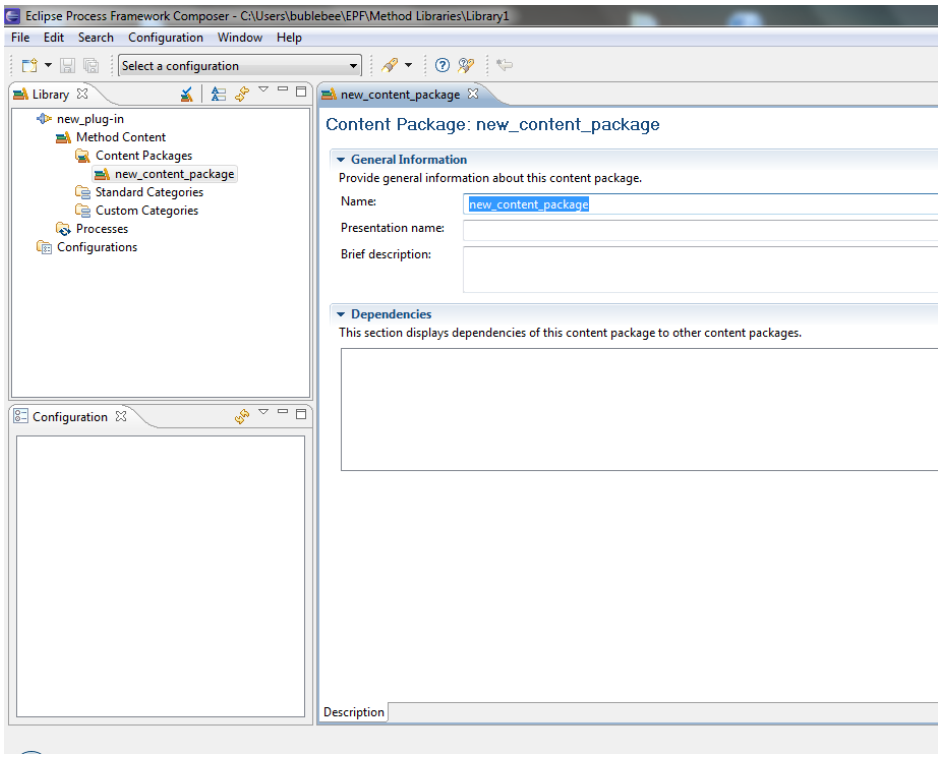


Figura 10 - Eclipse Composer

3 *OPENUP*

Openup é uma metodologia que segue o Processo Unificado de engenharia de *software* porém com uma cultura aberta e enxuta, ele utiliza a filosofia ágil a qual foca na natureza colaborativa de desenvolvimento de *software*. A metodologia é construída através de processos simples os quais podem ser expandidos para quaisquer tipos de *software*s.

O *Openup* foi criado como um projeto de exemplo aplicando as melhores práticas de engenharia de *software* do EPF, assim sendo, o *Openup* é um conjunto das metodologias ágeis como *Scrum* e XP (*Extreme Programming*) somado a um pouco da arquitetura do Processo Unificado.

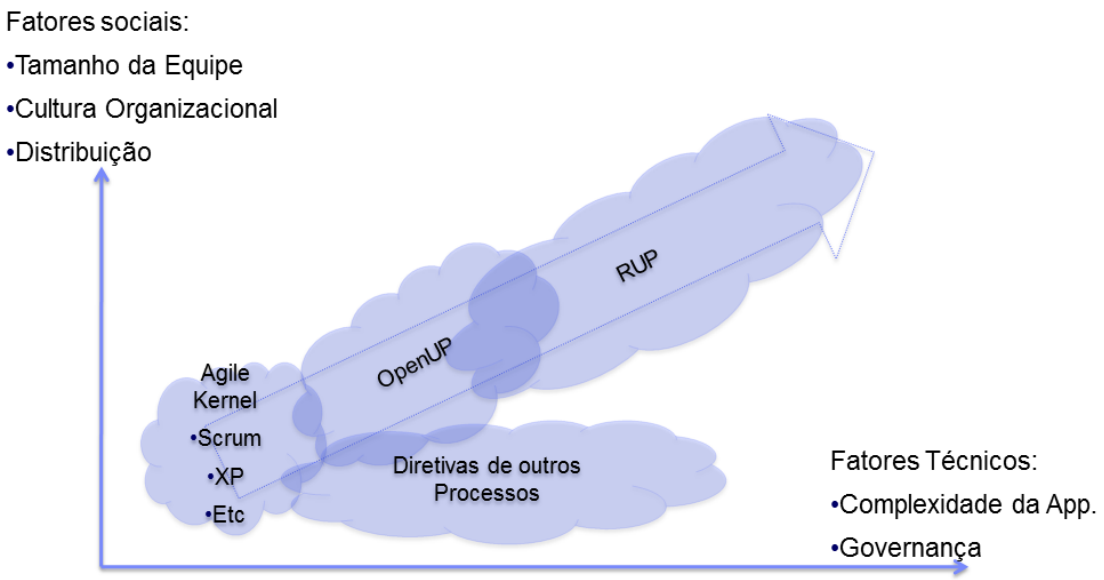


Figura 11 - Perspectiva *Openup*
Fonte: (ESSENTIALS, 2011)

Openup é possui seu núcleo em desenvolvimento aberto pela comunidade, assim ele pode ser facilmente adaptado para qualquer tipo de desenvolvimento de *software,* torna-se possível também adicionar práticas e processos únicos aplicados pela cultura organizacional da empresa.

Em seu ciclo de vida existem micro incrementos organizados visando diminuir o *feedback*. De acordo com KROLL (2007), este processo estimula a colaboração intensiva assim que o sistema é desenvolvido utilizando um ciclo incremental em uma equipe dedicada.

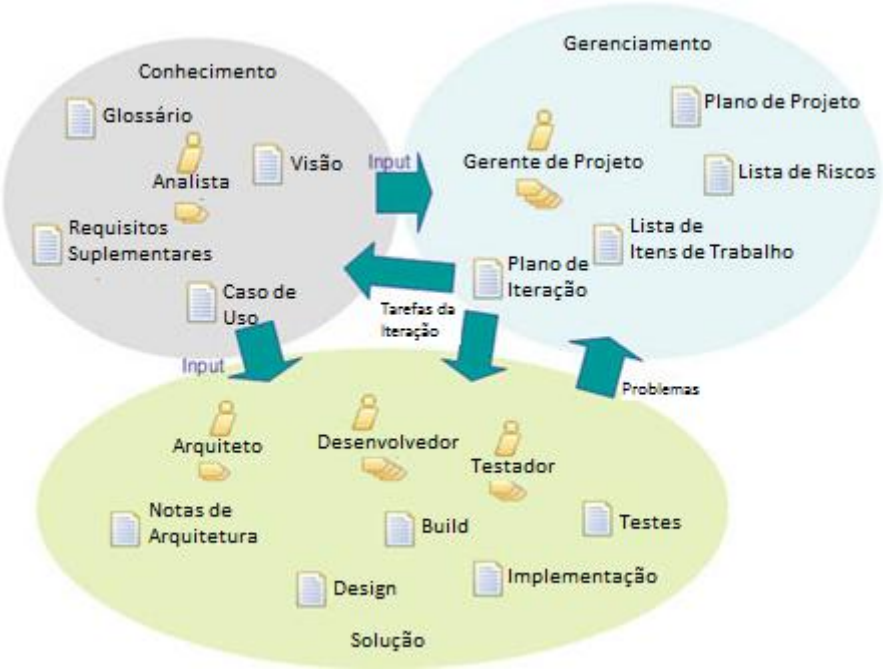


Figura 12 - Disposição do *Openup*
Fonte: (GUSTAFSSON, 2008)

Através da metodologia é estabelecido um conjunto de papéis, tarefas e artefatos que são utilizados pelos membros da equipe durante a execução e gerenciamento do projeto (Figura 12). Como a metodologia é aberta também é disponibilizada uma página (Figura 13) a qual contém todo o conteúdo necessário para a implantação e utilização.

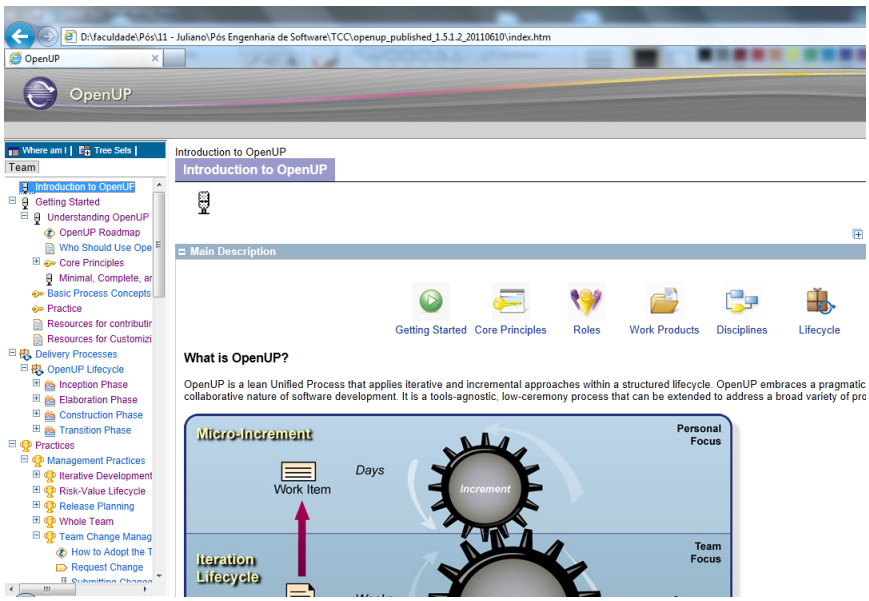


Figura 13 - Página com conteúdo do *Openup*

3.1 PRINCÍPIOS E PREMISSAS

Toda metodologia segue no mínimo um princípio para atingir metas, devido a isto se pode dizer então que o *Openup* possui em seu núcleo um conjunto de premissas que faz dele uma metodologia funcional e que ao seguir estas premissas é possível obter boas referências ao projeto e tornar este um produto com qualidade e funcional.

A seguir estão descritas os quatro pilares nos quais a metodologia se sustenta para trazer bons resultados e o conjunto de melhores práticas para ajudar a atingir o resultado e uma associação entre os princípios descritos e os princípios do Manifesto Ágil (Quadro 1).

Princípio *Openup*	**Princípio no Manifesto Ágil**
Colaborar para alinhar os interesses e compartilhar os conhecimentos	Indivíduos e iterações acima de processos e ferramentas
Balancear as prioridades concorrentes para maximizar os valores dos *stakeholders*	Colaboração com o cliente acima da negociação de contrato.
Focar primariamente na arquitetura visando minimizar os riscos e planejar o processo	*Software* funcional invés de documentação compreensiva.
Envolver os *stakeholders* para obter contínuo *feedback* do desenvolvimento	Responder a mudanças ao invés de seguir os planos.

Quadro 1 - Mapeamento entre os princípios *Openup* e manifesto ágil
Fonte: (BALDUINO, 2007)

3.1.1 Balancear as prioridades concorrentes para maximizar os valores dos stakeholders

Promover práticas que permitam à equipe de desenvolvimento e aos *stakeholders* desenvolver uma solução que contrabalanceie todas as necessidades

da parte interessada e que esteja de acordo com as restrições propostas no projeto. (DISTILIED, 2011)

Esta regra se aplica a para resultar em um acordo extremamente importante que é a entrega do projeto para satisfazer as necessidades do *stakeholder*, porém muita das vezes é neste ponto que há divergências por falta de conversa ou falta de entendimento da necessidade por parte do projetista, para atingir este princípio é necessário ter três pontos de conhecimento comum e aprovado entre os interessados pelo sistema e pela equipe desenvolvedora da mesma, são eles:

- O problema a ser resolvido;
- As restrições inerentes à equipe de desenvolvimento (custo, recursos, habilidades, etc.);
- As restrições inerentes à solução proposta.

Para se atingir este resultado é esperado que fossem seguidos oito práticas deste escopo, as práticas podem ser visualizadas no APÊNDICE A.

3.1.2 Colaborar para alinhar os interesses e compartilhar os conhecimentos

Promover as práticas que promovam um ambiente saudável de desenvolvimento em equipe, possibilitando a colaboração e possibilitando a compreensão e concordância sobre os principais requisitos que definem o sistema. (DISTILIED, 2011)

Em um ambiente saudável é esperado que a equipe conseguisse um melhor desempenho, que o desenvolvimento flua melhor e com mais qualidade, já quando há imprevistos, discussões internas acabam causando limitando um membro da equipe a pedir ajuda de outro membro o qual possui o conhecimento necessário para realizar a resolução de um problema. As melhores práticas estão descritas no APÊNDICE B.

3.1.3 Focar primariamente na arquitetura visando minimizar os riscos e planejar o processo

Promover as práticas que permita à equipe de desenvolvimento focar suas ações na arquitetura, buscando minimizar os riscos e organizar o processo de desenvolvimento da solução proposta. (DISTILIED, 2011)

Segundo MEIRA (2010), a arquitetura de um *software* pode ser entendida como a representação da organização ou estrutura dos componentes significantes do sistema, bem como, a representação dos comportamentos desses componentes. Sem uma arquitetura íntegra e funcional o projeto de *software* evolui de forma ineficiente causando a incapacidade de ser extensível e reutilizável, neste ponto também se torna aplicável que todos os membros da equipe possuam um conhecimento comum sobre a arquitetura do *software* e aspectos técnicos assim é possível minimizar riscos e organizar o desenvolvimento. As melhores práticas estão descritas no APÊNDICE C.

3.1.4 Envolver os stakeholders para obter contínuo *feedback* do desenvolvimento

Promover práticas que permitam à equipe de desenvolvimento obter *feedback* contínuo dos *Stakeholders* sobre a solução proposta e demonstrar o incremento de seu valor. (DISTILIED, 2011)

Muitas vezes é impossível saber todos os riscos envolvidos em uma aplicação, muitas vezes o escopo pode alterar antes do fim do desenvolvimento, então se torna necessário que a equipe possa demonstrar valores de forma incremental para os *stakeholder* de formar que consiga descobrir possíveis trocas prematuramente.

Com este pensamento a equipe acaba de acostumando com os clientes e com a sua maneira de pensar, assim encontram maneiras de se comunicar e trabalharem juntos. O ataque prematuro aos riscos do projeto e o constante trabalho em cima do *feedback* dos clientes tendem a gerar um resultado final com maior valor agregado. (MEIRA, 2010) As melhores práticas estão descritas no APÊNDICE D.

3.2 ESCOPOS DO FRAMEWORK

A melhor maneira de entender o *Openup* é saber quais são as necessidades e os objetivos da equipe para desenvolver o *software*, umas vez que o *Openup* é aberto para alterações é possível que seja facilmente adaptável para equipe ou pode-se escolher um escopo de conhecimento mínimo, porém, para chegar a esta conclusão se faz necessário que sejam respondidas as três perguntas. (GUSTAFSSON, 2008)

- Faz-se necessário somente o mínimo dos processos para atingir algum valor?
- Faz-se necessário se não utilizar alguns produtos de trabalho afim de não ser sobrecarregado?
- Faz-se necessário utilizar um processo que pode ser alterado e extensível para necessidades adicionais que podem ocorrer durante a execução do projeto?

Almejando a capacidade de aplicação ao maior número de projetos possíveis, bem como, à maior gama de equipes possíveis, independentemente de suas culturas de trabalho, a equipe Eclipse projetou o *Openup* buscando atingir às seguintes características. (BALDUINO, 2007)

- Metodologia Mínima: Somente os conteúdos fundamentais são adicionados.
- Metodologia Completa: É o conjunto de processos completos para desenvolver um sistema.
- Metodologia Extensível: Pode ser utilizados como uma base para outros processos criados pela própria empresa onde cada processo pode ser alterado de acordo as necessidades.

Desta maneira, torna-se possível limitar ou estender a metodologia facilmente, quando, por exemplo, não há grandes recursos disponíveis ou o projeto não possui grande complexidade pode-se utilizar todos os conteúdos que são ditos pelo *Openup* como essenciais. Ou, se necessário, é possível adicionar novos passos para o desenvolvimento utilizando a forma Extensível, alterando *templates* de artefatos ou até mesmo renomeando os papéis disponíveis nas equipes.

3.3 CICLO DE PROJETO

Assim como o Processo Unificado o *Openup* se baseia em um ciclo com iterativo e incremental, entretanto, o *Openup* foca em criar um modelo ágil e enxuto , o que torna outro diferencial da metodologia é que ela possui além da iteração normal, um micro incremento, este microincremento é uma pequena unidade de trabalho de uma pessoa da equipe e o conjunto são executadas em uma iteração. O funcionamento completo do ciclo de vida do *Openup* pode ser visualizado na Figura 14.

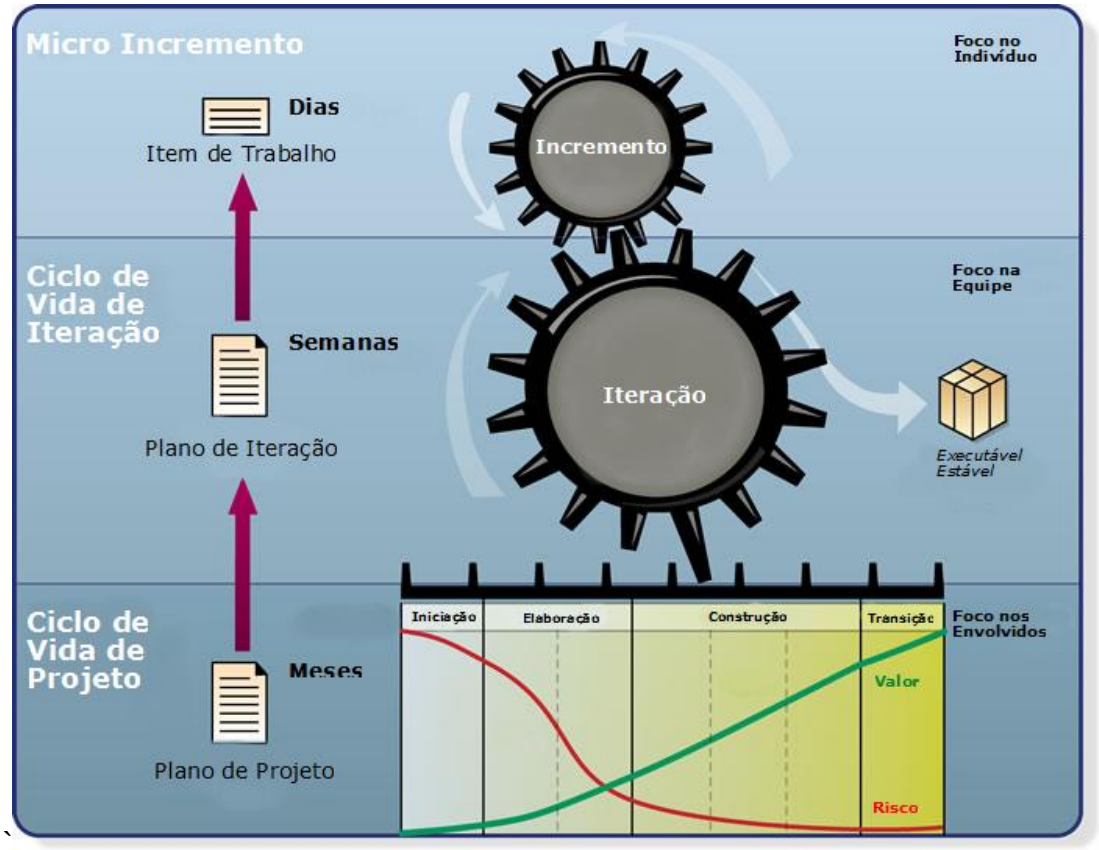


Figura 14 - Ciclo de Vida *Openup*
Fonte: (GUSTAFSSON, 2008)

Durante as quatro fases é necessário que alguns itens sejam respondidos e completos, a não existência de algum item ou algum item incompleto significa que o projeto pode atrasar ou ter problemas, são eles:

- Fase Concepção: Neste ponto todos devem concordar com o escopo e objetivos do projeto.

- Fase de Elaboração: Na elaboração é necessário que haja uma arquitetura concreta e funcional, além de ter os riscos mitigados e possuir valores aceitáveis para a entrega final.
- Fase de Construção: Neste ponto é realizado o desenvolvimento do sistema, que ao final devem-se haver entregas incrementais demonstrando valores funcionais ao *stakeholder*.
- Fase de Transição: Se a aplicação conseguir sair da fase de construção é necessário que ela seja totalmente funcional e sem problemas para ser colocada em produção.

O ciclo de vida do projeto possui tempo a ser terminado, e todas as datas de atualização e fases a serem seguidas são declaradas em um artefato chamado de Plano de Projeto.

O ciclo de vida do projeto é baseado em quatro fases (Figura 15), dependendo em qual das quatro fases o projeto se encontra é possível e necessário tomar ações diferentes de acordo com o risco detectado. De acordo com KROLL (2007), o ciclo de vida do projeto fornece uma visão geral, com transparência e com mecanismos para gerenciar os custos do projeto, o escopo, riscos, valores, e outros aspectos do processo.

Figura 15 - Ciclo de Vida do Projeto
Fonte: (KROLL, 2007)

3.4 CICLO DA ITERAÇÃO

Segundo KROLL (2007), as iterações do *Openup* mantém a equipe focada na entrega de valores para o cliente a cada algumas semanas entregando um protótipo funcional da aplicação. Desta maneira, em todas as algumas semanas é possível detectar erros ou requisitos mal levantados, e quando achados não possuem um tempo para discussão, o foco é somente corrigir os erros e continuar com o cronograma.

O plano da iteração, tempo gasto e progresso são centrados em itens entregáveis, estes itens possuem prioridades e quanto mais alta a prioridade do caso de uso mais cedo deve entrar em um plano de iteração. Porém se faz necessário em certos casos preencher o plano de iteração com conteúdos que serão facilmente entregues e que aproveitem melhor o plano de iteração.

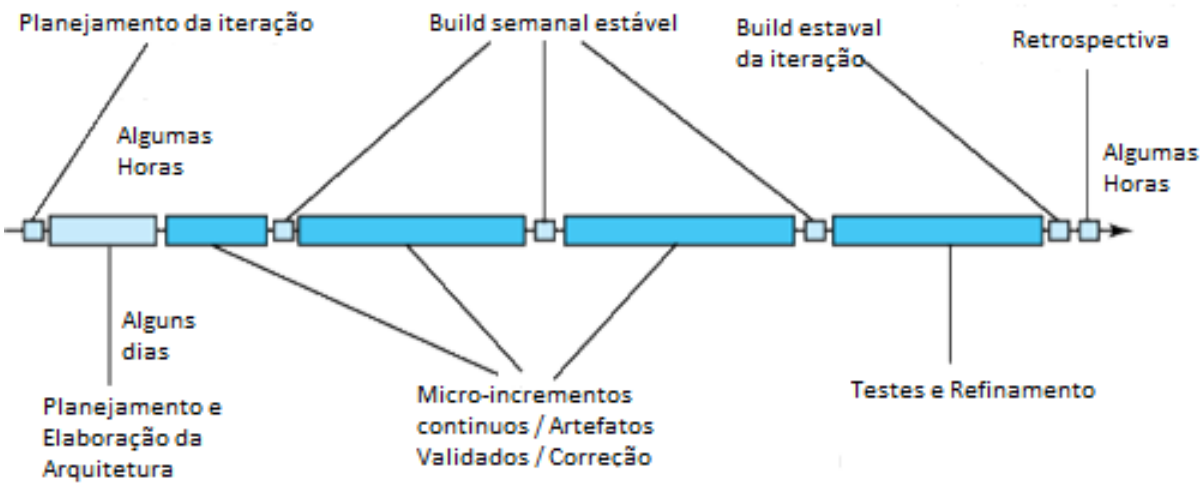


Figura 16 - Ciclo de vida da iteração no *Openup*
Fonte: (KROLL, 2007)

Uma iteração inicia-se com uma reunião a qual são eleitos os casos de uso a serem implementados, a reunião não deve demorar mais que algumas horas e devido a isso a equipe deve ser bem centrada na decisão que é montar o Plano de Iteração. Nos próximos dias são discutidos os impactos os itens escolhidos na arquitetura do sistema e são verificadas as ordens lógicas dos itens de trabalhos

escolhidos, e após isto a execução do plano de iteração é feito dos microincrementos.

Ao final da iteração é necessário que todo o conjunto de item de trabalho esteja completo e funcional, pois senão causam atrasos e problemas tardios.

3.5 MICRO-INCREMENTO

Os microincrementos são pequenas unidades de trabalho pessoais de um ou mais membros da equipe, as unidades de trabalho podem representar algumas horas a alguns dias de trabalho em algum item de produto de entrega aceitável pré-estabelecido no Plano de Iteração. Este provê um retorno extremamente rápido dos envolvidos no projeto o que torna possível tomar pequenas decisões que podem corrigir erros.

De acordo com KROLL (2007), o conceito de um microincremento ajuda ao membro da equipe a particionar o seu trabalho em unidades menores as quais quando terminadas entregam uma pequena quantidade de valor para a equipe. Um microincremento equivale ao *Sprint* disponibilizado pela metodologia *Scrum*.

É importante que a cada microincremento terminado tenha um novo item de trabalho entregue e que na conclusão do sejam adicionados os resultados em uma lista comum para que todos os membros da equipe vejam o resultado da iteração com transparência.

3.6 PAPÉIS

Os papéis em uma equipe auxiliam a diferenciar as tarefas e atribuições que cada grupo de pessoas irá realizar, esta separação é feita através dos conhecimentos dos indivíduos ou mais comumente de acordo com o seu cargo na empresa. O *Openup* separa os papéis em 7 grupos, são eles:

- Analistas: Pessoa responsável por capturar as necessidades do usuário final e priorizará os requisitos, além disso, ele é a ponte entre a equipe interna e os clientes.

- Arquitetos: O arquiteto é responsável por definir a arquitetura de *software*, e faz parte de sua função realizar decisões técnicas sobre o *design* e implementação do sistema.
- Desenvolvedores: O desenvolvedor é o responsável por pegar os dados levantados pelo analista e transformá-los em pontos funcionais do sistema, seguindo as regras definidas pelo arquiteto e integrando componentes da solução e criando testes unitários[5].
- Gerente de Projeto: O gerente realizando o planejamento do *software* coordena iterações com os clientes e matem a equipe focada nos objetivos.
- *Stakeholders*: É caracterizado pelo grupo interessados no projeto, um membro deste grupo deve estar no mínimo afetado com escopo do projeto.
- Testadores: É responsável por identificar, implementar, conduzir, registrar e analisar os testes necessários para garantir a qualidade do *software*.

Cada grupo possui a sua importância e é fundamental que haja a comunicação transparente entre todos do grupo, pois a falha em qualquer grupo pode causar risco ao projeto. No APÊNDICE A pode ser visualizada a referência entre o grupo de trabalho e as ações que cada um realiza detalhadamente.

Além dos grupos considerados básicos como *stakeholders*; analistas e desenvolvedores, o *Openup* fornece um novo grupo com atribuições especiais, o grupo “Qualquer outro Grupo”. Neste grupo é atribuído todo e qualquer membro da equipe que tem como foco realizar tarefas gerais, porém que auxiliam os demais grupos diretamente, por exemplo:

- Acesso a artefatos no sistema de controle de versão a fim de atualizar artefatos ou melhorar.
- Realizar o processo de pedidos de mudança.
- Participar em revisões.
- Ser voluntário de participar em uma iteração particular.
- Fazer café para os desenvolvedores.

[5] O teste unitário é implementado com base no menor elemento testável (unidades) do software e implica em testar a estrutura interna (como fluxo lógico e de dados), a função da unidade e os comportamentos observáveis. (WTHREEX, 2002)

A ideia principal deste grupo é auxiliar na realização de pequenas tarefas do cotidiano de cada membro das outras equipes e melhorar a qualidade do ambiente de trabalho.

3.7 DISCIPLINAS

O grupo de ações a serem realizadas durante o ciclo de vida do projeto é dividido em várias disciplinas e estas representam os produtos de trabalho que devem ser completados antes do término do ciclo, cada disciplina dependendo no estágio no qual o projeto se encontra pode possuir uma ação e contribuição da equipe do que as outras disciplinas. (Figura 17)

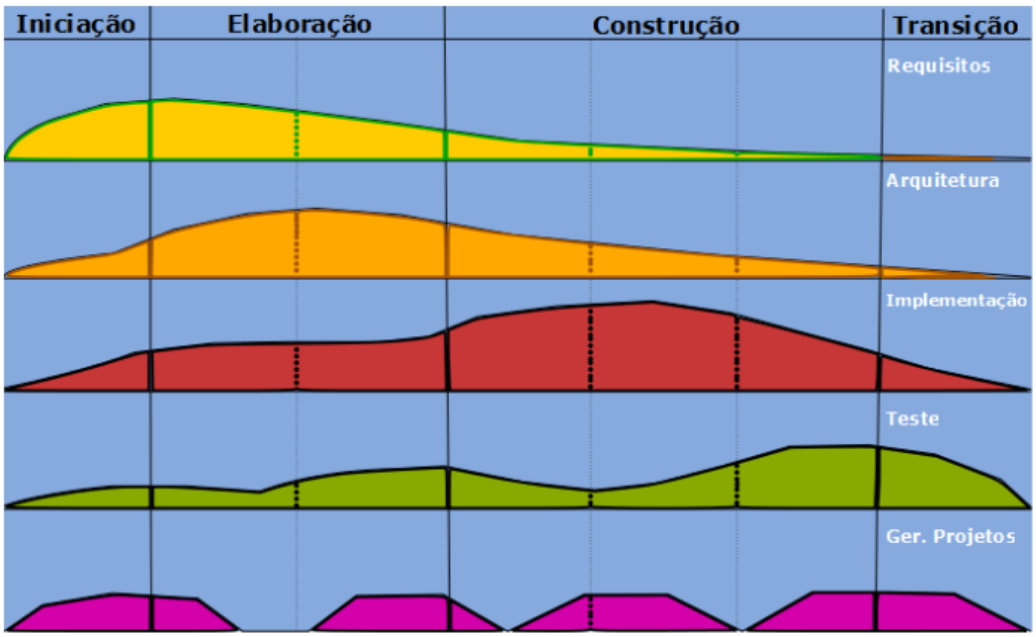


Figura 17 - Separação do ciclo de vida do projeto e disciplinas
Fonte: (MEIRA, 2010)

A disciplina de Requisitos agrupa todas as tarefas relativas ao processo de Análise de Negócios e de Análise e Especificação de Requisitos do *Openup*, tendo maior ênfase na fase de Iniciação do Processo. (MEIRA, 2010)

Para atingir os objetivos finais do projeto e a satisfação da parte interessada é necessário que se tenha bem estabelecidos logo no começo do projeto as metas,

para isso é necessário identificar os s*takeholders*, possíveis riscos e não só as necessidades que o mesmo relata, mas também se devem identificar outras necessidades que possam ser levantadas durante a execução do projeto. Com base nesta disciplina são tomadas as decisões para os estágios futuros e as decisões de arquitetura do projeto.

A disciplina de Arquitetura tem como principal objetivo apresentar uma arquitetura estável para o desenvolvimento do sistema, baseando-se nos requisitos especificados. A disciplina de Arquitetura tem maior ênfase na fase de Elaboração. (MEIRA, 2010)

Neste estágio são convertidos os requisitos e necessidades levantadas no *design* do próprio projeto, e assim são definidas as alterações na arquitetura para suprir as necessidades do projeto. Para auxiliar na representação da arquitetura são utilizados outros artefatos como diagramas disponíveis na UML (*Unified Model Language*) e outros documentos que os desenvolvedores e analistas podem utilizar durante a execução o projeto.

A disciplina de Implementação organiza as tarefas que irão transformar a arquitetura proposta na implementação do sistema, buscando atender os requisitos definidos pelos *stakeholders*. (MEIRA, 2010)

Nesta fase são desenvolvidos os requisitos utilizando como base de informação os documentos gerados na fase da elaboração, os requisitos são transformados em componentes executáveis e testáveis pelos clientes. A separação de qual requisito desenvolver é dividida nos microincrementos das iterações, e ao final de cada iteração é necessário que haja um produto executável para ser mostrado aos futuros usuários para achar possíveis erros ou má interpretação dos requisitos.

A disciplina de Teste agrupa as tarefas relacionadas a teste, que se preocupam em prover *feedback* sobre a maturidade do sistema, projetando, implementando, executando e avaliando testes. (MEIRA, 2010)

Esta disciplina visa criar projetos de qualidade aceitáveis pelos clientes realizando testes precoces a cada iteração e microincrementos também são realizados testes de capacidade para ver se os sistemas conseguem suprir as necessidades do ambiente no qual será utilizado.

A disciplina de Gerência de Projeto tem como objetivo apresentar técnicas para que o gerente de projetos possa liderar facilitar e oferecer suporte à sua

equipe, auxiliando-a a lidar com os riscos e obstáculos encontrados durante o processo de desenvolvimento de *software*. (MEIRA, 2010)

Nesta disciplina é necessária que haja um fator líder na equipe, este líder devera motivar os membros desenvolvedores e analistas assim como enfrentar a parte interessada no projeto em casos de riscos identificando requisitos desnecessários e que possam atrasar o projeto. É o papel do gerente do projeto estabelecer metas e ações corretivas a fim de sempre proporcionar um ambiente colaborativo e um foco para atender todas as soluções propostas.

4 PROCEDIMENTOS METODOLÓGICOS DA PESQUISA

Para demonstrar a utilização dos recursos do *Openup*, foi criado um estudo de caso assumindo um projeto fictício sem uma aplicação em um ambiente real envolvendo *stakeholders* e equipes não inexistentes.

Serão apresentados os principais artefatos gerados e seus benefícios no gerenciamento do projeto assim como as melhores práticas que foram aplicados para ser atingidos os mesmos, ou, as melhores práticas que podem melhorar o ambiente do projeto na devida fase.

4.1 LOCAL E ESCOPO DO SISTEMA

O sistema hipotético utilizado será de uma distribuidora de produtos para a empresa *DistriMais*, este sistema terá como foco principal receber pedidos dos clientes que desejam comprar produtos.

No escopo do sistema deverá gerenciar as faturas dos clientes que devem ser pagas até a data do seu vencimento; as faturas dos fornecedores que são controladas por outro sistema.

Além disso, periodicamente, o fornecedor deve atualizar seus produtos assim a distribuidora envia catálogo de produtos aos clientes ao fim de cada mês o qual deve estar registrado no sistema.

A empresa da equipe desenvolvedora localiza-se em Foz do Iguaçu e atende pelo nome de Desenvolvimento TRI-LOGIC.

4.2 COLETA DOS DADOS

Durante a confecção deste estudo foi elaborado a utilização dos artefatos fornecidos pela metodologia de acordo com o progresso do projeto, os dados

utilizados nestes documentos foram inventados para simular a real utilização dos mesmos.

Os dados têm como finalidade focar no sistema de distribuidora, além disso, outros dados podem fornecer uma visão fora do sistema simulando um curso anormal no desenvolvimento, por exemplo, um requisito não identificado durante a concepção do projeto.

4.3 ANÁLISE DOS DADOS

A análise dos dados será feita contanto como todos os artefatos e suas informações são relevantes para o processo através de evidencias perceptíveis, também será descrito os resultados das informações cruzando com o conteúdo do material do referencial teórico que norteiam o desenvolvimento do estudo de caso.

Além dos artefatos gerados serão avaliadas as melhores práticas aplicáveis em cada faze do projeto e sua ajuda no projeto como um todo que fazem gerar resultados de qualidade aplicáveis em um ambiente real de desenvolvimento de um *software*.

Em suma, será mostrada a análise feita dos resultados alcançados (Artefatos) e do emparelhamento dos resultados com a fundamentação teórica (Melhores práticas).

5 RESULTADOS E DISCUSSÃO

Para facilitar a apresentação e discussão dos resultados, os mesmos foram divididos em itens, de acordo com a execução dos processos durante o ciclo de vida do projeto de *software*. A análise descritiva foi realizada separadamente, considerando os seguintes grupos: planejamento, análise de requisitos, desenvolvimento da solução e testes. É apresentado também um tópicos com a comparação entre a metodologia de desenvolvimento em cascata com o *Openup*.

5.1 PLANEJAMENTO E EXECUÇÃO DAS ITERAÇÕES

Durante o planejamento do projeto é possível notar que o foco dos artefatos é o essencial para atingir as metas do cliente, no documento de visão (APÊNDICE G) é possível notar o foco nos requisitos principais e na concepção do conhecimento da solução que é o *software* da distribuidora de alimentos.

Também é desenvolvido o plano de projeto (APÊNDICE H), este documento para guia os membros da equipe em como o projeto deve ser desenvolvido, o importante deste artefato é que ele descreve alguns dados simples como organização e quais práticas serão seguidas, porém, de suma importância pra novos membros de equipes.

Antes de ser iniciada a iteração o gerente cria um plano de iteração do projeto (APÊNDICE I), neste plano de iteração possuem os objetivos da iteração e quais são os pontos para considerar o nível de aceitação da iteração, através dos planos de iteração são avaliados os resultados no final da iteração. Além disso, o foco do plano de iteração é ser um ponto central de informação da iteração para todos os membros da equipe.

Para gerenciar as tarefas em andamento são utilizados gráficos no estilo *burndown*[6], este tipo de facilita a compreensão do andamento do projeto por ser uma maneira de demonstração visual. (Figura 18)

[6] O gráfico de burndown é considerado um dos mais úteis para monitorar o progresso de um time ágil. O gráfico representa a quantidade de trabalho que falta ser feito no eixo vertical (y) versus o tempo no eixo horizontal (x). (HAZRATI, 2010)

Neste gráfico é possível visualizar que existem um número de pontos (Eixo X) para cada número de iterações (Eixo Y) realizadas, ao ponto de vista de gerenciamento estes pontos são dívidas que devem ser pagas antes da finalização de todas as iterações, ou seja, ao serem realizadas todas iterações todas as tarefas foram realizadas zerando o número total de pontos.

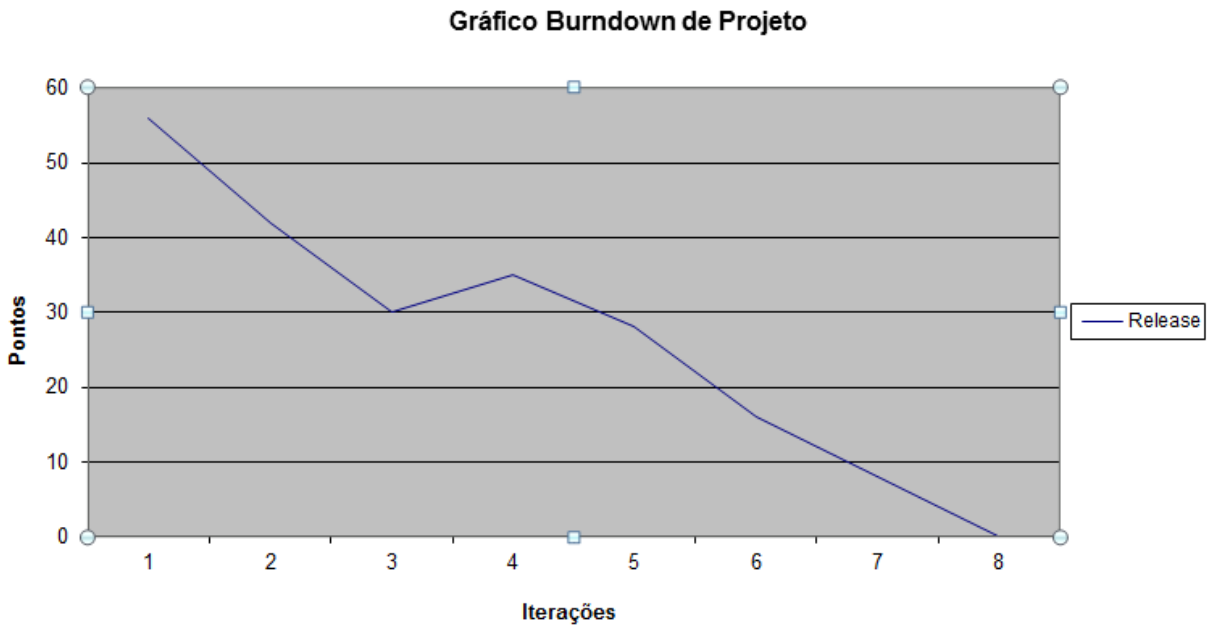


Figura 18 - Gráfico Burdown do Projeto

A atividade de gerenciamento é realizada durante todos os processos de ciclo de vida do projeto, a cada iteração é analisado e realizado um trabalho especial para identificar riscos e problemas o mais cedo possível, assim, nas próximas iterações os mesmos são adicionados para serem resolvidos e prevenidos antes que aconteçam.

É valida também, ao término de uma iteração a priorização de itens de trabalho (Quadro 2) que focam na entrega de maior valor na próxima entrega gerente de projeto, stakeholder, e membros da equipe concordam com os itens que supostamente devem ser desenvolvidos. Para atingir também uma estimativa apurada sobre o tempo de desenvolvimento de cada item de trabalho cada membro realiza uma quebra destes itens em tarefas de desenvolvimento com um tempo de estimativa, assim é possível criar um emparelhamento do tempo gasto em itens de trabalho de uma determinada iteração.

Nome / Descrição	Prioridade	Estimativa (Points)	Responsável	Estimativa(hours)
Suportar um controle de vendas simples	5	8	João	48
Realizar o *design*			Carlos	12
Implementar e testar códigos do servidor (Server)			Lucas	14
Implementar e testar códigos do cliente (PDV)			Lucas	28
Atualizar documentação de usuário			Maria	6
Produzir uma versão demo e disponibilizar para stakeholders	10	5	Lucas	4
Atualizar documentação final	2	5	João	65
Atualizar manual de instalação	2	1	João	5
Atualizar notas de versão	2	1	João	4

Quadro 2 - Priorização de itens de trabalho

Outro item importante notado na utilização do *Openup* é que mesmo durante a execução do projeto as lições aprendidas são capturadas afim de modificar as próximas iterações melhorando o processo do mesmo e quando necessários são realizados ajustes para atingir o escopo da iteração nas próprias micro-incremento.

Além das melhores práticas aplicáveis como o desenvolvimento iterativo; ciclo de vida baseado em riscos; foco de entrega ao final da iteração é utilizado para suprir necessidades à prática de alocar sempre que necessário e possível um membro de equipe para outra tarefa (*Team Change Management*). Esta prática permite que qualquer possa ser alocado para outras tarefas de maneira mais flexível suprindo as necessidades de um determinado requisito, porém, o membro da equipe já deve estar familiarizado com o escopo do projeto.

5.2 IDENTIFICAÇÃO E REFINAÇÃO DOS REQUISITOS

Durante a execução da identificação dos requisitos, tornou-se necessário especificar, analisar e validar um conjunto de requisitos do sistema de acordo com a

implementação. Este conjunto de ações é realizado para garantir que tanto a equipe de desenvolvimento quanto os *stakeholders* tenham a mesma visão consistente sobre os requisitos.

É possível notar que através das diferentes fases do ciclo do projeto a atividade recebe um foco diferente, com aspecto de interesse diferente, por exemplo, na fase da concepção o foco é entrar em acordo com o problema a ser resolvido, identificando as necessidades do *stakeholder* e descobrindo requisitos de alto nível do sistema.

Na fase de elaboração o foco é em definir uma proposta para a solução, a proposta consiste em identificar os requisitos que têm mais valor para o stakeholder e que influem diretamente na arquitetura do *software*, ou, que possuem riscos de serem implementados. Os itens que são identificados como prioritários são adicionados à lista de Itens de Trabalho (Figura 19) para a implementação logo nas primeiras iterações, assim os itens prioritários servem de base para validação e entendimento do projeto juntamente com o *stakeholder*.

A	B	D	E	F	G	H
Nome / Descrição	**Prioridade**	**Estado**	**Iteração**	**Responsável**	**Estimativa (hours)**	**Horas Trabalhadas**
Definição da Arquitetura	10	OK	I1	Lucas	5	3
Caso de Uso 01	10	OK	I2	José	48	40
Caso de Uso 02	3	OK	I3	José	8	12
Caso de Uso 03	5		I4	Maria	22	
Caso de Uso 04	4		I5	José	15	
Caso de Uso 05	7		I6	Maria	30	
Caso de Uso 06	7		I7	Maria	30	
Caso de Uso 07	7		I8	Maria	30	
Caso de Uso 08	9		I9	José	36	
Levantamento de Teste	10		I10	João	48	

Figura 19 - Lista de itens de trabalho parciais

O foco da fase de construção é o refinamento da definição do sistema que consiste em detalhar os itens restantes e associar os casos de teste guiando o desenvolvimento e verificando e gerenciando alterações.

Mesmo com os requisitos bem definidos é necessário que seja realizado a conversão dos requisitos funcionais para casos de uso, os casos de uso fornecem uma visão melhorada sobre as ações do sistema, ajuda na identificação de casos de teste, auxilia na escrita da documentação do usuário e para os desenvolvedores do

software é a fonte principal de informação. Os casos de uso são definidos em um único documento de casos de uso (APÊNDICE J) seguindo o modelo fornecido pelo *Openup*, os modelos do casos de uso definem o essencial para a implementação do sistema, por exemplos, as pessoas que utilizarão a função específica e o curso normal do processo (Quadro 3).

Caso de Uso	**RegistrarPedido – RP1**
Atores	Cliente (Iniciador)
Finalidade	Registrar um pedido feito por um cliente.
Visão Geral	O cliente solicita determinados produtos através de um pedido. Caso o cliente não esteja cadastrado ou os produtos sejam inexistentes, o pedido é descartado.
Tipo	Primário
Pré-Condições	O cliente deve estar cadastrado no sistema. Todos os produtos devem estar cadastrados no sistema.
Sequência Típica de Eventos	
Ação do Ator	*Resposta do Sistema*
1 – Cliente informa seus dados	2 – Sistema recebe os dados
	3 – Sistema habilita a entrada dos dados do pedido
4 – Cliente informa os dados do pedido	5 – Sistema registra o pedido
	7 – Sistema emite mensagem Msg01 dizendo que o pedido foi registrado
Exceções	
2 – Cliente não cadastrado 2.1 – Sistema emite Msg01 dizendo que o cliente não está cadastrado 2.2 – Sistema habilita o cadastro do cliente **5 – Produtos solicitados inexistentes** 5.1 – Sistema emite Msg01 dizendo que existem produtos não cadastrados 5.2 – Sistema devolve o pedido ao cliente 5.3 – Fim do caso de uso	
Pós-Condições	O pedido está registrado e o cliente também.

Quadro 3 - Caso de uso - Registrar Pedido

Em concorrência com a identificação dos requisitos também são realizados a definição do escopo técnico da aplicação, esta atividade defini a visão técnica do sistema que suporta os requisitos do sistema. Durante a atividade o arquiteto de *software* do projeto é focado em:

- Trabalhar com a equipe para criar um protótipo para a proposta do sistema;
- Assegura que todas as decisões técnicas sejam comunicadas a todos;
- Assegura que toda a equipe possua informações suficientes para entender a arquitetura do sistema.

A especificação do escopo técnico abrange criar a arquitetura para um *software* funcional, e a definição do mesmo deve estar sempre de acordo com os membros da equipe, afinal, todos podem opinar em melhorias. Se a equipe de desenvolvimento já desenvolveu uma solução equivalente anteriormente, é possível utilizar o conhecimento já empregado na solução anterior.

Além disso, a definição do escopo técnico mostra ao cliente a viabilidade do projeto, o que permite o projeto ser cancelado ou prosseguir com o aval do mesmo.

5.3 DESENVOLVIMENTO DA ARQUITETURA E DA SOLUCAÇÃO TÉCNICA

O desenvolvimento da arquitetura e da solução técnica emprega todo o conhecimento entregue pela equipe na definição do escopo do projeto a fim de tornar o projeto em um produto utilizável e estável. Os arquitetos de *software* e os desenvolvedores participam ativamente juntos para:

- Refinar o protótipo inicial da arquitetura e um projeto estável e concreto;
- Garantir que as decisões de equipe e arquitetura sejam transmitidas e aplicadas na solução;
- Garantir que a equipe tenha o conhecimento para produzir os casos de uso e pedaços de código utilizáveis pelo cliente;
- Garantir que os requisitos priorizados durante a iteração sejam entregues.

A comunicação da equipe é sempre interativa e a mesma sempre foca em não desenvolver todo o *software* de uma única vez, mas sim, desenvolver o projeto em pequenos pedaços entregáveis e testáveis pelo cliente. Durante o desenvolvimento a equipe gera inúmeros artefatos de *software*, como códigos

fontes, manuais, e arquivos de configuração, todos esses artefatos são o resultado do trabalho da equipe em concluir os planos de iteração.

Quando um requisito é escolhido para ser desenvolvido na iteração, a equipe deve definir a abrangência do requisito, por exemplo, a quebra do requisito em micro-incrementos focando a cada micro-incremento desenvolver primeiramente cada separação lógica em camadas da aplicação (Interfaces de Usuário, negócios, modelo). O desenvolvimento também pode ser quebrado em componentes, ficando assim a critério dos desenvolvedores agruparem os requisitos em componentes.

Depois de realizada a implementação o desenvolvedor tem como foco criar os testes unitários do componente, estes testes não são aplicados pela equipe de testes, pois possuem uma abrangência na parte de baixo nível da aplicação agindo diretamente no código fonte da mesma. Mesmo neste ponto do projeto é necessário que as alterações realizadas sejam avaliadas e concordadas entre as partes, algumas decisões triviais refletem diretamente na existência na implementação.

Com a definição do escopo pronta e de acordada com os desenvolvedores é utilizada a melhor prática do (Test Driven Design - TDD), onde o desenvolvedor define os testes a serem realizados para verificar a implementação antes mesmo de a mesma existir. Esta abordagem garante que o *design* da implementação seja valido somente se passar pelos testes desenvolvidos anteriormente, teste que falham levam a refatoração do código fonte até que todos os testes passem de acordo com necessário.

Está prática funciona melhor em conjunto com a prática do *design* evolucionário (*Evolutionary Design*), o conjunto de ambas as práticas focam em criar uma pequena implementação para uma parte do caso de uso, criar os testes para esta pequena parte e após isto avaliar o desempenho e qualidade do mesmo, desta maneira em pequenas partes atingem o caso de uso como um todo passando em todos os testes necessários.

Também é aplicável a prática de integração contínua (*Continous Integration*), segue a linha em que o esforço para integrar o atual estado da aplicação em um ambiente de teste deve ser o menor possível, integrando o *build* de *software* frequentemente é possível identificar erros e corrigi-los de maneira mais rápida enquanto a equipe ainda possui sua mente focada no problema. Segundo ESSENTIALS (2011), está é uma lista dos benefícios:

- *Feedback* aprimorado;

- Melhor detecção de erros;
- Maior colaboração entre cliente e empresa;
- Melhor integração do sistema com o ambiente de produção
- Reduz o número de alterações paralelas;
- Reduz o número de erros;
- Reduz risco técnico;
- Reduz riscos de gerenciamento do projeto;

A integração continua funciona ainda melhor utilizando algum sistema de informação para automatizar os processos de compilação e disponibilização da versão, uma das ferramentas que auxiliam esse processo é o *JenkinsCI*[7].

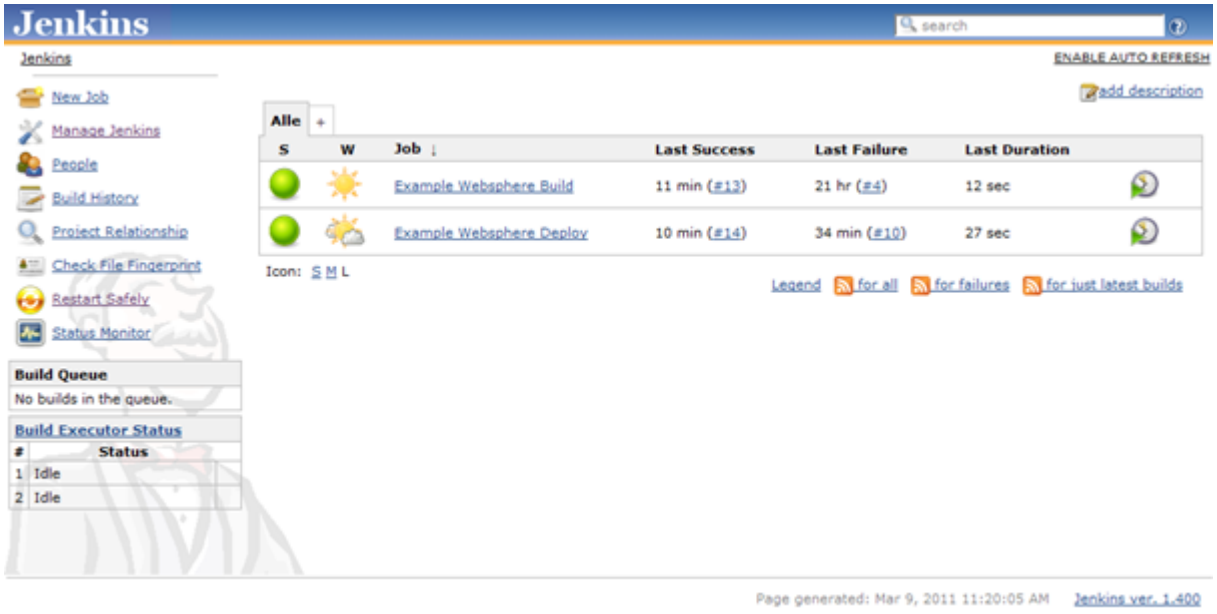


Figura 20 - Screenshot da ferramenta JenkinsCI

[7] Jenkins é um ferramenta que monitora a execução de trabalho repetitivo, como a compilação de software ou execução de teste pré-definidos. (JENKINS, 2011)

5.4 TESTES DA SOLUÇÃO

Os testes da solução são realizados em todas as fases do projeto, a cada versão entregue são realizados teste para verificar se a solução satisfaz os requisitos alocados na última iteração.

Durante a execução das iterações, a ideia é validar os itens já implementados refletindo em uma arquitetura de *software* robusta, e garantir que os requisitos faltantes sejam desenvolvidos sobre a mesma arquitetura consistente.

Assim que os desenvolvedores desenvolvem os casos de uso e criar os testes unitários para os mesmos, os testadores entram em ação criando testes em nível de utilização do sistema em paralelo com o desenvolvimento. Estes testes são constantemente integrados e garante à especificação do caso de teste no script de teste (Quadro 4), o script de teste define quais técnicas, dados, e quais suítes de testes são necessários para a criação do caso de teste com sucesso.

Nome da Propriedade	Breve Descrição
Nome	SCR-Realizar Venda.
Descrição	Este script testa as ações realizadas por um vendedor no PDV.
Finalidade	Validar as ações que um vendedor pode realizar e garantir a boa execução do caso de uso.
Itens de Teste e Avaliação Dependentes	Caso de Uso 01 – Manter Venda; Caso de Uso 02 – Manter Cliente; Caso de Uso 03 – Manter Produto.
Precondições	Os clientes devem estar cadastrados no sistema. Os produtos devem estar em uma lista disponível de fornecedor.
Instruções	-
Pontos de Observação	Será observado o tempo de resposta do sistema em situações normais e anormais, e em caso de situações anormais será analisado o comportamento do sistema como um todo.
Pontos de Controle	Mensagem resultante do sistema “Venda realizada com sucesso”. Mensagem resultante do sistema “Não é possível realizar está operação”
Pós-condições	O registro de venda deverá estar cadastrado no sistema e a fatura deverá ser gerado para o cliente.

Quadro 4 - Script de testes, realizar venda

Com a execução dos testes, os defeitos são encontrados e adicionados à lista de item a serem desenvolvidos na próxima iteração, assim é focada a entrega de dados consistentes o mais cedo possível.

Os dados de testes gerados são salvos em um arquivo de log, este arquivo captura qualquer tipo de informação que mostra o estado do último teste executado. Uma maneira de gerenciar scripts e logs de teste é utilizando uma ferramentas chamada *TestLink* para a automatização deste processos.

A ferramenta *TestLink* (Figura 21) é uma ferramenta *WEB* de gerenciamento de testes, ele gerencia testes em níveis de projeto, níveis de execução e de usuários que estão executando os teste. A ferramenta é *open-source* e disponibilizada livremente na internet.

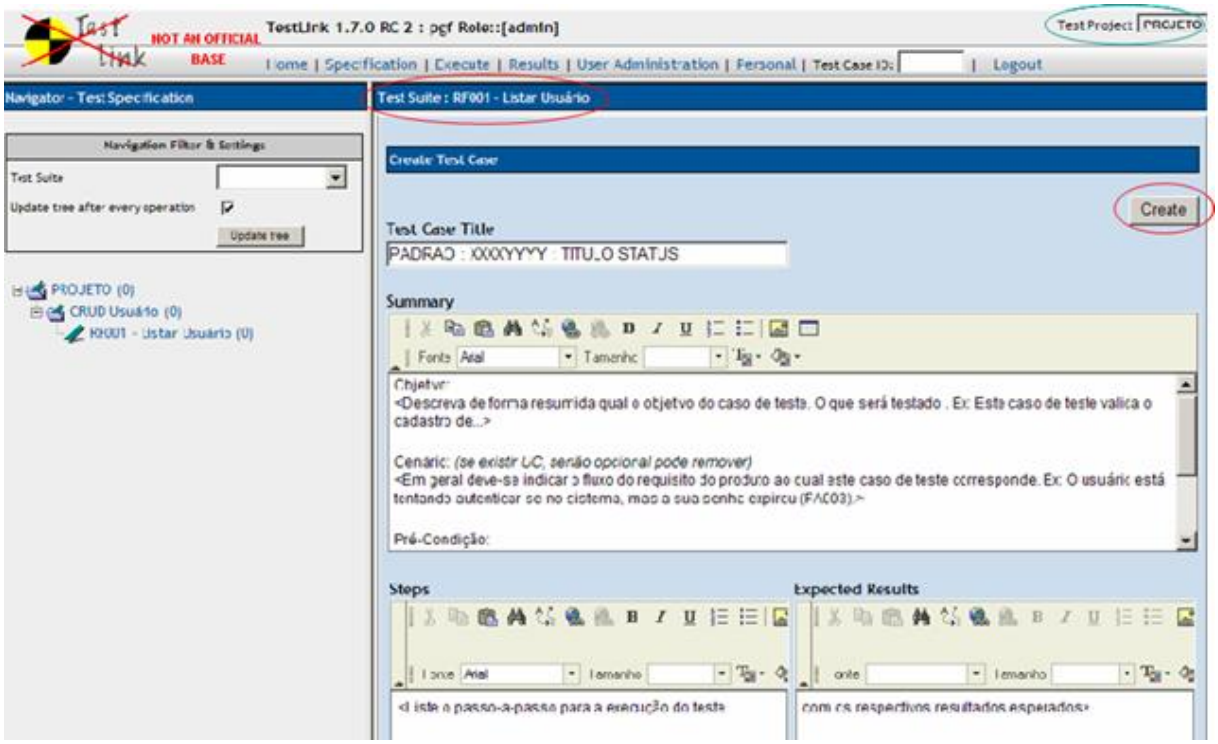


Figura 21 - Print ferramenta de testes TestLink

5.5 *OPENUP* X MODELO CASCATA

A metodologia *Openup* possui alguns pontos em comum com o modelo em cascata, porém mesmo esses pontos em comum funcionam melhor que o esperado no modelo em cascata, uma das razões para este fato é que o *Openup* possui em sua filosofia a entrega ágil do produto.

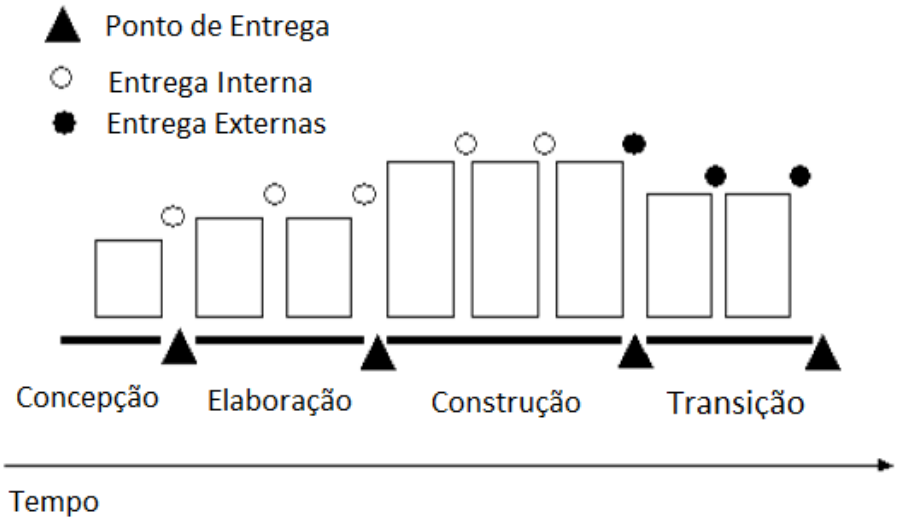


Figura 22 - Ciclo de vida com o *Openup*
Fonte: (KRUCHTEN, 204)

Sem dúvida um dos pontos mais fortes do *Openup* sobre o modelo em cascata é o seu estilo de ciclo de vida iterativo incremental juntamente com a prática de integração contínua (**Erro! Fonte de referência não encontrada.**), assim a cada teração o próprio cliente recebe automaticamente uma nova versão para testes, autorizando ou não a conclusão do requisito.

Este tipo de função não é aplicável ao modelo cascata, pois ele possui um foco em realizar a entrega do produto inteiro somente no final do projeto, assim, se o cliente não gostar de alguma funcionalidade acaba causando um maior risco na alteração de códigos, e até mesmo a estouro do prazo de entrega do projeto.

Está entrega de parte do produto fornecido pelo *Openup* permite que o projeto ao seu fim seja completo com qualidade e em acordo com o cliente, além disso, alterações feitas assim que erros são identificados são mais fáceis de serem realizados durante o desenvolvimento da funcionalidade do que na entrega final do projeto. Assim, ao identificar erros no começo é possível mitigar erros do futuro do projeto.

6 CONSIDERAÇÕES FINAIS

Esta pesquisa teve como principal objetivo explicar uma parcela dos processos da metodologia para desenvolvimento ágil de *software Openup*. A metodologia consegue garantir um processo de desenvolvimento robusto como o RUP com a simplicidade e foco como o *SCRUM*.

Através de um estudo de caso hipotético é possível perceber que a integração com os processos de desenvolvimento é fácil de ser realizado, isto demonstra que com a tecnologia disponibilizada torna-se possível criar uma aplicação robusta com menor tempo de desenvolvimento e com mais qualidade. Além disso, sua arquitetura modular propõe um novo nível de separação dos processos de maneira intuitiva, assim, tornando possível a adaptação de novos processos de acordo com o funcionamento de cada empresa.

Com certeza o *Openup* é uma metodologia que pode ser utilizada para o desenvolvimento de qualquer tipo de aplicação utilizando quaisquer tipos de recursos, sua integração, robustez e simplicidade, fazem da metodologia preparada para o desenvolvimento eficaz e com uma maior agilidade.

6.1 TRABALHOS FUTUROS/CONTINUAÇÃO DO TRABALHO

Um possibilidade interessante para estudos futuros é a aplicabilidade da metodologia *Openup* com outra metodologia de gerenciamento de projeto visando melhores os processos de maneira mais robusta , o PMBOOK.

Para colocar a aplicabilidade deste estudo, é interessante integrar esta metodologia em um projeto real, visando focar em todos os itens de trabalho gerados e recursos utilizados.

O desenvolvimento de uma aplicação WEB para a automatização do processos e do gerenciamento dos artefatos gerados focando no *Openup*, basicamente seria uma ferramenta de gerencia de projeto com foco na metodologia *Openup*.

REFERÊNCIAS

BALDUINO, R. Introduction to Open UP (Open Unified Process). **Eclipse Foudation**, 2007. Disponivel em: <http://www.eclipse.org/epf/general/*Openup*.pdf>. Acesso em: 10 nov. 2011.

BAUER, F., 1969.

BROOKS, F. No Silver Bullet. **Essence and Accidents of *Software* Engineering**, Chapel Hill, abr. 1987.

CAETANO, R. Metologias de Desenvolvimento. **Computerworld**, 2009. Disponivel em: <http://computerworld.uol.com.br/gestao/2009/08/05/metodologias-de-desenvolvimento-qual-a-mais-adequada/>. Acesso em: 4 out. 2011.

DISTILIED, O. *Openup*. **Eclipse**, 2011. Disponivel em: <http://epf.eclipse.org/wikis/*openup*/index.htm>. Acesso em: 11 out. 2011.

DONATO, R. Fábrica de *Software* – Definir processo de desenvolvimento. **Caverna do *Software***, 2010. Disponivel em: <http://voat.com.br/rdal/?tag=incremental>. Acesso em: 11 out. 2011.

ECLIPSE. Description. **Eclipse Process Framework**, 2011. Disponivel em: <http://www.eclipse.org/epf/general/description.php>. Acesso em: out. 2011.

EPF. **Eclipse Process Framework Project**, 2011. Disponivel em: <http://www.eclipse.org/projects/project_summary.php?projectid=technology.epf>. Acesso em: 31 ago. 2011.

ESSENTIALS, O. Eclipse. **Getting Started**, 2011. Disponivel em: <http://www.eclipse.org/epf/general/*openup*_essentials.zip>. Acesso em: 18 set. 2011.

FOWLER, M. **The New Methodology**, 2003. Disponivel em: <http://martinfowler.com/articles/newMethodology.html>. Acesso em: 2 ago. 2011.

FUGGETTA, A. ***Software* Process: A Roadmap**. [S.l.]. 2000.

GUSTAFSSON, B. *Openup* –The Best of Two Worlds. **Methods and Tools**, 2008. Disponivel em: <http://www.methodsandtools.com/archive/archive.php?id=69>. Acesso em: 11 out. 2011.

HAZRATI, V. Analisando Gráficos de Burndown. **InfoQ**, 2010. Disponivel em: <http://www.infoq.com/br/news/2010/02/deciphering-burndown-charts>. Acesso em: 10 nov. 2011.

IFSC, A. Ciclo de Vida Iterativo e Incremental. **IF-SC**, 2009. Disponivel em: <http://wiki.sj.ifsc.edu.br/wiki/index.php/Ciclo_de_Vida_Iterativo_e_Incremental>. Acesso em: 10 nov. 2011.

JENKINS. Meet Jenkins. **Jenkins CI**, 2011. Disponivel em: <https://wiki.jenkins-ci.org/display/JENKINS/Meet+Jenkins>. Acesso em: 01 dez. 2011.

KROLL, P. IBM. ***Openup*** **In a Nutshell**, 15 set. 2007. Disponivel em: <http://www.ibm.com/developerworks/rational/library/sep07/kroll/index.html>. Acesso em: 31 ago. 2011.

KROLL, P. Who will benefit from the Eclipse Process, out. 2011. Disponivel em: <http://www.eclipse.org/proposals/beacon/Who%20will%20benefit%20from%20Eclipse%20Process%20Framework.pdf>. Acesso em: 18 out. 2011.

KROLL, P.; LYONS, B. ***Openup*** **Distilled**, 2010. Disponivel em: <http://www.eclipse.org/epf/community/Open%20Unified%20Process%20Distilled%20by%20Kroll%20and%20Lyons.ppt>. Acesso em: 18 out. 2011.

KRUCHTEN, P. Going Over the Waterfall with the RUP. **Developer Works**, 204. Disponivel em: <http://www.ibm.com/developerworks/rational/library/4626.html>. Acesso em: 11 out. 2011.

MANIFESTO ÁGIL. **Agile Manifesto**, 31 ago. 2001. Disponivel em: <http://agilemanifesto.org/iso/ptbr/>. Acesso em: 31 ago. 2011.

MEIRA, F. L. OPEN UP - PROCESSO UNIFICADO ABERTO. **Conceitos Básicos do Open UP**, 2010. Disponivel em: <http://open2up.blogspot.com/p/principios-basicos-do-open-up.html>. Acesso em: 13 nov. 2011.

PRESSMAN, R. S. **Engenharia de *Software***. 6. ed. [S.l.]: McGraw-Hill, 2006.

RODRIGUES, L. D. S. *Scrum* é um framework de gerenciamento de projetos. **Massimus**, 2011. Disponivel em: <http://massimus.com/lang/pt-br/archives/458>. Acesso em: 13 dez. 2011.

ROYCE, W. ***Software*** **Project Management**. [S.l.]: The Addison, 2000.

SCHWABER, K.; SUTHERLAND, J. *Scrum*: The Definitive Guide. ***Scrum***, 2011. Disponivel em: <http://www.*scrum*.org/storage/*Scrum*%20Guide%202011%20-%20PTBR.pdf>. Acesso em: 11 out. 2011.

SOARES, S. C. B. Processo Iterativo Incremental. **UFPE**, 2010. Disponivel em: <www.cin.ufpe.br/~phmb/RUP/./ProcessoIterativoIncremental.ppt>. Acesso em: 10 dez. 2011.

SOMMERVILLE, I. ***Software*** **Engeneering**. 8. ed. [S.l.]: [s.n.], 2006.

WTHREEX. **Rational Unified Process**, 2002. Disponivel em: <http://www.wthreex.com/rup/portugues/index.htm>. Acesso em: 31 ago. 2011.

APÊNDICES

APÊNDICE A - Melhores práticas da primeira premissa do *Openup*

PRÁTICA	DESCRIÇÃO
Conheça sua audiência	É impossível realizar negociações se não temos conhecimentos sobre a pessoa que está comprando o produto de *software*, por isso é muito importante que além de você conhecer bem o *stakeholder* é necessário estar sempre presente para realmente suas necessidades. Para isto é necessário identificar todos os *Stakeholders* e manter comunicações frequentes.
Separe o problema da solução	É necessário ter um amplo conhecimento do problema, saber como e onde está acontecendo para depois saber o porquê e como achar uma solução para o mesmo, o que torna mais fácil achar o foco do problema.
Documente e compartilhe a compreensão do domínio da solução	É importante saber como compartilhar o conhecimento sobre o domínio do projeto, por exemplo, todo o conhecimento poderia ser disponibilizado em forma de um documento interno na empresa ou em um site interno que siga o padrão da Wikipédia.
Utilize casos de uso para especificar os requisitos	A utilização de casos de uso facilita a compreensão dos requisitos pelos desenvolvedores, eles compreendem como uma ação que o usuário irá realizar no sistema. Os requisitos não funcionais também devem ser bem estabelecidos em um documento separado utilizando outro padrão.
Estabelecer e manter a concordância em relação às prioridades	É necessário que sejam escolhidos os casos de uso que tenham maior prioridade, além disso, é necessário gerenciar as propriedades afim de entregar maior valor e diminuir os riscos, por isto o contato com os *Stakeholders* são imprescindíveis.
Faça trocas para maximizar o valor	Esta prática implica na prática citada anteriormente, é necessário que sejam realizadas trocas nas prioridades dos casos de uso focando em sempre entregar a cada iteração um produto utilizável com maior valor e com qualidade.

Gerenciar o escopo	O gerente de projeto deve saber como gerenciar o escopo geral do projeto, durante a execução do projeto pode ser que surgem novas necessidades por conta do *stakeholder*, porém cabe ao gerente estabelecer se esta nova funcionalidade não implicará em outras funcionalidades ou no atraso do projeto.
Saiba quando parar	Ter o conhecimento para dizer que um caso de uso está em um nível aceitável, saber quando para de trabalhar na arquitetura do projeto quando o mesmo já possui uma arquitetura de níveis aceitáveis ou quando o problema está acontecendo saiba que arrumou e não precise mais se preocupar com o mesmo.

Quadro 5 - Melhores práticas primeira premissa *Openup*

APÊNDICE B - Melhores práticas da segunda premissa do *Openup*

PRÁTICA	DESCRIÇÃO
Manter uma visão compartilhada dos conhecimentos	É importante que os membros da equipe sejam proativos em compartilhar informações, pode ser necessário utilizar item de trabalho para compartilhar as informações como, por exemplo, um documento de casos de usos.
Promover um ambiente de autoconfiança	A fim de conseguir um ambiente sustentável, é necessário que o gerente e os outros membros devam ser auto gerenciáveis, além disso, todos devem entender a cultura organizacional da empresa.
Compartilhe responsabilidades	Compartilhando responsabilidades atribuem ações para alguns membros específicos realizarem, porém, mesmo sendo responsabilidade de outro participante da equipe é necessário que membros mais experientes auxiliem os menos experientes.
Aprenda continuadamente	Aprenda com erros dos companheiros, aprender com os sucessos dos outros participantes, ter a habilidade de ensinar os conhecimentos obtidos e saber resolver dúvidas dos outros.
Gerenciar em torno da arquitetura	É importante que a equipe seja centrada ao redor da arquitetura e das suas necessidades, o importante é que se houverem necessidades a equipe deve saber como resolvê-las e a organização da equipe não pode fugir do escopo da arquitetura.

Quadro 6 - Melhores práticas segunda premissa *Openup*

APÊNDICE C - Melhores práticas da terceira premissa do *Openup*.

PRÁTICA	DESCRIÇÃO
Elabore uma arquitetura considerando seus conhecimentos	Deve-se criar uma arquitetura que supra as necessidades reais do *stakeholder*, esta arquitetura deve ser fácil de ser expandida, porém não pode ser muito simples, deve ser empregado todo o conhecimento que é possuído até então. A medida que o projeto evolua também deve evoluir a sua arquitetura.
Influência da arquitetura como uma ferramenta colaborativa	A arquitetura deve servir como um ponto de discussão entre os desenvolvedores, uma vez que os desenvolvedores não possuem as mesmas opiniões é possível que hajam ideias que melhorariam a arquitetura, mas, os desenvolvedores sempre devem respeitar a arquitetura como um ponto de referência.
Tratar a complexidade do sistema incrementando o nível de abstração do projeto	Para aumentar o nível de abstração dos processos e ações realizadas no projeto podem-se criar modelos referenciais a serem seguidos, desta forma os membros da equipe sempre possuem uma base para começar.
Organize a arquitetura em componentes coesos e fracamente acoplados	Uma arquitetura coesa e fracamente acopla é fácil de ser reutilizada e é muito flexível, uma arquitetura assim economiza no tempo de reuso e no caso de pequenas alterações as mesmas são realizadas sem muita burocracia.
Reutilize componentes existentes	Esta prática implica diretamente com a prática anterior, quando se possui uma arquitetura coesa é possível muitas vezes reutilizar componentes de outras aplicações o que diminui os custos gerais do projeto.

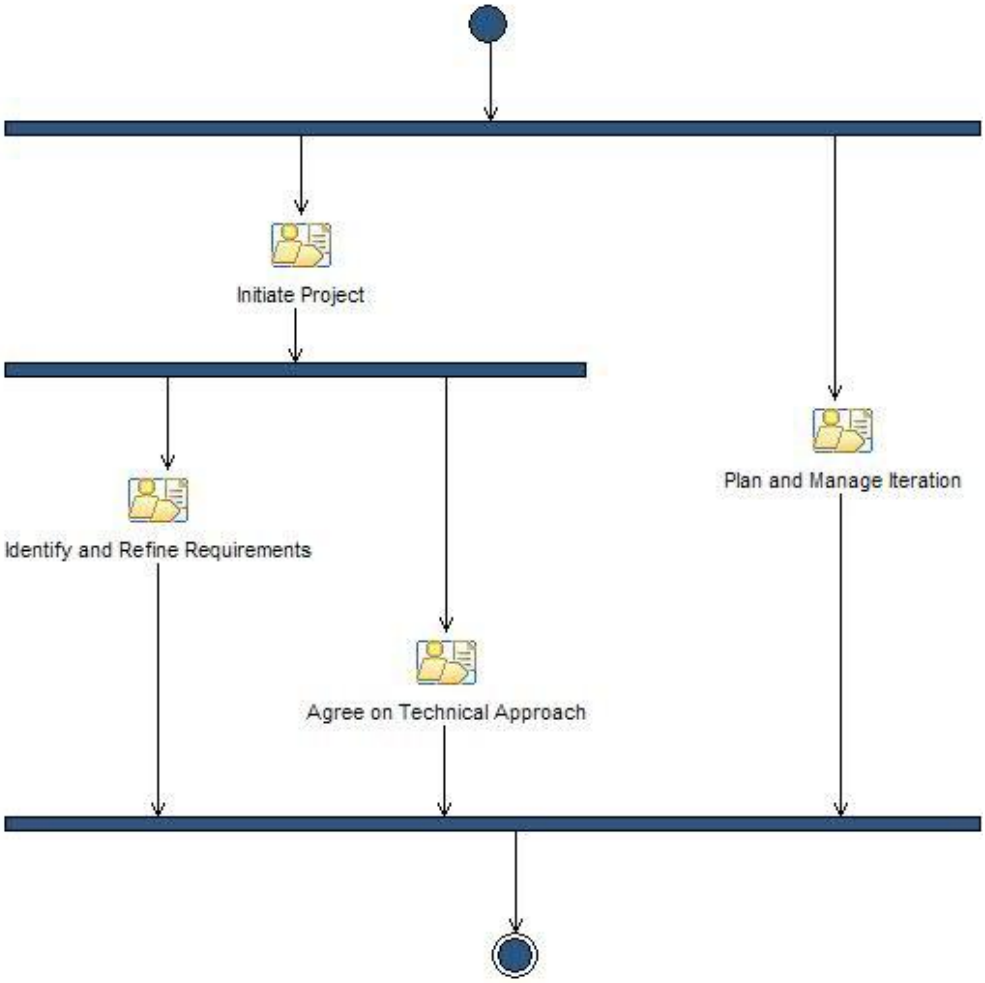
Quadro 7 - Melhores práticas teceira premissa *Openup*

APÊNDICE D - Melhores práticas da quarta premissa do *Openup*

PRÁTICA	DESCRIÇÃO
Desenvolva seus projetos de forma iterativa	Desenvolver *software* de iterativa fornece à equipe a chance de constantemente evoluir o *software*, outra função do desenvolvimento iterativo é identificar erros de análise ou arquitetura as quais possam ser concertadas antes do final do projeto.
Foque as iterações de forma a atingir os próximos marcos	Um marco pode ser um conteúdo entregue, ou seja, foca as iterações focando no resultando que são as entregas de produtos utilizáveis e funcionais pelo *stakeholder*.
Gerencie riscos	É necessário certo manuseio em relação aos riscos encontrados, se percebemos um risco cedo e é tentada uma atualização para sanar o risco pode ser que o risco se torne ainda maior, porém, certos tipos de riscos devem ser tratados o quanto antes, como por exemplo, um risco detectado na arquitetura do projeto.
Gerencie mudanças	Mudanças no decorrer do projeto são inevitáveis, porém, as mudanças devem ser controladas e documentadas e o *stakeholder* responsável pela requisição da mudança deve estar ciente dos custos de tempo e esforço que a mudança acarretara no projeto.
Mensure o progresso do projeto de forma objetiva	Deve-se mensurar o progresso do projeto sempre que possível, ao fim das iterações deve ser verificados se os objetivos foram atingidos e se através da última iteração é possível aplicar melhorias para as próximas iterações. Ao final de casa iteração e microincremento pode ser feitos pequenas reuniões de alinhamento do status do projeto, e toda a atualização de status devem ser preenchidos em algum documento de comum acesso aos membros da equipe.

Quadro 8 - Melhores práticas quarta premissa *Openup*

APÊNDICE E – Fluxo analítico das atividades da fase de concepção.

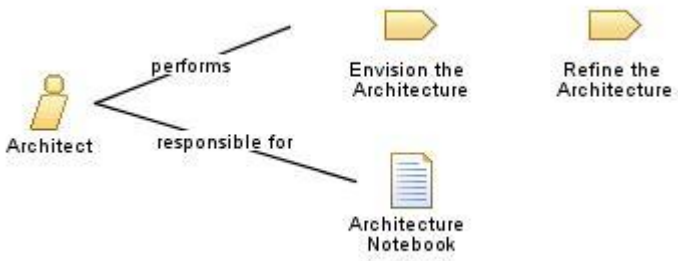


APÊNDICE F – Relação entre grupo de trabalho e ações realizadas

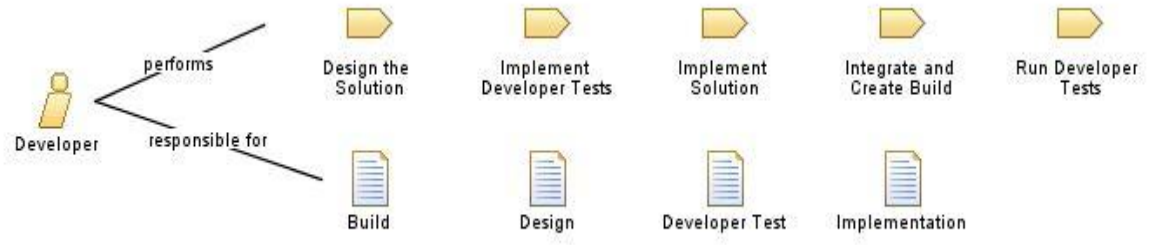
ANALISTA

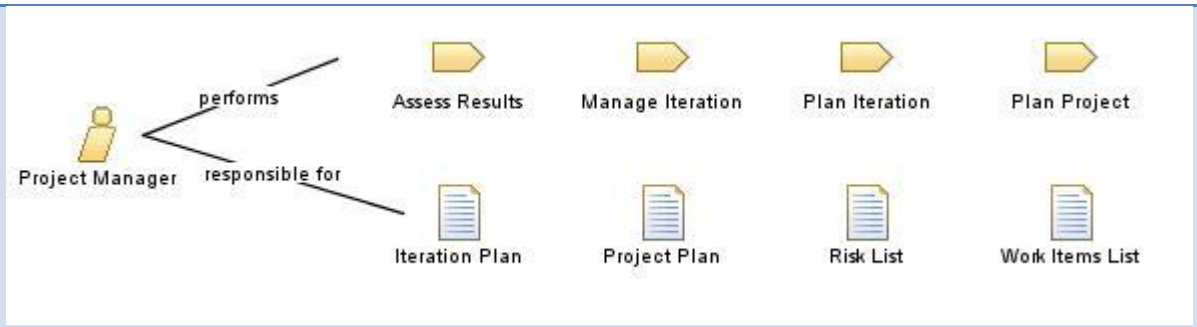
REALIZAÇÕES E PARTICIPAÇÕES	TRABALHA OU ESTÁ PRESENTE EM
Avaliar Resultados	Glossário
Criar Casos de Teste	Requisitos gerais do sistema
Design da Solução	Casos uso
Visualiza a Arquitetura	Modelo de Casos de Uso
Implementa Testes	Documento de Visão
Gerenciar Iteração	Lista de itens de trabalho
Plano de Iteração	
Plano de projeto	

ARQUITETO



REALIZAÇÕES E PARTICIPAÇÕES	TRABALHA OU ESTÁ PRESENTE EM
Avaliar os resultados	**Documento de Arquitetura**
Design da solução	
Detalha os requisitos gerais	

Detalha de Casos de Uso e Cenários Desenvolver Visão Técnica Identificar e Esboçar Requisitos Gerenciar Iteração Plano de Iteração Plano de Projeto	
DESENVOLVEDOR	
	
REALIZAÇÕES E PARTICIPAÇÕES	**TRABALHA OU ESTÁ PRESENTE EM**
Avaliar os resultados Criar Casos de Teste System-Wide detalhes Requisitos Detalhe de Casos de Uso Identificar e Esboçar Requisitos Implementar testes Gerenciar Iteração Plano de Iteração Plano de Projeto Refinar a Arquitetura	Build estável Design Desenvolvimento de testes Implementação Cria o log de testes
GERENTE DE PROJETO	

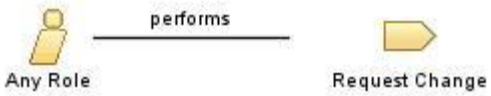


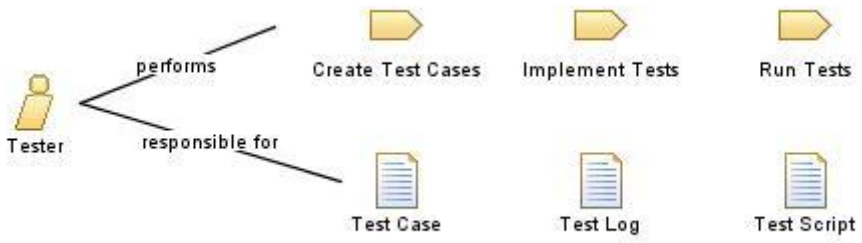
REALIZAÇÕES E PARTICIPAÇÕES	TRABALHA OU ESTÁ PRESENTE EM
Desenvolve a visão técnica	Plano de Iteração
Apontar os caminhos da Arquitetura	Plano do Projeto
Refinar a Arquitetura	Lista de Riscos
	Lista de itens de trabalho

STAKEHOLDER

REALIZAÇÕES E PARTICIPAÇÕES	
Avaliar os resultados	Identificar e Esboçar Requisitos
Elicitar casos de Teste	Implementar solução
Design da solução	Implementar testes
Detalhar os requisitos	Gerenciar Iteração
Detalhe de Casos de Uso Cenários	Iteração plano
Desenvolver Visão Técnica	Plano de Projeto

QUALQUER OUTRO GRUPO



REALIZAÇÕES E PARTICIPAÇÕES
Ajudar na lista de Itens de Trabalho Fazer café Etc..
TESTADOR


REALIZAÇÕES E PARTICIPAÇÕES	TRABALHA OU ESTÁ PRESENTE EM
Avaliar os resultados Design da solução Requisitos gerais da aplicação Detalhe de Casos de Uso e Cenários Identificar e Esboçar Requisitos Implementar testes de desenvolvedor Implementar solução Gerenciar Iteração Iteração plano Plano de Projeto	Caso de Teste Registros de Teste Scripts de Teste

APÊNDICE G – Documento de Visão Distri+

Visão

1. Introdução

O sistema será de uma distribuidora de produtos para a empresa DistriMais, este sistema terá como foco principal receber pedidos dos clientes que desejam comprar produtos, administrar o cadastro de produtos e fornecedores.

2. Declaração do problema

O Problema de	Gerenciar vendas e distribuição de produtos para o cliente.
Afetados	Todos os funcionários da empresa são afetados por falta de uma gerencia de vendas.
Impacto	Déficit na distribuição dos produtos e os resultados das vendas.
Solução de sucesso	Criar uma ferramenta capaz de gerenciar os produtos, gerenciar vendas e entregas e as escolhas de compras dos clientes.

3. Declaração de posição do produto

Para	DistriMais – Outras distribuidoras
Quem	Sistemas de gerenciamento de distribuidoras são ineficazes e lentos, além disso, não abrangem todas as necessidades dos nossos clientes.
Oque	A ferramenta Distri+ é um *software* completo e customizável de acordo com a necessidade de cada cliente, além disso, é leve e robusto funcionando com poucos recursos de informática.

Diferença	Sistema com processos de gerenciamento de estoque dinâmico.

4. Descrições do Stakeholder

Nome	Descrição	Responsabilidades
João	Vendedor	Realiza as vendas na frente do caixa, responsável por utilizar o PDV (Ponto de venda). Pessoa que tem contato direto com o cliente.
Maria	Gerente de Vendas	Gerencia as vendas, promove a equipe de vendas, autoriza vendas de grande porte, emite requisição de produtos para o estoque. Gerencia as faturas.
Joselito, Dagoberto	Gerente de Estoque	Controla todo o estoque, e contata fornecedores.

5. Ambiente do Usuário

A distribuidora de produtos recebe pedidos dos clientes. O pedido é aceito se o cliente e os produtos estiverem previamente cadastrados. Caso o cliente não esteja cadastrado, seu cadastro deve ser realizado previamente. Caso nenhum dos produtos pedidos exista, o pedido é devolvido ao cliente.

Ao final da semana, a distribuidora emite requisições de produtos para os fornecedores com base nos pedidos recebidos.

Quando os produtos são fornecidos, a distribuidora confere a nota de entrega do fornecedor com a requisição, devolve as que estiverem com erros e atende aos pedidos dos clientes, emitindo as respectivas faturas.

As faturas dos clientes devem ser pagas até a data do seu vencimento. As faturas dos fornecedores são controladas por outro sistema.

Periodicamente, o fornecedor atualiza seus produtos. A distribuidora envia catálogo de produtos aos clientes ao fim de cada mês.

6. Necessidades

Necessidade	Prioridade	Requisito	Data de previsão de Teste
Administrar Vendas	10	Manter Vendas	15/07/2011
Gerenciar Faturas	8	Consultar Fatura	30/07/2011
Administrar fornecedores	6	Manter fornecedor	15/08/2011
Gerenciar requisições de Produtos	9	Manter Requisição	20/08/2011
Gerenciar clientes	10	Manter Cliente	30/09/2012

7. Outros requisitos

Requisito	Prioridade
O sistema deve ser complemente funcional nos fins de semana e durante o horário comercial	10
O sistema deve ter uma interface intuitiva e simples de ser utilizada	7
O sistema deve alarmar o vendedor sobre faturas de um cliente que está realizando uma requisição.	4

APÊNDICE H – Plano de Projeto Distri+

Plano de Projeto

1. INTRODUÇÃO

Este documento possui as informações de projeto do sistema Distri+. Neste plano são encontrados dados fundamentais durante a utilização do mesmo, práticas utilizadas e as lições aprendidas.

2. ORGANIZAÇÃO DO PROJETO

A equipe de projeto está sob a gerência do João Calor Manuel (Gerente de Projetos), o mesmo levantou a equipe de desenvolvedores (José Calor Manuel e Maria Calor Manuel). O responsável por realizar o teste é o Thiago Kinamutuamutu, todos os membros foram escolhidos de acordo com suas especializações e méritos de outros projetos.

A ponte de comunicação entre os desenvolvedores e os *stakeholders* é feita pelo Sr. Joao Calor Manuel, porém os desenvolvedores devem possuir os e-mails diretos para dúvidas simples.

3. PRÁTICAS DE PROJETO E MENSURAS

Durante a execução do projeto serão as práticas de desenvolvimento iterativo, integração continua TDD (Test-driven Development). Para realizar a mensura do estado do projeto e entregas serão realizados gráficos no estilo *burndown* o qual será fornecedo para todos os membros da equipe.

4. ENTREGAS E OBJETIVOS

Iteração	Objetivo Primários	Data de Entrega	Velocidade
I1	Definição de Arquitetura base	15/05/2011 – 25/05/2011	15
I2	Entrega do Caso de Uso 01	26/06 – 10/07	16
I3	Entrega do Caso de Uso 02	11/-7 – 13/07	20
I4	Entrega do Caso de Uso 03	15/07 – 30/07	10
I5	Entrega do Caso de Uso 04	02/08 – 15/08	11
I6	Entrega do Caso de Uso 05	18/08 – 25/09	5
I7	Entrega do Caso de Uso 08	27/09 – 10/10	10
I8	Entrega do Caso de Uso 06	12/10 – 29/10	5
I9	Entrega do Caso de Uso 07	01/11 – 05/11	17

5. TÉCNICA DE TRANSIÇÃO

A técnica utilizada para realizar a transição das builds será a utilização de uma ferramenta de gerenciamento de integração continua, a ferramenta é chamada Hudson. Está automatizado através da ferramenta toda quarta-feira ás 20:00, a execução e compilação do código fonte encontrado no repositório.

Caso haja a falha durante a compilação ou execução dos teste será enviado um e-mail ao membro da equipe de desenvolvimento, se não, será enviando um e-mail ao *stakeholders* para que testem a nova versão em um ambiente de testes.

APÊNDICE I – Plano de iteração do projeto

Plano de Iteração

1. Pontos chaves

Pontos	Datas
Inicio da Iteração	26/06
Demonstrar a versão DEMO	07/07
Fim da Iteração	10/07

2. Objetivos Primários

- Disponibilizar a função de registro de vendas
- Resolver problemas de usabilidade sobre o gerenciamento de vendas.
- Realizar uma entrega de uma versão demo do *software*.
- Comunicar e aguardar aprovação do *stakeholder*

3. Lista de Itens de Trabalho

Nome / Descrição	Prioridade	Estimativa (Points)	Responsável	Estimativa(hours)
Suportar um controle de vendas simples	5	8	João	48
Realizar o *design*			Carlos	12
Implementar e testar códigos do servidor (Server)			Lucas	14
Implementar e testar códigos do cliente (PDV)			Lucas	28
Atualizar documentação de usuário			Maria	6
Produzir uma versão demo e disponibilizar para stakeholders	10	5	Lucas	4
Atualizar documentação final	2	5	João	65
Atualizar manual de instalação	2	1	João	5
Atualizar notas de versão	2	1	João	4

4. Problemas

Problema	Status	Notas
Garantir processamento do PDV em menos de 30%.	OK	
Garantir boa resposta entre PDV e servidor	**OK**	**A comunicação entre PDV e servidor foi melhorada através re-estruturação da rede cabeada , as redes devem utilizar o padrão 1000Base-T.**

5. Critérios de avaliação

- O sistema deve passar em 100% dos testes de sistema.
- O stakeholder deve garantir a aprovação da função com base no demo.
- O sistema deve possuir uma boa usabilidade não ocupando 45% da capacidade de processamento da máquina e servidor.

6. Entrega

Alvo	Função de cadastro de Vendas – PDV
Data	15/06
Participantes	Utilizadores do PDV e desenvolvedores.
Status	OK

Em reunião entre os desenvolvedores e os utilizadores da empresa DistriMais, foi notado que o sistema de ponto de venda (PDV) funcionou melhor que o esperado pelos utilizadores. O sistema superou todas as expectativas e garantiu a aprovação do *stakeholder*.

APÊNDICE J – Documento de Casos de Uso

CASOS DE USO

1. Lista de Casos de Uso

Nr.	Descrição do Caso de Uso	Entrada	Caso de Uso	Resposta
01	Cliente envia solicitação de pedido	Dados Solicitação Pedido	RegistrarPedido	Msg01
02	Cliente efetua cadastro	Dados Cliente	CadastrarCliente	Msg02
03	Distribuidora requisita produtos de fornecedores	Dados Requisição Produtos	RequisitarProdutos	Msg03
04	Distribuidora atende pedido	Dados Pedido Atendido	AtenderPedido	Msg04
05	Cliente efetua pagamento	Dados Pagamento	EfetuarPagamento	Msg05
06	Distribuidora mantém produtos	Dados Produto	ManterProduto	Msg06
07	Distribuidora envia catálogo de produtos	Dados Catalogo	EnviarCatalogoProdutos	Msg07
08	Distribuidora lança fatura	Dados Fatura	LancarFatura	Msg08
09	Distribuidora confere produtos	Dados Entrega	ConferirProdutosEntrega	

Obs:

Msg01 = Pedido registrado | Cliente não cadastrado | Produtos não cadastrados

Msg02 = Cliente cadastrado

Msg03 = Requisição efetuada

Msg04 = Pedido atendido

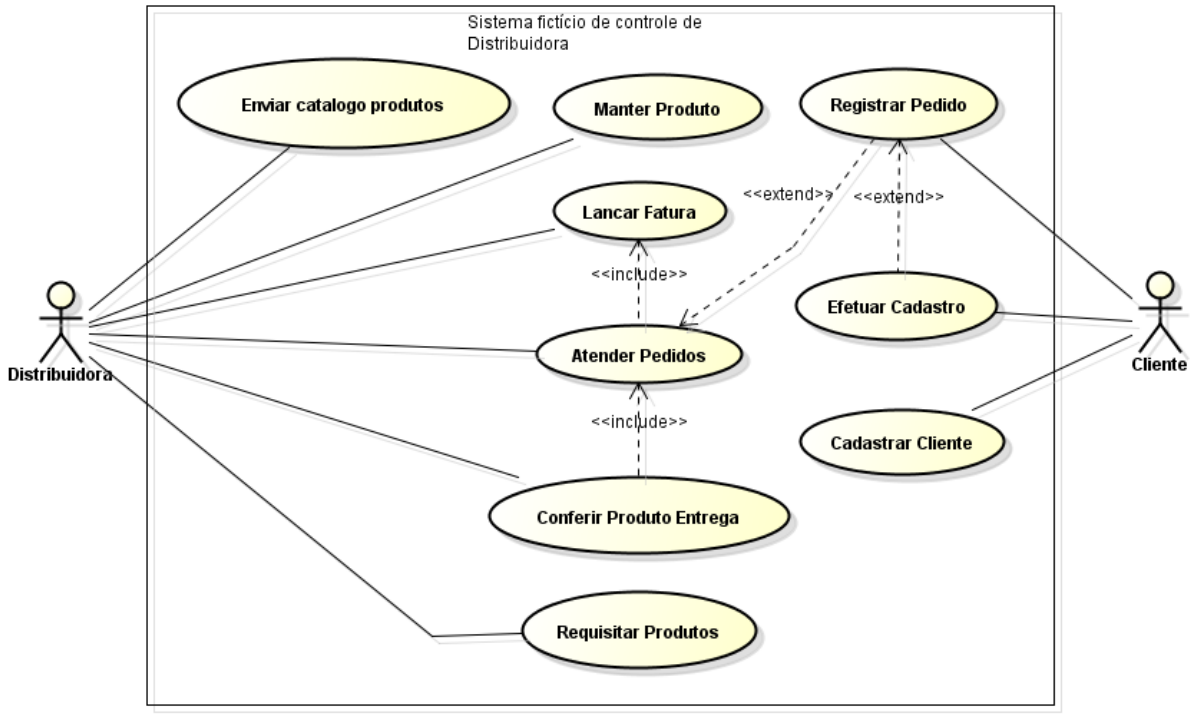
Msg05 = Pagamento efetuado

Msg06 = Produto cadastrado | Produto atualizado | Produto excluído

Msg07 = Catálogo enviado

Msg08 = Produtos conferidos | Entrega com erros

2. Diagramas de Casos de Uso



Caso de Uso	RegistrarPedido – RP1
Atores	Cliente (Iniciador)
Finalidade	Registrar um pedido feito por um cliente.
Visão Geral	O cliente solicita determinados produtos através de um pedido. Caso o cliente não esteja cadastrado ou os produtos sejam inexistentes, o pedido é descartado.
Tipo	Primário
Pré-Condições	O cliente deve estar cadastrado no sistema. Todos os produtos devem estar cadastrados no sistema.
Sequência Típica de Eventos	
Ação do Ator	*Resposta do Sistema*
1 – Cliente informa seus dados	2 – Sistema recebe os dados

	3 – Sistema habilita a entrada dos dados do pedido
4 – Cliente informa os dados do pedido	5 – Sistema registra o pedido
	7 – Sistema emite mensagem Msg01 dizendo que o pedido foi registrado
Exceções	
2 – Cliente não cadastrado 2.1 – Sistema emite Msg01 dizendo que o cliente não está cadastrado 2.2 – Sistema habilita o cadastro do cliente 5 – Produtos solicitados inexistentes 5.1 – Sistema emite Msg01 dizendo que existem produtos não cadastrados 5.2 – Sistema devolve o pedido ao cliente 5.3 – Fim do caso de uso	
Pós-Condições	O pedido está registrado e o cliente também.

Caso de Uso	CadastrarCliente – CC1
Atores	Cliente (Iniciador)
Finalidade	Efetuar o cadastro de cliente
Visão Geral	O cliente informa seus dados para efetuar o cadastro, a fim de poder realizar pedidos na distribuidora.
Tipo	Secundário
Pré-Condições	
Sequência Típica de Eventos	
Ação do Ator	*Resposta do Sistema*

1 – Cliente informa seus dados	2 – Sistema registra os dados do cliente
	3 – Sistema emite mensagem Msg02 dizendo que cliente foi cadatrado
Pós-Condições	O cliente está cadastrado e pode efetuar pedidos

Caso de Uso	RequisitarProdutos – RP1
Atores	Distribuidora (Iniciador)
Finalidade	Requisitar do fornecedor os produtos necessários para atender os pedidos
Visão Geral	Distribuidora informa os dados da requisição. O sistema registra a requisição e a envia para o fornecedor.
Tipo	Primário
Pré-Condições	
Sequência Típica de Eventos	
Ação do Ator	*Resposta do Sistema*
1 – Distribuidora informa os dados da requisição	2 – Sistema registra os dados da requisição
	3 – Sistema envia requisição ao fornecedor
	4 – Sistema emite mensagem Msg03 dizendo que requisição foi efetuada.
Pós-Condições	A requisição foi registrada e enviada ao fornecedor.

3. **Diagrama de classes**