MARC M. MARINHO DE ALCÂNTARA NETO

Processo de Desenvolvimento Iterativo

Guarulhos

2009

MARC M. MARINHO DE ALCÂNTARA

Processo de Desenvolvimento Iterativo

Trabalho de conclusão de curso apresentado à Faculdade Eniac, referente ao curso de Sistemas de Informação.

Prof. Orientador: Mauro Roberto Claro

Guarulhos

2009

de Alcântara, Marc M. Marinho

Processo de Desenvolvimento Iterativo – São Paulo, 2009.
nf. 76

Trabalho de Conclusão de Curso – Faculdade Eniac – Sistemas de Informação.

Orientador: Mauro Roberto Claro

Aluno: Marc M. Marinho de Alcântara Neto
Título: Processo de Desenvolvimento Iterativo

A banca examinadora dos Trabalhos de Conclusão em sessão pública realizada em __/__/____, considerou o (a) candidato (a):

( ) aprovado ( ) reprovado

1) Examinador(a)_____
2) Examinador(a)_____
3) Examinador(a)_____

Dedicatória

Dedico este trabalho de conclusão de curso a Edmundo Monson Tiossi Júnior, que sempre sonhou com meu título de bacharelado, e nisto muitas vezes era mais entusiasta que o autor deste trabalho, e que infelizmente não está conosco hoje para ver este momento. Obrigado por tudo Pai, fique em paz onde está e olhe por nós**.**

“Que não se cale, nunca a voz

que veio sempre comigo...”

(Augusto Frederico Schimidt)

Resumo

Esta dissertação propõe reunir uma compilação de informações ao apoio do Processo de desenvolvimento iterativo de software, abordando algumas técnicas pesquisadas contidas na engenharia de software e em metodologias existentes. O autor buscou informações em artigos técnicos e em obras que ensinam técnicas no desenvolvimento de softwares contidas em textos de engenharia de software como de Pressman, e técnicas de Projetos Orientados a Objetos em obras de Larman.

Após percorrer pelos textos destes autores, proponho o Processo de Software Iterativo, depois de encontrar nos textos de Larman a possibilidade da união do processo incremental, da metodologia Rup e técnicas ágeis.

Após montar este processo, o autor faz exemplificações na visão do Papel do Analista de Sistemas, e do Designer, que ficariam incumbidos de dar diretivas para a implementação do projeto usando diagramações UML, aderindo os requisitos levantados com o cliente, emergidos no processo iterativo aqui proposto.

Palavras Chaves: Projetos de Software, UML, Engenharia de Software, RUP, SCRUM, XP.

LISTA DE FIGURAS

Lista de Quadros

INTRODUÇÃO

Há pouco tempo atrás muito se falava da necessidade de implementar tecnologia de informação nas empresas em curto espaço de tempo, então grandes equipes de profissionais de TI e de desenvolvedores de sistemas foram mobilizados a criar soluções desde pequenas a grandes instituições visando unicamente à conclusão do serviço (não importando os meios) e a entrega do produto final, deixando a documentação e o planejamento em segundo plano. O nível de maturidade do desenvolvimento dos sistemas hoje já está bem maior do que era há 10 anos, porém, ainda hoje, muitos sistemas são desenvolvidos sem qualquer metodologia ou modelos de processos de desenvolvimento de software. O resultado de um sistema sem um direcionamento formal, como é o caso dos que não seguem alguma metodologia, no geral culminam em sistemas que não atendem o que o cliente requisitou, não funciona com a devida eficiência e ainda torna quase impossível uma prospecção de prazo de entrega, ou de acompanhamento de metas do projeto, gerando ainda muitas horas extras improdutivas dos recursos de TI. Muitos descrevem esse cenário como “crise do software”, que surgiu nos meados dos anos 70 e ainda hoje perdura, descrita no Capitulo I.

Este trabalho visa explorar algumas soluções encontradas para evitar o mau andamento de um projeto de software, citando a engenharia do software no Capitulo II, encontradas desde no mundo acadêmico em alguns artigos e obras de autores renomados, como também na experiência do

autor, onde em algumas situações testemunhou algumas soluções efetivas em âmbito comercial e corporativo.

Este trabalho usará como base, o processo unificado (RUP), explicado no Capítulo III como metodologia e irá explanar o que é, e como pode ser aplicado no dia a dia em um projeto de desenvolvimento de software. Uma vez explanada a aplicação do RUP, torna-se necessário citar suas fases e algumas técnicas de procedimento para auxiliar a conclusão destas.

O trabalho também tem o propósito de servir de guia para aqueles que precisam de uma compilação compacta para o desenvolvimento de seus próprios softwares, oferecendo algumas opções de ações que podem evitar imprevistos em tempo de projeto, onde mostro o plano de Ação proposto por Larman no Capitulo IV.

Usando a ferramenta CASE Enterprise Architect, para organizar a diagramação, proponho um modelo de documentação no levantamento dos requisitos, a construção de casos de uso e a associação de regras de negócio a esses artefatos, passando depois pela especificação técnica que faz uma abordagem profundamente técnica das funcionalidades em questão servindo de apoio a equipe de desenvolvedores.

Uma vez esses artefatos concluídos e validados têm um real retrato estrutural e funcional do projeto, auxiliando a consulta de futuros colaboradores no projeto ou ainda possibilitando a mitigação de riscos de projeto, como a falta de devida execução dos requisitos estabelecidos.

As práticas de desenvolvimento ágil e processo iterativo são apoiados pela IBM e Microsoft que usam este modelo que permitem recursos

trabalhar de uma forma mais planejada evitando perder a qualidade do produto final e permitindo a aplicação de métricas reais.

Primeiramente, o trabalho vai explanar o processo de desenvolvimento iterativo e ágil de software na visão do profissional de Analista de Sistemas, que pode atuar com diversos papeis em um projeto. O autor para isso se baseou no “Processo” descrito pela engenharia de Software proposta por Pressman no Capitulo II, e na metodologia RUP, e métodos ágeis Scrum e XP, todos explanados no Capitulo II. Nesse capítulo tentarei expor o porquê do interesse destes métodos, além deles serem a base das idéias citadas ao desenrolar do trabalho.

Após a explanação dos métodos no primeiro e segundo capítulo, onde vou descrever os processos de desenvolvimento de software, apresentar um modelo de abordagem das fases, mostrarem as atividades de criação dos artefatos para o apoio no desenvolvimento e mapeamento de elementos para discussão em reuniões para tomadas de decisões no projeto, culminando no plano de ação de Larman.

CAPITULO 1 – A CRISE DO SOFTWARE SEGUNDO PRESSMAN

Neste tópico, busquei nos livros de Pressman a razão de haver uma metodologia formal para o desenvolvimento do software, e Pressman nos deu algumas razões começando pela crise do software.

Em Pressman (PRESSMAN, 2001), ele cita a possibilidade da crise do software no item 1.3 do primeiro capitulo em “Software: A crisis on the horizon”.

Crise essa por razão da baixa de qualidade nos softwares, e no fim da página 11 ele cita que o problema não é somente “não funciona adequadamente”. São questões como “Como vamos suportar um crescente volume de software existente?”. Depois dessa reflexão, ele lista os mitos de software. Mitos esses que desorientam os profissionais, os fazendo a agarrar em mitos muitas vezes para amenizar a pressão exercida de seus superiores. Mitos esses como (PRESSMAN , 2001, p. 35):

- “Nós já temos um livro cheio de padrões e procedimentos, isso já dá suporte para minha equipe no desenvolvimento de softwares” – O livro pode existir, mas ele é aplicável ao seu problema? Os possíveis usuários deste têm o conhecimento de sua existência? Ele reflete o que há na moderna engenharia do software? Ele está direcionado para a qualidade de software? Na maioria dos casos, a resposta é Não.

- “Nós temos equipamentos de ultima geração, isso garantirá a efetividade do sistema” – Qualidade do software vai muito além de hardware, muitas vezes a utilização de ferramentas CASE garantem mais a qualidade que um maquinário caro.

- “Se estivermos atrasados no cronograma, podemos colocar mais recursos e isso nos garantirá alcançar o tempo perdido” – O desenvolvimento de software não é um processo mecânico de manufatura. É necessário considerar o devido treinamento do novo membro da equipe. As pessoas podem ser adicionadas, mas é necessário planejar de uma forma bem coordenada sua entrada.

- “Se eu decidir por terceirizar meu processo, eu posso relaxar e deixar que a contratada resolva tudo” – Se o contratante não souber como gerenciar um projeto de software, ele irá invariavelmente se envolver em muitas discussões com o fornecedor.

- “Uma definição generalista é suficiente para o início do desenvolvimento do software, nós podemos definir os detalhes depois.” - Uma definição pobre em detalhes

de domínio, função, performance, interfaces, restrições e critérios de validações, é a maior causa de falha nos esforços de desenvolvimento de software. Essas características devem ser determinadas somente depois da comunicação entre o desenvolvedor analista e o cliente.

- “Os requisitos do projeto sempre mudam, e essa mudança é acomodada porque o software é flexível” - Os requisitos de software podem mudar, mas quanto mais já tiver sido feito do projeto, maior será o impacto. Estudos mostram que o impacto da alteração em fase de definição de requisitos é transparente, porém em desenvolvimento pode ser de 1.5 vezes a 6 vezes do esforço, e depois da liberação do software, esse impacto chega de 60 a 100 vezes o valor do esforço inicial.

- “Uma vez o software tenha sido escrito e liberado, meu trabalho está terminado” – Estudos indicam que 60% a 80% do esforço gasto em software, foi gasto depois da primeira liberação do software.

- “Só posso avaliar a qualidade meu software quando estiver pronto” – Um dos mecanismos de garantia de

qualidade de software mais efetivos pode ser usado desde a fase de concepção é chamado de “Formal technical review”. Que funciona como um filtro de qualidade através do processo de desenvolvimento, mais efetivo que testes para identificar classes defeituosas (Test Cases e Unitary Tests).

- “O único trabalho passível de distribuição ao cliente, é um programa funcionando.” – Um programa é somente uma parte da “Software Configuration” que incluem muitos elementos. Documentos podem oferecer um conjunto de definições para uma engenharia bem sucedida, e o mais importante, uma guia para o suporte do software.

- “Engenharia do software irá nos gerar um volume muito grande de documentação inútil e perderemos tempo com isso” – Engenharia não prevê gerar documentos, e sim gerar qualidade. Uma melhor qualidade nos leva a pouco retrabalho e isso nos resulta em entregas rápidas.

Segundo Pressman (PRESSMAN, 2001) o autor ainda cita alguns pontos de atenção na computação atual, que leva a crer que muitos mitos estão se dissipando, mas os desafios atuais criam outros mitos que tomam o lugar dos mitos antigos.

Softwares são distintos entre si, por exemplo, se pensarmos em três produtos da computação, ao menos um destes será diferente dos demais se considerarmos o software e desconsiderarmos o hardware.

Muitos autores vêm discutindo o impacto da “Era da informação” na sociedade. O que nos leva a entender a maior responsabilidade e impacto dos sistemas de informação nas atividades humanas. Aumentando ainda mais a demanda por qualidade e a preocupação com riscos de sistema.

Para Pressman (PRESSMAN, 2001) ele discute o impacto negativo de softwares nas atividades humanas, citando as obras de Minasi “The Software Conspiracy: Why Software Companies Put out Faulty Products, How They Can Hurt You, and What You Can Do, McGraw-Hill, 2000)” e DeMarco (Why Does Software Cost So Much? Dorset House, 1995).

CAPITULO 2 – A ENGENHARIA DO SOFTWARE

Segundo a Wikipédia:

> "*Engenharia do software é uma área do conhecimento da computação voltada para a especificação, desenvolvimento e manutenção de sistemas de software aplicando tecnologias e práticas de gerência de projetos e outras disciplinas, objetivando organização, produtividade e qualidade.*" [1]

Para Pressman (PRESSMAN, 2001), a engenharia do software é dividida em três camadas, Processo, Métodos, Ferramentas.

- Processo: Processo é a fundação da engenharia do software, e também desta monografia, o processo é o que garante a aderência das camadas dos métodos e das ferramentas a fim de construir um software de qualidade.

- Métodos: Os métodos fornecem um suporte técnico de COMO FAZER, no momento da construção do software. Os métodos compreendem uma gama de tarefas que incluem a análise de requisitos, design

(Modelagem), construção do software (codificação), testes e suporte.

- Ferramentas: As ferramentas fornecem um suporte automatizado ou semi automatizado para os processos e métodos.

2.1 – CMMI: A MATURIDADE DE UM PROCESSO

Em Pressman (PRESSMAN, 2001, p.24) também define os níveis de maturidade de processo de software (CMMI):

- Nível 1 – Inicial: Processo de software caracterizado por ser imediatista e muitas vezes caótico
- Nível 2 – Repeateble: Gerenciamento básico do processo foi estabelecido para rastrear custos, cronograma e funcionalidades. E o mesmo processo é aplicado com produtos similares.
- Nível 3 – Definido: É o processo que gerencialmente e tecnicamente é documentado, padronizado e integrado a um processo de desenvolvimento amplo da organização.
- Nível 4 - Gerenciado: Métricas detalhadas e bem definidas são coletadas, tornando possível gerenciar quanticamente tanto o processo quanto o produto. E

deve compreender todas as características do nível três.

- Nível 5 – Otimizado: Através de uma contínua coleta de feedbacks do processo e de idéias inovadoras de teste, se torna possível uma contínua melhora no processo.

2.1.1 – TIPOS DE PROCESSOS

Após descrever as camadas da engenharia do software, podemos através das definições de Pressman (2001, p.34) descrever alguns processos de engenharia de software existentes. Um deles é o processo seqüencial linear, conforme a imagem abaixo.

FIGURE 2.4
The linear sequential model

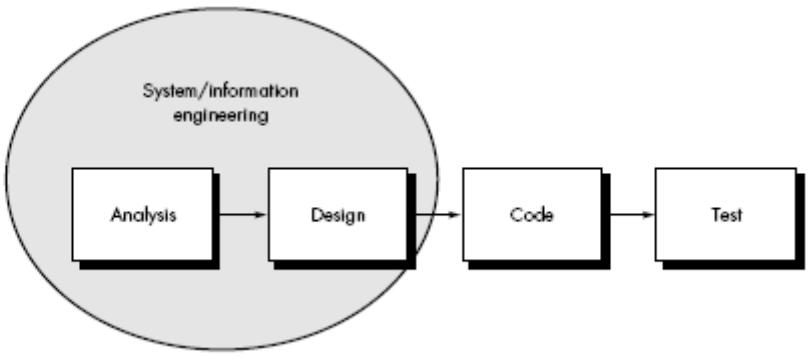

Figura 1: Linear Seqüencial (PRESSMAN, 2007, p.34)

Esse modelo consiste em aplicar os métodos na seqüência acima, de forma linear.

Já em Pressman (2001, p. 31), o autor discute a idéia do processo por prototipação (Prototyping Paradigm) para a definição de detalhes não passados pelo cliente, ou no caso de dúvidas técnicas de qual forma deve ser definida a interação Homem/Maquina:

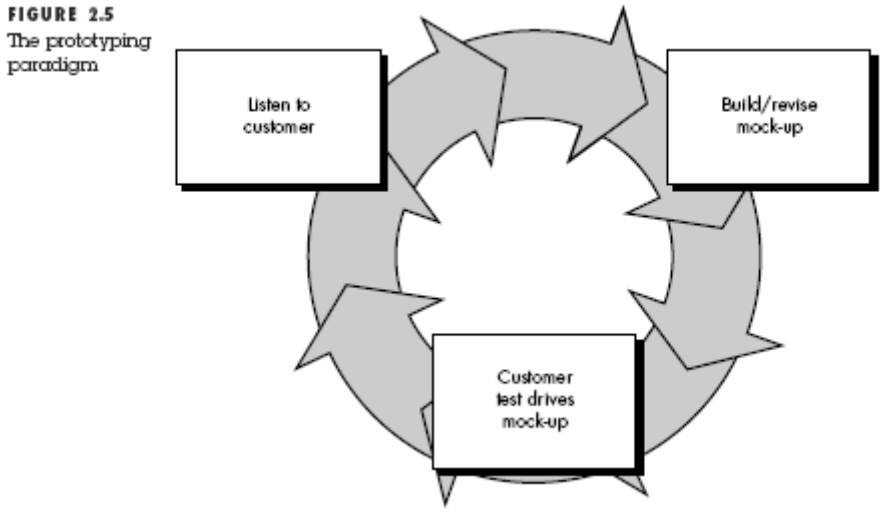

Figura 2: Paradigma de Protótipo (PRESSMAN, 2001, p. 31)

O protótipo então é avaliado pelo usuário, e então pode haver um refino nos requisitos do software a ser desenvolvido. A iteração ocorre enquanto ao mesmo tempo em que o protótipo satisfaz as necessidades do cliente, o desenvolvedor compreende melhor o que precisa ser feito.

Pensando ainda em processos iterativos, o autor faz uma mescla do processo linear seqüencial com a interatividade do modelo de processo protótipo e discutiu sobre o modelo que ele chama de Modelo de Processo Incremental.

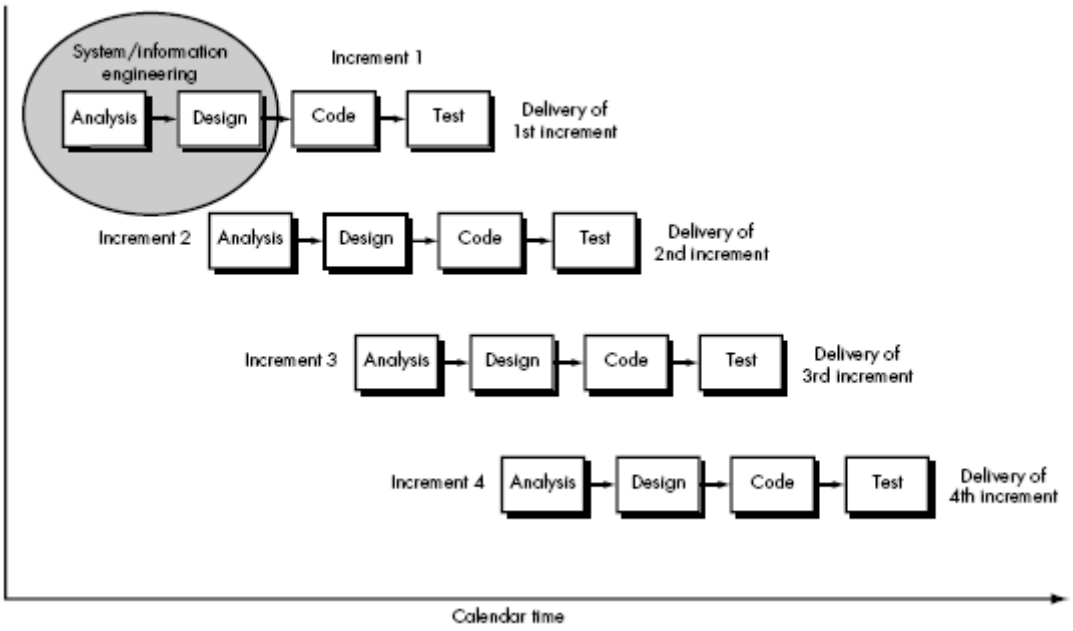

Figura 3: Processo Incremental (PRESSMAN, 2001, p. 34)

Porém a principal diferença do Incremental para o Protótipo, segundo o autor, é que diferentemente do modelo protótipo, o incremental é liberado para o cliente como uma versão, que poderá servir tanto para avaliação como para uso final, servindo para o que o usuário solicitou.

Craig Larmam recomenda um processo iterativo, na seguinte frase. (LARMAN, 2007, p45) "*O Desenvolvimento interativo representa a parte central sobre como a análise orientada a objeto é melhor praticada...*"

Uma coisa notada também na pesquisa, confrontando o trabalho de Pressman, RUP e de Larman, é que o processo definido por Pressman como "Processo Incremental" é o processo utilizado na metodologia RUP, que por sua vez é utilizada nas recomendações de projetos orientados à objetos por Larman.

CAPÍTULO 3 – Metodologia

3.1 - UP

Segundo Larman (LARMAN, 2007) UP (Unified Process) é um processo iterativo de desenvolvimento de projeto de software que descreve uma abordagem para a construção, implantação e possivelmente, a manutenção de software. Para o desenvolvimento de sistemas orientados a objetos, um refinamento que merece destaque é o RUP (Rational Unified Process), que segundo Larman é muito adotado.

O que torna a pesquisa do UP mais interessante, que descreve Larman (LARMAN, 2007), é a flexibilidade com outras práticas, como outros modelos de iteração ou desenvolvimento ágil tais como Scrum e Extreming Programming. Isso no modo de ver do autor permite uma customização em diferentes empresas, utilizando o UP para o processo de análise funcional e criação de artefatos de negócio, e Scrum ou XP como prática na fase de construção do software.

3.2 - Iteratividade

Larman explica que a iteratividade é a vantagem do processo unificado sobre outros modelos por ele citados, como o modelo em cascata por exemplo. Ele também faz uma pesquisa comparativa entre a eficiência dos dois processos. E segunda segundo suas fontes resumidas o UP levou vantagem sobre os modelos não interativos e incrementais, ou seja, que não evoluem

empiricamente, concluindo suas fases antes de começarem outras, como um modelo de cascata. Esses modelos não permitem interação de uma fase na outra, como por exemplo, notar que na codificação do software, que a funcionalidade não atenderá o cliente sem que antes seja feito um novo módulo, ou um novo subsistema. Esse problema poderia ter sido evitado se no momento da elaboração do software (momento onde é interativamente implementada) tivesse uma interação com a fase de concepção, questionando como seria possível a funcionalidade ser implementada sendo que não foi pensado em um requisito que não foi pedido ao cliente. Nesse cenário então seria possível alinhar o entendimento funcional do software entre a equipe sem antes codificar de forma equivocada, economizando muito retrabalho por parte do desenvolvimento. (LARMAN, 2007)

Esse cenário descrito acima, não é previsto em um modelo em cascata, que se dá por um modelo restritamente seqüencial, porque a análise dos requisitos do cliente ficou definida e não possibilitou uma interação entre quem levantou os requisitos e quem definiu os casos de uso do software.

Craig Larman, um renomado engenheiro de software, citou uma pesquisa que o departamento de defesa dos EUA havia adotado um padrão de cascata no fim da década de 70, e no começo da de 80 depois de significativas falhas, orçamentos de pelo menos 50% dos projetos de software foram cancelados ou não usáveis. No fim da década de 1980, o Dr. Frederick Brooks, escreveu sobre a necessidade de aplicar métodos DII (Desenvolvimento iterativo e incremental) que nos anos 90 foi amplamente aceito como sucessor do método e a partir daí nasceram os métodos de UP, DSDM,Scrum,XP e etc. (LARMAN, 2007)

Por essa razão optei por abordar esses métodos no trabalho, um procedimento utilizando o UP ou RUP, como modelo de ciclo de vida de software, Scrum, como modelo de interação de informações entre a equipe, e XP – Extremme Programming, para agilidade na codificação com qualidade e boa funcionalidade.

3.3 – CICLOS DE VIDA DE UM SOFTWARE (SEGUNDO RUP)

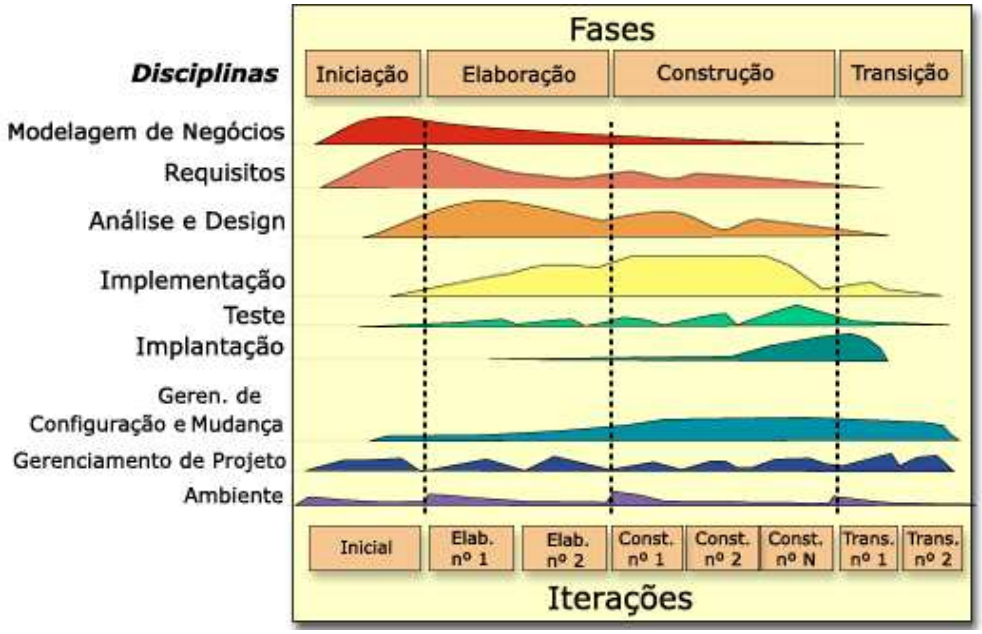

Figura 4: RUP (RUP, 2002) [2] [2]

3.3.1 – FASES DO CICLO DE VIDA DO RUP

Inception/Iniciação: Visão do negócio, onde é definido o escopo e estimativas vagas. A concepção é um passo inicial que abrange talvez 10% dos casos de uso, requerimentos não funcionais, criação de um caso de negócio e preparação do ambiente de desenvolvimento visando preparar o início do desenvolvimento comece na fase seguinte da elaboração.

Elaboration: É uma visão refinada, implementação interativa da arquitetura central, resolução de altos riscos, indefinição da maioria dos requisitos e do escopo e estimativas mais realistas. (LARMAN, 2001)

Construção: Segundo ainda Larman (LARMAN, 2001), é a implementação interativa dos elementos restantes de menor risco e mais fáceis e preparação para a implantação.

Transição: testes betas e implantação.

3.3.2 – DISCIPLINAS DO RUP

Segundo o manual RUP, uma disciplina mostra as atividades que são necessárias para construir um determinado número de artefatos. As disciplinas estão escritas de uma forma geral com um resumo de todas as atividades, os papéis e artefatos envolvidos. (RUP, 2002) [2]

As disciplinas segundo o manual do RUP são as seguintes:

- Modelagem de Negócios
- Modelagem de Requisitos
- Análise e Design
- Implementação
- Teste
- Implantação
- Ambiente
- Gerenciamento de Projeto
- Gerenciamento de Configuração e Mudança

3.4.1 – PAPEIS DO RUP

Segundo o manual do RUP (RUP, 2002) [2], um papel é uma definição abstrata de um conjunto de atividades executadas e dos respectivos artefatos.

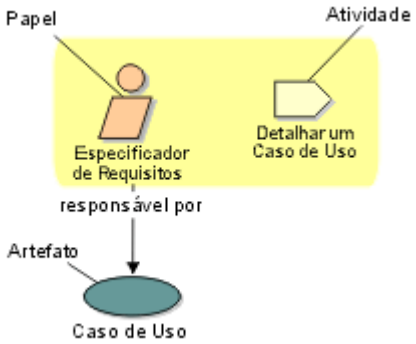

Figura 5: Papeis (RUP, 2002) [2]

Normalmente os papéis são desempenhados por uma pessoa ou um grupo de pessoas que trabalham juntas em equipe. Um membro da equipe do projeto geralmente desempenha vários papeis distintos. Os papéis não são pessoas, somente descreve como as pessoas se comportam no negócio e quais suas responsabilidades .

Os papéis podem ser descritos como: (RUP, 2002) [2]

- Analistas
    - Analista de Sistemas
    - Designer de Negócios
    - Analista de Processos de Negócios
    - Especificador de Requisitos
    - Analista de Teste

        - Designer de Interface de Usuário

- Desenvolvedores
        - Designer de Cápsula (Componentes)
        - Revisor de Código
        - Designer de Banco de Dados
        - Implementador
        - Integrador
        - Arquiteto de Software
        - Revisor e Design
        - Designer
        - Designer de Teste
- Testadores
- Gerentes
        - Gerente de Projeto
        - Engenheiro de Processos
- Outros envolvidos
        - Envolvidos
        - Desenvolvedor do Curso
        - Artista Gráfico
        - Especialista Ferramenta
        - Administrador de Sistemas

- Redator Técnico

3.5.1 – ARTEFATOS

Artefatos são produtos de trabalhos finais ou intermediários produzidos e usados durante os projetos. Os artefatos são usados para capturar e transmitir informações do projeto. Um artefato pode ser um dos seguintes elementos: (RUP, 2002) [2]

Um documento: Um caso de Uso, Documento de arquitetura etc.

Um modelo: Um modelo de Casos de Uso, Modelo de Design

Um elemento do Modelo: Como uma classe ou um subsistema

Esta monografia tem o foco de explicar o processo de desenvolvimento de software na visão profissional Analista. Portanto as disciplinas mais abordadas são as relacionadas diretamente a ele, como a Modelagem de Requisitos, Análise e Design e Teste, mais especificamente dos papéis de Analista de Sistemas, Analista do Processo de Negócios, Designer de Negócio, Especificador de requisitos e Analista de Teste.

3.6.1 – ARTEFATOS E TIVIDADES DO ANALISTA DE PROCESSO DE NEGÓCIOS

O analista de negócios é responsável pela arquitetura de negócios do projeto, ele coordena a modelagem de casos de uso de negócio e da modelagem de negócio, definindo quem são os autores de negócio e seus casos de uso, definindo e delimitando a organização que está modelando (RUP,2002)[2] .

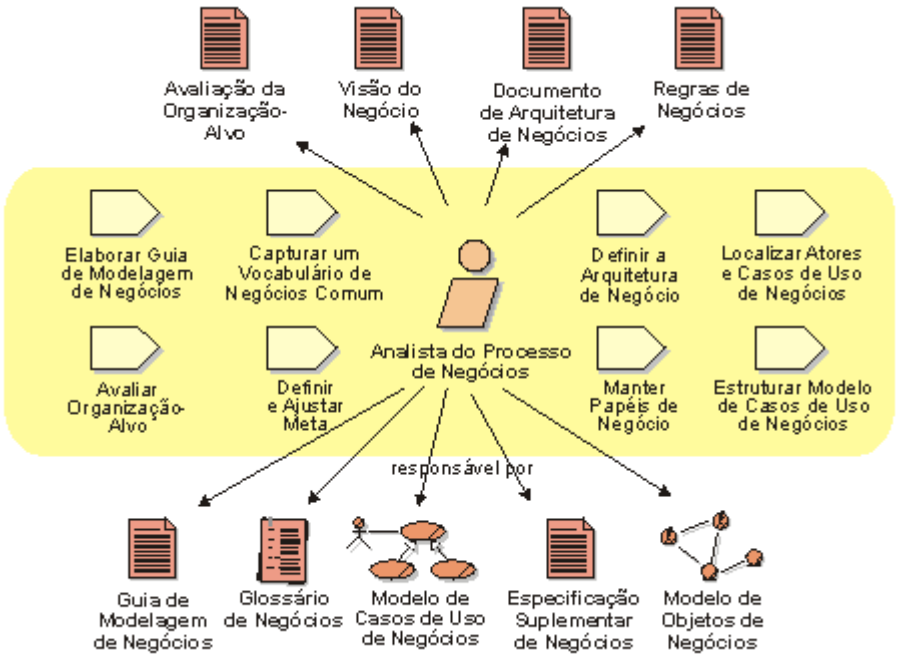

Figura 6: Analista de Processo (RUP, 2002) [2]

O analista de negócios deve estar preparado para: (RUP, 2002) [2]

- Avaliar a situação da organização-alvo na qual o produto final do projeto será implantado.
- Entender as necessidades do cliente e do usuário, suas estratégias e metas.
- Facilitar a modelagem da organização-alvo.

- Discutir e facilitar um esforço da engenharia de negócios, se necessário.
- Realizar uma análise de custo/benefício de quaisquer mudanças sugeridas na organização-alvo.
- Analisar e auxiliar aqueles que comercializam e vendem o produto final do projeto.

Ele também está a cargo de escrever o documento de regras de negócios, que deve ser organizado conforme seu escopo, e o documento de Visão. (RUP,2002)

Pelo analista ser o responsável pela coordenação do design de negócio, ele deve também descrever as diretrizes da modelagem de negócio, utilizando do artefato “Guia de modelagem de negócios” e organizar o modelo de objetos de negócio. (RUP, 2002) [2]

3.7.1 – ARTEFATOS E ATIVIDADES DO DESIGNER DE NEGÓCIOS

O Designer de Negócio detalha a especificação de uma parte da organização, descrevendo o fluxo de trabalho de um ou vários casos de uso de negócios. Este papel define as operações, atributos e relacionamentos de um ou vários trabalhadores de negócio e entidades de negócio. (RUP, 2002) [2]

Uma pessoa que atua como designer de negócio deve estar familiarizado com as ferramentas para capturar os modelos de negócio, o conhecimento do negócio seria uma vantagem (RUP, 2002) [2].

Figura 7: Designer Negocio (RUP,2002)

Caso de Uso de Negócios: Descreve uma seqüência de ações que produz um resultado de valor para um ator de negócio. (RUP, 2002) [2]. Seu conteúdo principal é a descrição do Fluxo de trabalho, e as metas que se deve atingir em um ponto de vista do autor do negócio.

Trabalhador do Negócio: Se trata do autor do negócio, que devem ser identificados enquanto é escrito o documento de Caso de Uso de Negócios

Entidade de Negócio: Representa uma entidade passiva, que não inicia interações por si. Um objeto de entidade de negócio pode participar de várias realizações de casos de uso de negócio e normalmente dura mais que uma iteração (RUP, 2002) [2]. Essa entidade é uma classe estereotipada “entidade de negócio” na notação UML.

Unidade Organizacional: é um conjunto de trabalhadores de negócios, entidades de negócios, diagramas e outras unidades organizacionais.

(RUP,2002) [2]. Esse elemento é utilizado para dividir os grupos de elementos de um negócio em sub-partes, representados como pacotes no UML.

3.8.1 – ARTEFATOS E ATIVIDADES DO ANALISTA DE SISTEMAS

O analista de sistemas deve liderar e coordenar a identificação de requisitos e a modelagem de casos de uso, definindo os atores e as funcionalidades do sistema (RUP,2002) [2].

Ainda segundo o manual RUP o analista de sistemas é um facilitador que deve possuir habilidades de comunicação acima da média, e é fundamental que os profissionais que desempenham este papel tenham domínio do negócio e da tecnologia (RUP, 2002) [2].

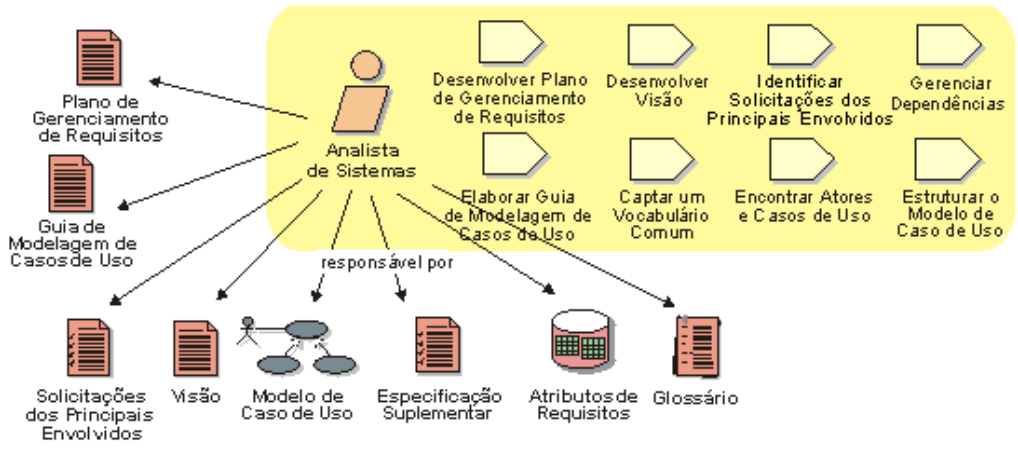

Figura 8: Analista de Sistemas (RUP, 2002) [2]

O analista de sistemas então seria responsável por construir os documentos:

Plano de Gerenciamento de requisitos:

> "*Descreve a documentação de requisitos, os tipos de requisitos e seus respectivos atributos de requisitos, especificando as informações e os mecanismos de controle que devem ser coletados e usados para avaliar, relatar e controlar mudanças nos requisitos do produto."*
> (RUP, 2002) [2]

Guia de Modelagem de Caso de Uso: Que descreve como se devem modelar os casos de uso. Isso também é recomendado em Cockburn (2005, p29), onde ele define que um bom caso de uso a ser escrito depende de três fatores; Técnica - que são as reflexões momento a momento de como pensar, redigir e a seqüência de como trabalhar; Qualidade- que define como saber se o que foi escrito é aceitável para seu propósito; Padrões - Que dizem no que as pessoas envolvidas no trabalho concordam quando escrevem os casos de uso.

Documento de Visão: onde é definida a visão que os envolvidos têm do produto a ser desenvolvido em termos de necessidades (RUP, 2002) [2].

Modelo de casos de uso: Que é o modelo de funções pretendidas e serve como contrato estabelecido entre o cliente e os desenvolvedores. Este documento é usado como fonte para as atividades de análise, design e teste. (RUP, 2002) [2].

Especificação Suplementar: São os requisitos que não são capturados na construção de casos de uso, podem ser requisitos legais, atributos de qualidade ou requisitos não funcionais como restrições de sistemas operacionais, compatibilidades ou design. (RUP, 2002) [2].

Atributos de Requisitos: O manual RUP diz o seguinte: "*Este artefato contém um repositório de dependências, atributos e requisitos de projeto para controlar a partir de uma perspectiva de gerenciamento de requisitos*.". Isso se daria por uma matriz de requisitos mostrando seus atributos, como status por exemplo. E também uma árvore de rastreabilidade, onde seria possível exibir graficamente o relacionamento dos requisitos (RUP, 2002) [2].

Glossário: Documento que define os termos importantes do projeto, um dicionário com o significado de cada termo pertinente no projeto. O glossário pode ser utilizado pelos desenvolvedores que ao projetar e definir classes pode usar termos aderentes ao negócio. Para os analistas, eles podem capturar termos específicos do projeto e definir claramente as regras de negócio (RUP, 2002) [2].

3.9.1 – ARTEFATOS E ATIVIDADES DO ESPECIFICADOR DE REQUISITOS

O especificador de requisitos deve detalhar a especificação de uma parte da funcionalidade, seja ele um pacote de casos de uso ou subsistema. Ele também fica a cargo de manter a integridade dos casos deste pacote. O RUP recomenda que o mesmo responsável por um pacote de casos de uso, seja responsável pelos casos de uso e atores contidos nele (RUP,2002) [2].

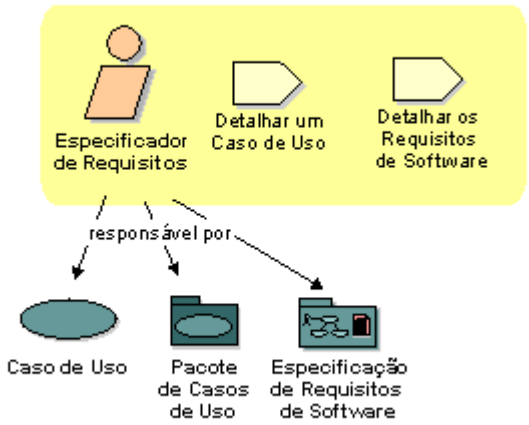

Figura 9: Especificador de Requisitos (RUP, 2002) [2]

A Especificação de Requisitos de Software (SRS) captura todos os requisitos de software para o sistema ou para uma parte do sistema. Quando uma modelagem de casos de uso é utilizada, este artefato consiste em um pacote contendo casos de uso do modelo de casos de uso e Especificações Suplementares aplicáveis. (RUP, 2002) [2].

3.10 – FASE INCEPTION (INICIAÇÃO)

3.10.1 – DISCIPLINA DE MODELAGEM DE NEGÓCIO

Esta disciplina é a maior atividade da fase de Iniciação (Inception), e ela é determinante para a definição do que o cliente quer que seja construído. Para

qualquer produto, o cliente visa um produto de boa qualidade, preço justo e que seja útil. E para determinar a utilidade deste nosso produto que é um software, compreender o que o cliente deseja é imprescindível para cumprir bem o projeto.

Uma vez já detalhados o que é feito na fase de análise, que são constituídos nas duas primeiras fases, nos itens abaixo podemos representar de forma gráfica na seguinte disposição:

Papeis e Atividades Disciplina de Modelagem de Negócios

Figura 10: Papeis Atividades em Negócios (RUP, 2002) [2]

Papéis e Artefatos Disciplina de Modelagem de Negócios

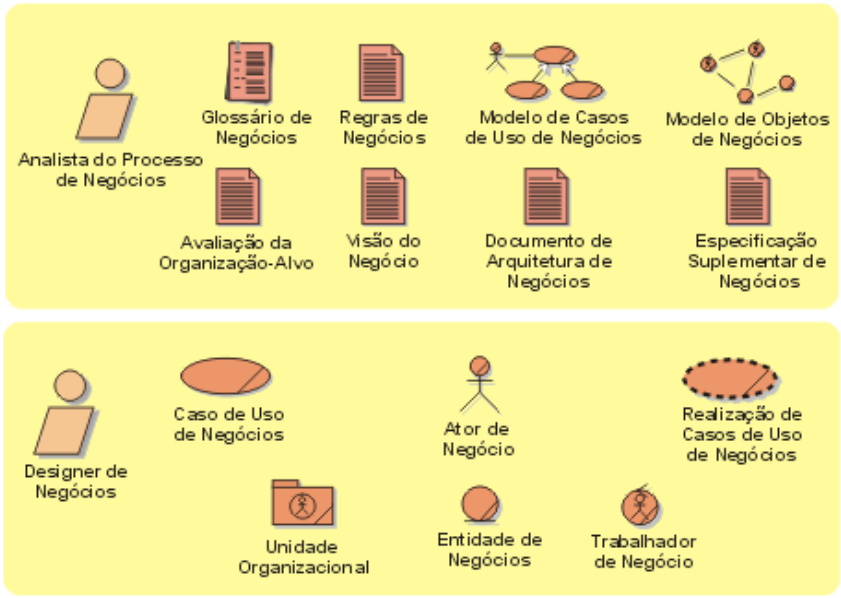

Figura 11: Papéis e artefatos de negócios (RUP, 2002) [2]

3.10.2 – DISCIPLINA DE REQUISITOS

Uma vez com os artefatos da fase de modelagem de negócios, ou com a interação dos envolvidos nesta fase, é possível desempenhar as atividades de requisitos, descritas anteriormente e esboçadas abaixo organizada por fase, papel e atividades/artefatos.

Conjunto de Papéis e atividades da disciplina de requisitos

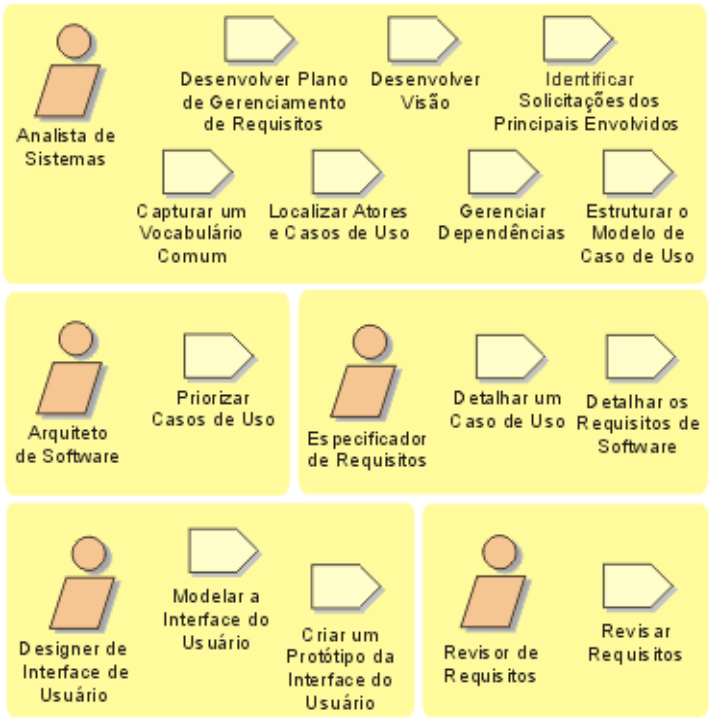

Figura 12: Atividades Requisitos (RUP, 2002) [2]

Conjunto de Papéis e Artefatos Disciplina de Requisitos

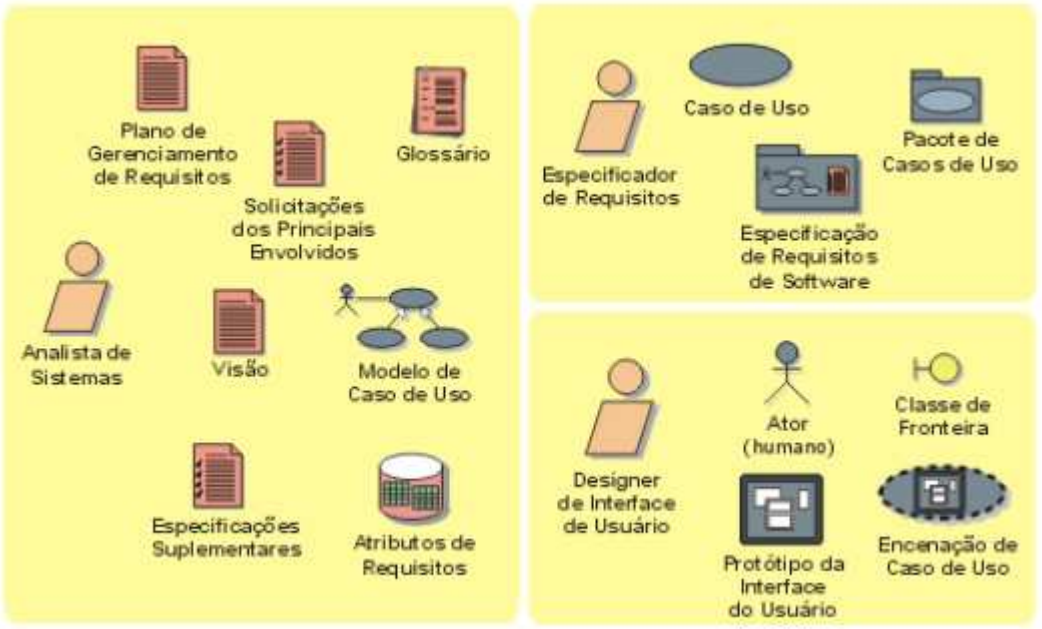

Figura 13: Artefatos requisitos (RUP, 2002) [2]

Neste diagrama, notamos que é outro artefato que nos pode ser muito útil na validação de requisitos junto ao cliente, que é o Protótipo da interface, feita pelo Designer de interface de usuário. Porém se trata de um papel opcional, porque nem todos os sistemas têm interface com usuário, ou que esta precisa ser validada.

Claro que um sistema não é feito somente de regras e definições alinhadas com o cliente, é necessário também criar definições de software, definições técnicas que tem que ser aderentes a todas essas informações levantadas e documentadas através dos artefatos.

Para o consumo destas informações, levantadas com o cliente, pela equipe técnica, existe a fase de Análise e Design, que transforma os requisitos em um design do sistema a ser criado. Além de definir uma arquitetura sofisticada para o sistema, e de adaptar este design para um design que corresponda ao ambiente de implementação, projetando-o para fins de desempenho. (RUP, 2002) [2]

3.11 – FASE DE ELABORAÇÃO

3.11.1 – DISCIPLINA DE ANÁLISE E DESIGN

A análise e design, é o foco central deste trabalho e a disciplina mais trabalhada na fase de elaboração, nesta fase podemos aplicar alguns conceitos como Arquitetura de Software, Divisão de Camadas, Eventos e Sinais, Mecanismos de análise. Se reparar no gráfico de atividades do RUP, esta disciplina se inicia no fim da fase de Iniciação (Inception) e por toda a fase de Elaboração (Elaboration) e boa parte da Implementação (construction). Esta disciplina está relacionada com as seguintes abaixo:

- A disciplina Modelagem de Negócios fornece um contexto organizacional do sistema.
- A disciplina Requisitos fornece uma contribuição básica para a disciplina Análise e Design.
- A disciplina Teste testa o sistema projetado durante a disciplina Análise e Design.
- A disciplina Ambiente desenvolve e mantém os artefatos de suporte que são utilizados durante a disciplina Análise e Design.

- A disciplina Gerenciamento de Projeto planeja o projeto e cada iteração (descrita em um Plano de Iteração).

(RUP, 2002) [2]

Segundo o RUP (RUP, 2002) [2], a finalidade desta disciplina é transformar os requisitos em um design do sistema a ser criado, desenvolver uma arquitetura sofisticada para o sistema e adaptar o design para que corresponda ao ambiente de implementação, projetando-o para fins de desempenho (performance).

3.11.2 – ATIVIDADES E ARTEFATOS DO ARQUITETO DE SOFTWARE

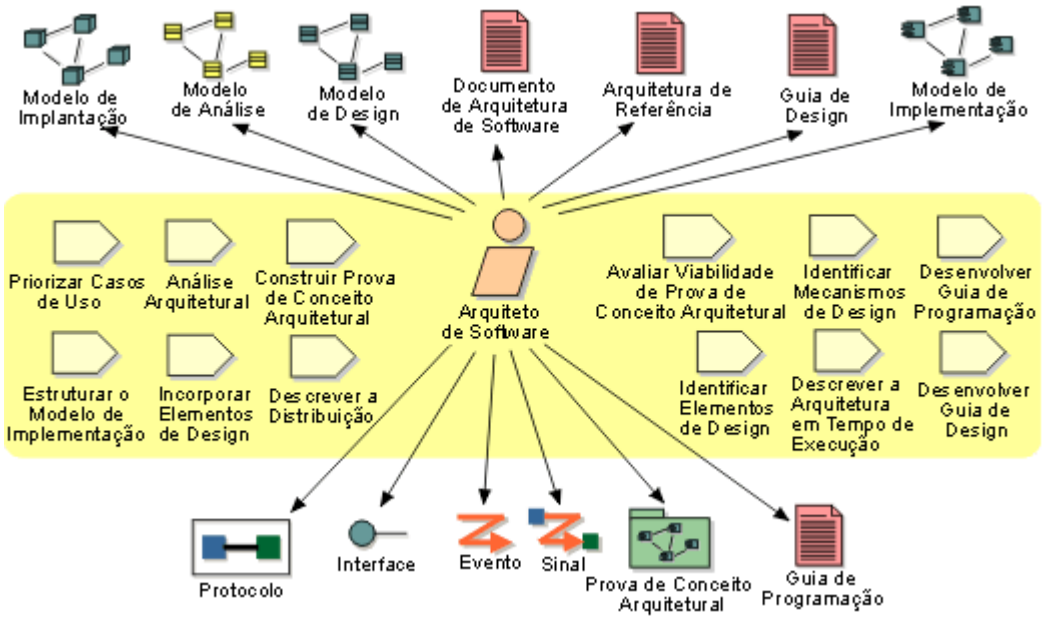

Figura 14: Arquiteto de software (RUP, 2002) [2]

Para esta disciplina, existe um papel muito importante figurado pelo Arquiteto de Software, ele deverá passar as diretivas para os Designers de software e banco de dados, a fim de atingir as finalidades citadas anteriormente.

Para atingir estes objetivos, ele será o responsável por cuidar dos modelos de implantação, modelo de análise, modelo de design, documento de arquitetura de software, Guia de design (com as diretivas de design) e programação, e modelo de implementação.

O arquiteto também é responsável por avaliar a viabilidade de recursos técnicos no ambiente de implementação, através de provas de conceito.

Resumo de atividades do arquiteto de software segundo o RUP (RUP, 2002) [2]:

- Priorizar Casos de Uso:

- Definir a entrada para a seleção do conjunto de cenários e casos de uso.
- Definir o conjunto de cenários e casos de usos centrais ou mais importantes do sistema.
- Definir o conjunto de cenários e casos de uso que possuem cobertura arquitetural por possuir vários elementos da arquitetura.

- Análise Arquitetural
  - Definir uma sugestão de arquitetura.
  - Definir os padrões de arquitetura.
  - Definir a estratégia de reutilização.
  - Fornecer dados para o processo de planejamento.

- Estruturar o modelo de implementação
  - Estabelecer a estrutura em que a implementação residirá.
  - Atribuir responsabilidades para Subsistemas de Implementação e seu conteúdo.
  -
- Incorporar Elementos de Design: Esta atividade será vista com mais detalhe no próximo capítulo, por se tratar da atividade que resulta nos artefatos de Modelo de Design e Documento de Arquitetura.
  - Analisar interações de classes de análise para localizar interfaces.
  - Refinar a arquitetura.

- Identificar soluções comuns para problemas de design detectados com freqüência, criando uma série de padrões ajudando na reusabilidade.
- Incluir elementos do modelo de design, na arquitetura lógica.

- Descrever Distribuição
  - Mostrando a distribuição de bibliotecas e artefatos compilados, e em quais subsistemas e em quais equipamentos.

- Identificar Mecanismos de Design
  - Refino do design a partir de um ambiente já definido delimitando as possibilidades e recursos.

- Desenvolver Guia de programação.
  - Guia de programação que é composto por convenções do projeto como padrões de codificação, padrões de nomenclatura para funções, campos, variáveis, classes, pacotes e propriedades. Além de definição de um padrão de tratamento de erros, interfaces e compilação.

- Desenvolver o guia de design.
    - O mesmo que guia de programação, mas para os artefatos de design.

3.12 - DEPENDÊNCIAS DE ARTEFATOS

O RUP, ele cria uma ciclo de dependências de artefatos desenvolvidos pelas atividades dos papéis descritos anteriormente, isso pode ser melhor visualizado no diagrama abaixo, extraído do RUP:

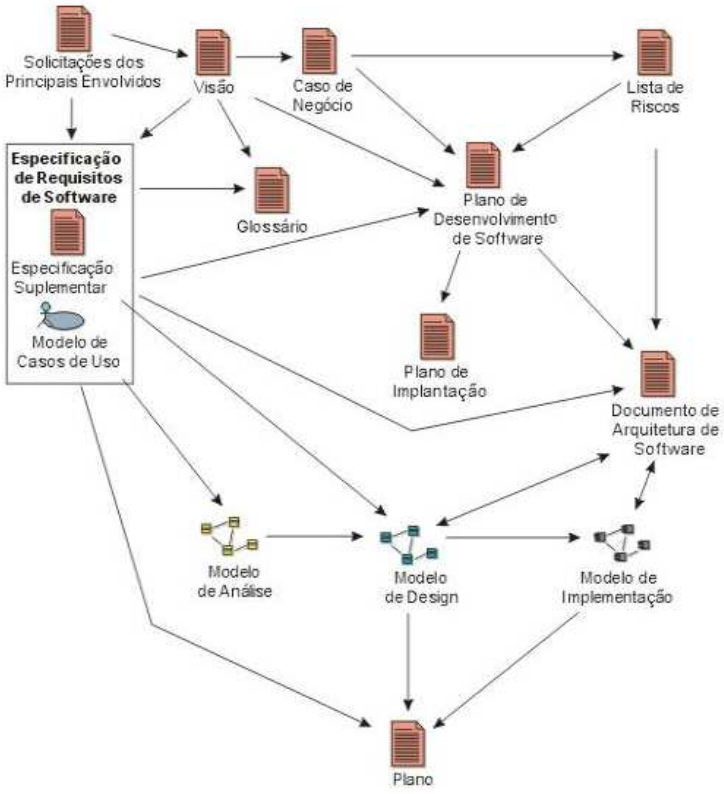

Figura 15: Dependência de artefatos (RUP, 2002) [2]

Já na fase de Elaboração, muitos destes artefatos devem estar já disponibilizados para o início do planejamento da codificação (Implementação).

CAPÍTULO 4 – MODELO PROPOSTO POR CRAIG LARMAN

Passamos pelo o que o RUP diz sobre o processo de desenvolvimento de software, portanto a partir de agora, podemos atribuir a este modelo as técnicas ágeis, como é o que sugere Craig Larman (LARMAN, 2007).

Ele sugere o desenvolvimento iterativo evolutivo, que é um processo que Pressman já descreveu e é citado no primeiro capitulo em “Tipos de Processo”.

Os métodos ágeis segundo Larman (LARMAN, 2007). São os que se aplicam interativamente e evolutivamente em um tempo limitado. Emprega planejamento adaptativo e entregas incrementais encorajando a agilidade, como resposta rápida e flexível às modificações.

E com isso ele elege 13 princípios ágeis: Larman (LARMAN, 2007, P. 45)

- Nossa prioridade mais alta é satisfazer o cliente por meio de entrega pronta e contínua de software de valor.
- Acolher modificações de requisitos, mesmo no final do desenvolvimento. Processos Ágeis valorizam a modificação para vantagem competitiva do cliente.
- Entregar software funcionando com freqüência, preferencialmente usando escala de tempo menos.

- O pessoal do negócio e os desenvolvedores devem trabalhar juntos diariamente ao longo do projeto
- Construir projetos em volta de indivíduos motivados. Dê a eles o ambiente e o apoio necessário e confie que eles vão fazer o serviço.
- O método mais eficiente e efetivo para levar informação de e para a equipe de desenvolvimento é a conversa face a face.
- Software funcionando é a principal medida de progresso.
- Processos ágeis promovem desenvolvimento sustentável.
- Os patrocinadores, desenvolvedores e usuários devem poder manter um ritmo constante indefinidamente.
- Atenção contínua para a excelência técnica e para um bom projeto aumenta agilidade.
- Simplicidade – a arte de maximizar a quantidade de trabalho não realizada é essencial
- As melhores arquiteturas, requisitos e projetos surgem de equipes auto-organizadas.
- Em intervalos regulares, a equipe reflete sobre se tornar mais efetiva, depois sintoniza e ajusta seu comportamento de acordo com isso.

Ainda sobre este tópico, Larman (LARMAN, 2007) aconselha o uso de:

UML com Análise Orientada a Objetos: Pensar em objetos e diagramando em UML, segundo Larman, é crucial para se construir um excelente projeto ou melhor avaliar um já feito.

POO (Programação Orientada a Objetos): Que se trata dos padrões de um projeto guiado a responsabilidades, ou GRASP, momento de definição de quais métodos irão pertencer a quais classes, e como esses objetos devem interagir entre si.

Casos de Uso: O POO está fortemente relacionado com a atividade de requisitos, que inclui escrever os casos de uso, toda a programação orientada a objetos, deve seguir o que foi pedido e formalizado em casos de uso pelo cliente.

Desenvolvimento iterativo, modelagem ágil e um PU (RUP) ágil: Utilizar o processo unificado RUP, como as disciplinas de requisitos e design, aliado a um processo que permite iterações e mudanças que é a abordagem ágil, leve e flexível, sendo este o modelo de processo de desenvolvimento iterativo.

A idéia destes conceitos é construir um projeto aplicando princípios e padrões para criar melhores projetos, seguir iterativamente um conjunto de atividades comuns de análise e projeto, baseando-se em uma abordagem ágil para RUP.

4.1 – PLANO DE AÇÃO

Larman (LARMAN, 2007, p.80), recomenda a definição dos 10 tipos de requisitos:

- Funcional: Que são as características, capacidade e segurança do sistema, estes tipos de requisitos podem ser definidor pelo artefato Caso de Uso ou documentos de regra de domínio (ou Regras de negócio).
- Usabilidade: Fatores humanos, usabilidade como (navegação do sistema e interface com usuário) e documentação obrigatória.
- Confiabilidade: freqüência de falhas, capacidade de recuperação, previsibilidade.
- Desempenho: tempos de resposta, precisão, disponibilidade ou uso de recursos.
- Facilidade de Suporte: Recursos como configurações de sistema, suportem a internacionalização.
- Implementação: limitação de recursos, de linguagens, equipamentos.
- Interfaces: Restrições impostas a interfaces de sistemas externos.
- Operações: Gerenciamento do sistema no ambiente operacional
- Empacotamento: Como setup ou política de processo de change.

Questões Legais: Questões legais envolvidas no software, como licenças de uso e etc.

Como o foco é o processo do desenvolvimento e software, vamos trabalhar somente com as seis primeiras.

Baseado nestes tipos de requisitos propostos por Larman pode preencher os artefatos citados pelo RUP da disciplina de Análise de Requisitos seguindo a ordem de atividades descritas abaixo pelo quadro proposto por Larman.

Na próxima página, há uma seqüência de atividades propostas por Larman divididos entre as iterações e as disciplinas:

Disciplina	**Artefato**	**Concepção** **Semana**	**Elaboração** **4 semanas**	**Elabo 2** **4 Semanas**	**Elab 3** **3** **Semanas**	**Elab 4** **3 semanas**
Requisitos	Modelo de Casos de Uso	Seminário de requisitos de dois dias. A maioria dos casos de Uso é identificada por nome, e estes são resumidos em um parágrafo curto. Pegue 10% dos casos da lista dos mais relevantes para analisar e redigir em detalhe. Esses 10% serão os mais arquiteturalmente importantes, arriscados e de maior valor de negócio	Perto do fim desta iteração faça um seminário de requisitos de dois dias. Obtenha percepções e realimentação a partir do trabalho de implementação, e, então complete 30% dos casos de uso em detalhe.	Perto do fim desta iteração fazer um seminário de requisitos de dois dias. Obtenha percepções e realimentação a partir do trabalho de implementação, e , então complete 50% dos casos de uso em detalhe.	Repita e complete 70% de todos os casos de uso em detalhe	Repita com a Meta de ter 80-90% dos Casos de Uso Esclarecidos E redigidos Em detalhe

Projeto	Modelo de Projeto	Nenhum	Projete pequeno conjunto de requisitos de alto risco significativos para a arquitetura	Repita	Repita	Repita os aspectos significativos de alto risco e do ponto de vista arquitetural deveriam estar estabilizados
Implementação	Modelo de Implementação (Código, etc.)	Nenhum	Implemente Testes	Repita. Nesta etapa 5% do sistema final	Repita nesta etapa, 10% do sistema final está construído.	Repita. Nesta etapa 15% do sistema final está construído.

Quadro 1: Amostra de esforço em requisitos Larman(LARMAN, 2007,p.122)

O que Larman cita como Modelo de Projeto em seu livro, se refere a atividade Realização de casos de Uso pelos papeis Desinger/Arquiteto de Software do RUP. Porque esta atividade inclui a construção de diagramas de seqüência e validando a arquitetura de software proposta.

Larman ainda cria uma seqüência de trabalho dos artefatos de requisitos e projetos percorrendos as iterações das fases como segue no quadro da próxima página (LARMAN, 2007 p.123):

Disciplina	**Artefato** **Iteração →**	**Concep.** **It. 1**	**Elab.** **E1.En**	**Const.** **C1..Cn**	**Trans.** **T1..T2**
Modelagem de Negócio	Modelo de Domínio	X	Inicio		
Requisitos	Visão	Início	Refino		
	Especificações Suplementares	Início	Refino		
	Glossário	Início	Refino		
	Modelo de Caso de Uso	**Início**	**Refino**		
Projeto	Modelo de Projeto	X	Início	Refino	
	Documento de Arquitetura de Software	X	Início		
	Modelo de Dados	X	Início		

Quadro 2 : Amostra de artefatos em requisitos Larman(LARMAN, 2007,p.123)

Após a modelagem o trabalho do Analista de Modelo de Negócio, podemos especificar os casos de uso como propõe Larman no quadro acima. Para escrever Casos de Uso, é necessário se atentar a três fatores, segundo Cockburn(2005, p29), que são:

Técnicas: Que se trata de como construir um caso de uso, Larman(2007, p105) propõe uma escrita dividida em duas colunas, que chaman de formato conversacional, que enfatiza a interação entre os atores e o sistema. Ou o formato convencional proposto pelo RUP (2002), que somente em uma coluna descreve todas as possíveis iterações dos atores com as funcionalidades do sistema separados entre cenário de sucesso principal e cenários secundários, terciários e exceção. De qualquer forma, as técnicas são definições de como pensar, como redigir sentenças e em qual seqüência se deve trabalhar, momento a momento.

Qualidade: Se refere a aceitação do caso de uso para seu propósito. Em CockBurn(2005,p27), Ele cita que um caso de uso não muito detalhado pode ser considerado de boa qualidade, e um caso de uso Muito formalizado e detalhado com um custo muito alto pode não ser, por não haver necessidade de tanta formalidade porque havia muita tolerância na comunicação entre a equipe de desenvolvedores e o usuário final. A frase que Cockburn cita é: “Um tamanho não serve para todos”. E que o grau de Tolerância define a formalidade dos casos de Uso. Larman ainda cita:

> “A discussão completa da tolerância e variação entre projetos é descrita em Software as a Cooperative Game (Cockburn, 2001). Não precisamos da discussão completa para aprender como escrever casos de uso. Precisamos separar técnica de escrita da qualidade dos casos de uso e dos padrões e convenções de projeto”. (LARMAN, 2007, p.115)

Padrões: Que é o que as pessoas do projeto concordam quando escrevem um caso de uso através dos seminários propostos no quadro de plano de ação. Isso pode ser definido como convenções de projeto, que Cockburn cita acima, criando uma série de padrões ajudando na sinergia da equipe (RATIONAL SOFTWARE, 2001).

Seguindo essas instruções de Larman e Cockburn, podemos criar casos de uso de uma boa qualidade para a fase de Design. A partir da construção dos casos de uso, Larman (2007, p159) recomenda partir para a modelagem de domínio.

4.1.1 – ITERATIVIDADE DO USUÁRIO NO PROCESSO

Importante ressaltar que é fundamental para a validade do processo, receber o aceite do cliente quanto aos requisitos, criando um vínculo de iteratividade com o usuário também. Uma forma de fazer isso é através da formalização final dos requisitos, sendo que para Larman (LARMAN, 2007), uma alteração nos requisitos impacta em todo o projeto, quanto mais houver trabalho já concluído no projeto, maior o impacto de uma alteração de um requisito.

Para Cockburn (COCKBURN, 2001), os casos de uso são contratos que detalham a iteração do usuário com o sistema, por esta razão então poderíamos tratar este documento como contrato de requisitos, pedindo que o usuário assine o caso de uso formalizando que ele está de acordo com os termos do documento. O mesmo pode ser feito com o documento de Interface, citado pelo RUP (RUP, 2002) [2], um documento descrevendo os campos e os

elementos da interface do usuário, que poderia servir de base para um protótipo somente contendo interfaces, a ser desenvolvido e apresentado ao usuário para a sua aprovação.

E para Larman (LARMAN, 2007) e Pressman (PRESSMAN, 2001), o processo de desenvolvimento Iterativo deve liberar versões que poderiam ser de entrega final do produto. Cada versão enviada ao cliente então poderia passar pela sua aprovação, uma vez aprovada isso deve ser formalizado em um documento ou carta ou email. Desta maneira podemos definir o que ficou prometido com o usuário e se todas essas promessas foram cumpridas, e que caso haja alguma outra necessidade para o cliente fora as que foram definidas, se configurariam de uma MUDANÇA, e não de uma promessa não cumprida.

4.2 – MODELOS DE DOMÍNIO

Que segundo Larman é:

> "O Passo mais essencialmente orientado a objetos na análise é a decomposição de um domínio em conceitos ou objetos importantes... Um modelo de domínio é uma representação visual de classes conceituais, ou objetos do mundo real, em um domínio..." (LARMAN, 2007, p.159).

Abaixo segue um exemplo de modelo de domínio, definido depois de levantados os requisitos de um Site WebCommerce, que vende produtos online:

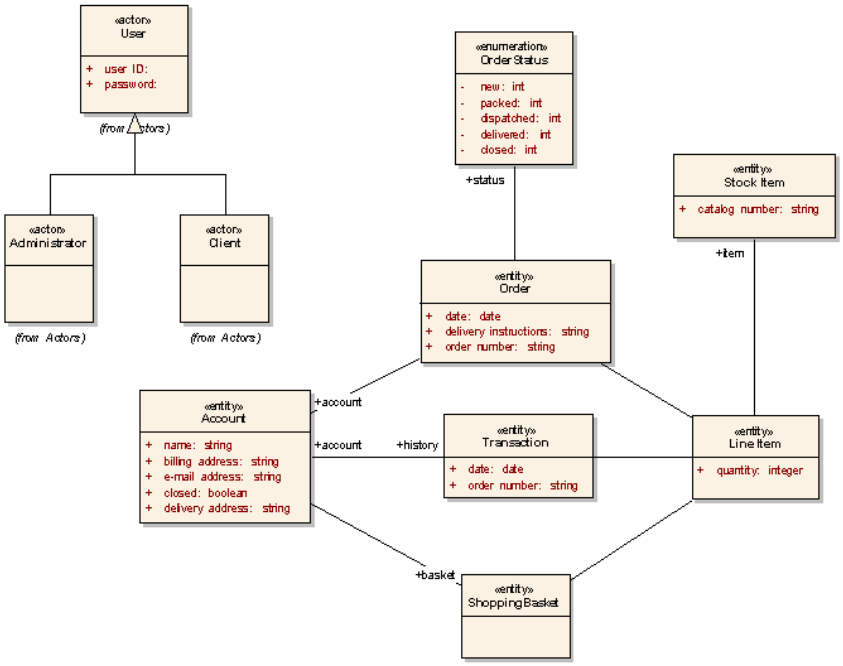

Figura 16: Modelo de domínio

O diagrama, feito pela notação UML, descreve visualmente que segundo os requisitos levantados o sistema deve ter um usuário com ID e senha, e podem ser de dois tipos, Administrador e Client. Que os possíveis status da ordem são New, Packed, Dispatched, delivered e Closed. Cita também que uma ordem tem que conter data instrução de despacho e Código de controle. Define ainda que a ordem E por fim, uma conta contendo os dados de nome, endereço, e-mail status e endereço de entrega, deve conter também um histórico contendo a data e a ordem com os itens transacionados, e estes itens devem conter o código de catalogo. E por sua vez essa conta deve ter um carrinho contendo os itens escolhidos na navegação do site.

Se compararem o texto que escrevi, com o diagrama acima, nota-se que fica muita mais clara a relação dos objetos reais e seus atributos pela diagramação por UML, uma vez bem definidas os requisitos do site.

E neste diagrama de domínio pode nos basear para criar o diagrama de classes do sistema, pois se nos baseamos nos requisitos para criar o modelo de domínio, um diagrama de classes extraído deste modelo estaria bem aderente aos requisitos, cumprindo um passo importante de qualidade de software.

O próximo passo que Larman sugere, é descrever como os atores externos interagem com o sistema passo a passo. E para isso é utilizado o diagrama de seqüência do sistema.

4.3 – DIAGRAMAS DE SEQUÊNCIA DO SISTEMA

O diagrama de seqüência, para Larman (LARMAN, 2007), é a melhor forma de identificar e detalhar os eventos externos de entrada, definindo os eventos do sistema. Desta forma podemos identificar o que o sistema deve fazer em cada evento, e o que deve usar para realizar o evento.

O Diagrama de seqüência deve ser gerado para cada cenário de um caso de uso.

Imagine o seguinte cenário de caso de uso:

UC01 – Criar conta de Usuário
1. O usuário do tipo Client ,seleciona a opção Criar Conta
2. Na funcionalidade criar conta, o usuário entra com os dados: Nome, endereço de pagamento, email e endereço de recebimento.
3. O Usuário então solicita a criação de uma conta com esses dados.
4. Se o usuário email do usuário não existir, então o usuário é criado e o sistema o redireciona para que ele volte as compras.
4a. Se o usuário já existir, o sistema exibe uma mensagem dizendo: “Este email já está cadastrado”

Quadro 3: UC01 – Criar conta de Usuário

Então para criar um diagrama de seqüência aderente aos requisitos e ao diagrama de domínio, relacionamos as entidades encontradas no modelo de domínio no diagrama e fazemos o relacionamento entre eles pelo comportamento do cenário.

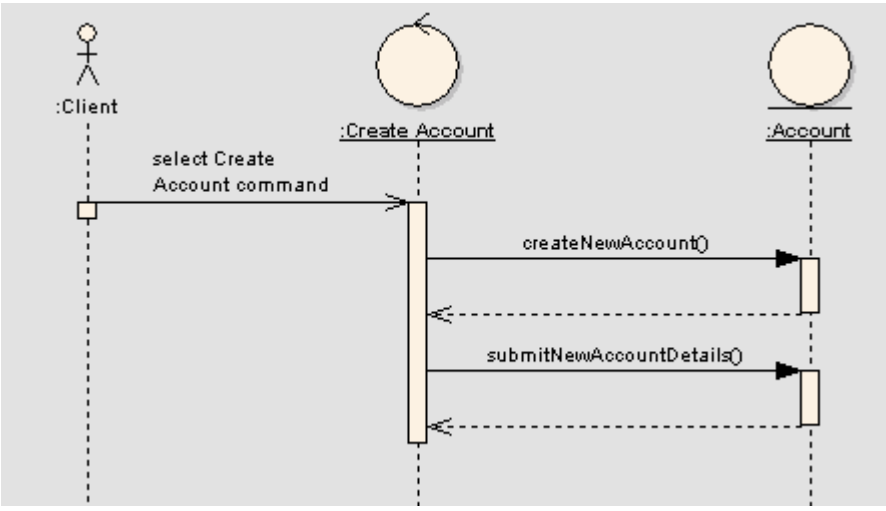

Figura 17: Seqüência do UC01

E identificamos dois eventos de sistema, o createNewAcount, e o submitNewAccount Details. Onde o createNewAcount pode verificar se o usuário é existente, e no caso de positivo, o submitNewAccountDatails o cadastra no sistema.

Iterativamente, Larman nos recomenda a já começar a arquitetura lógica criando a organização de pacotes usando conceito de camadas.

Com definições dos requisitos do sistema, modelo de domínio e eventos, podemos já começar a organizar os objetos dos sistemas em pacotes, e posteriormente em camadas.

Baseado nas informações recebidas até o momento, montei este exemplo de implementação em JavaBeans, tentando prever a estrutura de camadas e pacotes do projeto, porém um esboço que seguindo o modelo iterativo e ágil deve ser refinado até a fase de construção.

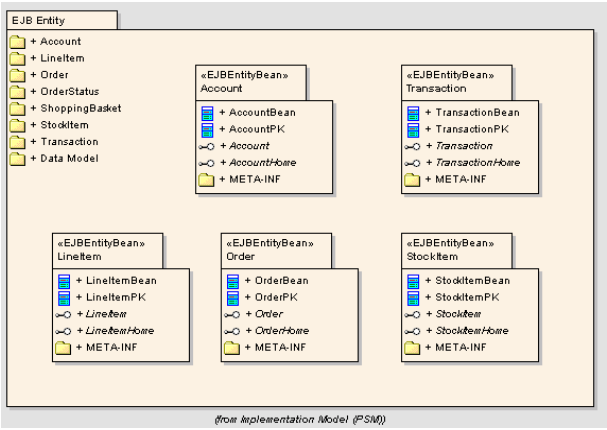

Figura 18: Diagrama de pacotes

Isso somente para termos noção da seqüência do processo de desenvolvimento de software. Lembrando também que foi escolhido Java para este exemplo, porém podemos aplicar o mesmo conceito em outra qualquer linguagem orientada a objetos como C# por exemplo.

4.4 - GRASP – PROJETO DE OBJETOS CO M RESPONSABILIDADE

Este conceito proposto por Larman (LARMAN, 2007), propõe a definição das responsabilidades de cada objeto. Isso já pensando em nível de objetos e não em nível de entidades de negócio do modelo de domínio, como havíamos feito anteriormente. Criar objetos tendo a exata noção do que cada objeto deve fazer é uma prática que nos ajudaria a criar um software de alta coesão e baixo acoplamento.

Alta Coesão: É o termo usado por Larman como LRG (low representational gap) usa para dizer que os objetos do software fazem exatamente o que esta sendo proposto a ele. Como por exemplo, o nosso caso de uso UC01 – Criar Conta. O caso de uso determina que o usuário acesse uma funcionalidade “criar conta”, então se criarmos um software onde o objeto Transaction criasse uma conta, o software não ficaria coeso, pois o certo seria que uma classe Chamada AccountManager, ou Account tivesse esta responsabilidade. Este padrão de projeto é chamado de Especialista na informação.

Baixo Acoplamento: Baixo acoplamento quer dizer o quão fortemente o objeto está conectado, tem conhecimento, ou depende de outros elementos.

Uma alta coesão, que determina bem o que cada objeto possui ou faz, nos ajuda em um baixo acoplamento. Um sistema de acoplamento baixo ajuda com o problema de reduzir o impacto de modificação, retirando todos os acoplamentos desnecessários. Um exemplo disso é a nossa entidade User, temos N métodos que dependem desta classe e preciso fazer uma alteração somente no User que é administrator, em um modelo de baixo acoplamento, eu teria uma subclasse Administrator onde as alterações seriam feitas somente nela, não impactando os métodos que utilizam da classe user. Agora em um acoplamento mais alto, eu não teria especialização de classes, concentrando somente em uma classe só os atributos que seriam de users Client e Adminsitrator. Qualquer alteração que fizesse impactaria em qualquer método que utilizar a classe User.

A recomendação de linguagens orientadas a objetos é intencional justamente por recursos como polimorfismo, herança e encapsulamento, que nos ajuda a reduzir o acoplamento do software, como este no exemplo, se utilizarmos o conceito de herança reduziríamos o acoplamento.

Com os eventos definidos pelo diagrama de seqüência, modelo de domínio maduro e uma arquitetura já pré-definida. Pode-se começar a fazer o modelo de classes, levando em consideração também o conceito GRASP proposto por Larman.

4.5 – MODELOS DE CLASSES

Ao executar a seqüência de atividades de um projeto, podemos chegar a um modelo como este do exemplo, ainda do WebCommerce.

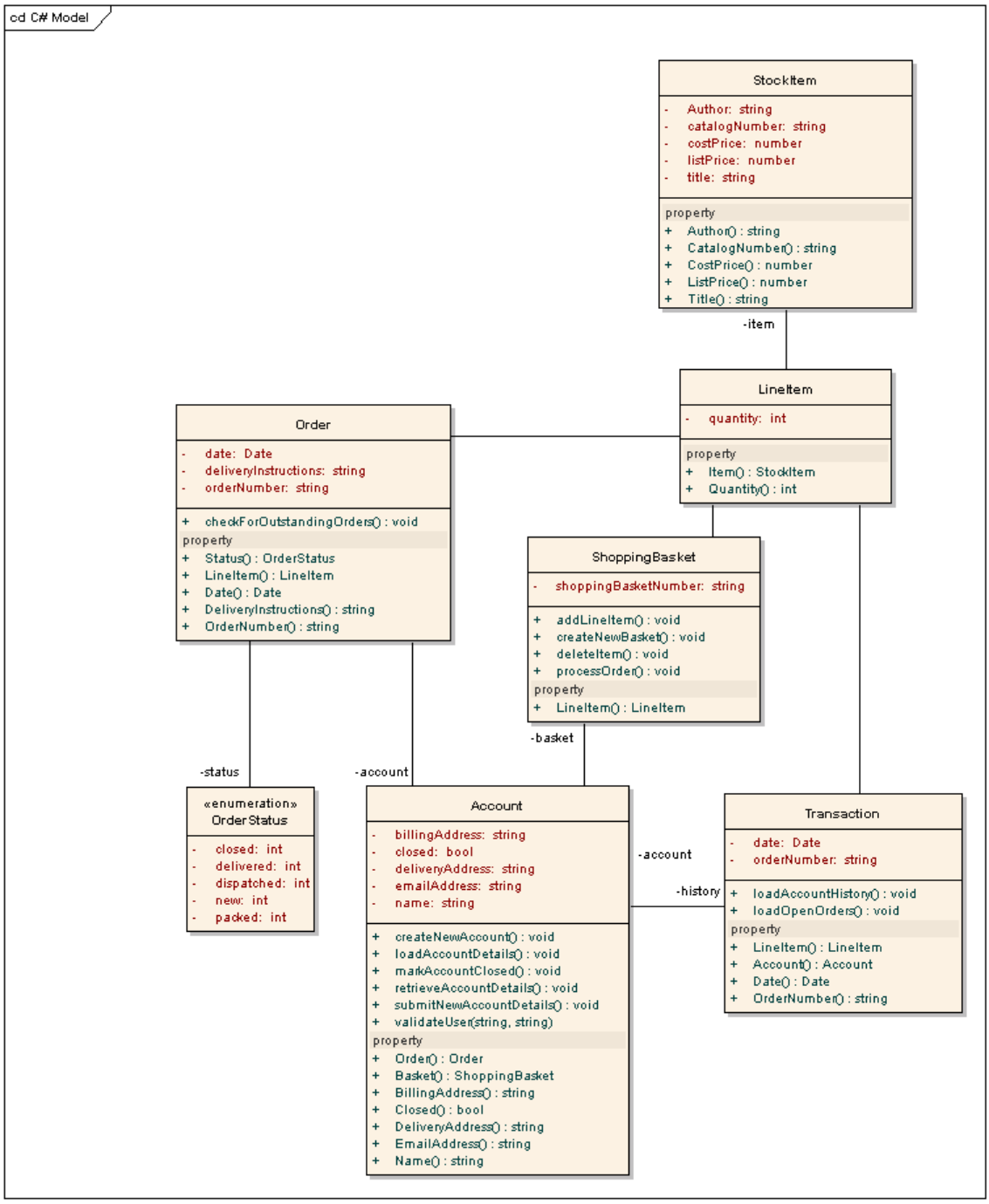

Figura 19: Diagrama de Classes

Depois de capturar os eventos e as relações das entidades com os diagramas de seqüência, podemos encontrar os métodos e cada objeto, como encontramos anteriormente o createNewAccount e o submitNewAccountDetails na Classe Account, quando executamos o diagrama de seqüência do caso de uso UC01-Criar Conta. E assim é feito para todos os casos de uso levantados na disciplina de requisitos.

Uma vez detalhado o diagrama de classe, e os subsistemas onde eles vão permanecer (no caso do .Net se diz namespace e Java Package). Podemos já desenhar diagramas de seqüência aderentes ao modelo de classes atual, á arquitetura atual e aderente aos requisitos levantados. Com um diagrama de seqüência de um esboço da tela de protótipo, podemos construir uma Especificação de software que poderia dar suporte á equipe de desenvolvimento que desconhece do negócio, como o modelo do Anexo I.

Craig Larman (LARMAN, 2007), Faz um relacionamento entre os artefatos do RUP, onde podemos comparar com o relacionamento de artefatos do RUP mencionado no segundo capítulo.

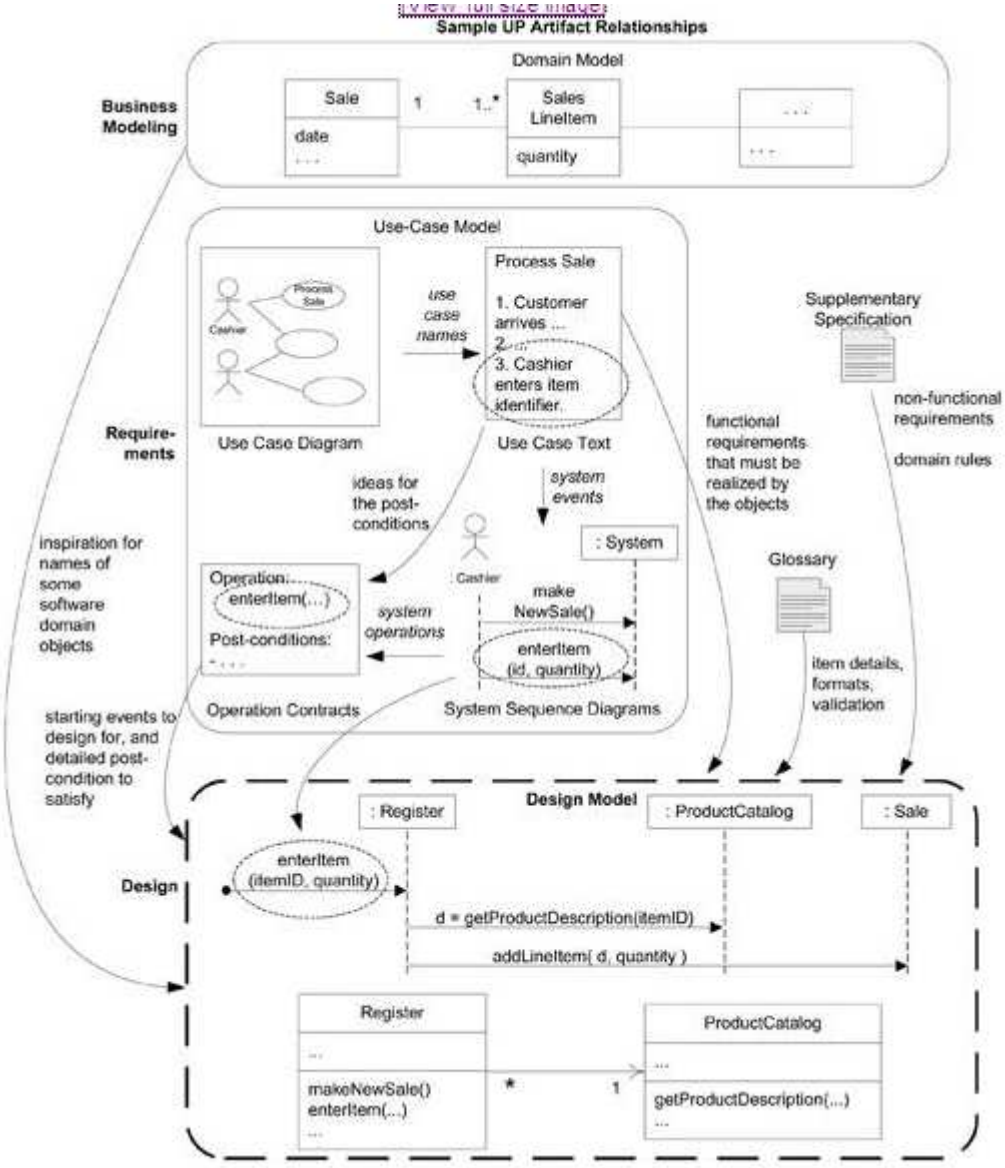

Figura 20: Dependência entre artefatos (LARMAN, 2007, p337),

Notemos que no painel Design Model, onde nos encontramos que é a definição de Design de Sistema, que para chegarmos até aqui, é necessário passar pelo Modelo de domínio, que é iniciado na Modelagem de negócio, Modelo de caso de Uso, que é iniciado na fase de requisitos, depois aplicamos um diagrama de seqüência nos cenários de caso de uso para encontrarmos as operações de cada objeto, e agora nos encontramos no Design do sistema já baseados em Objetos o modelo de domínio refinado e arquitetura definida.

Se compararmos os dois modelos, Craig Larman enfatiza a orientação a objetos e constrói somente os artefatos necessários para um design de qualidade passando pelas fases do RUP. Tem preocupação na aderência de Design junto aos requisitos e o modelo de negócio, e ao mesmo tempo, com a qualidade da construção de software, com orientação a objetos e iteração de fases, para refinamento dos artefatos quando se trabalha envolvido com outra disciplina, como por exemplo, a de Requisitos e Design, no modelo proposto anteriormente, o modelo de domínio foi refinado até virar uma estrutura de classes que poderiam ser utilizada para o design bem aderente tecnicamente e funcionalmente, abaixo segue um exemplo de como ficaria um diagrama de seqüência de preenchimento de combobox em uma página web, com o modelo de classes refinadas e com uma arquitetura definida em camadas seguindo a notação UML. Informações como estas, e mais um modelo de Banco de dados pronto, já é suficiente para começar uma implementação bem aderente à arquitetura e aos requisitos.

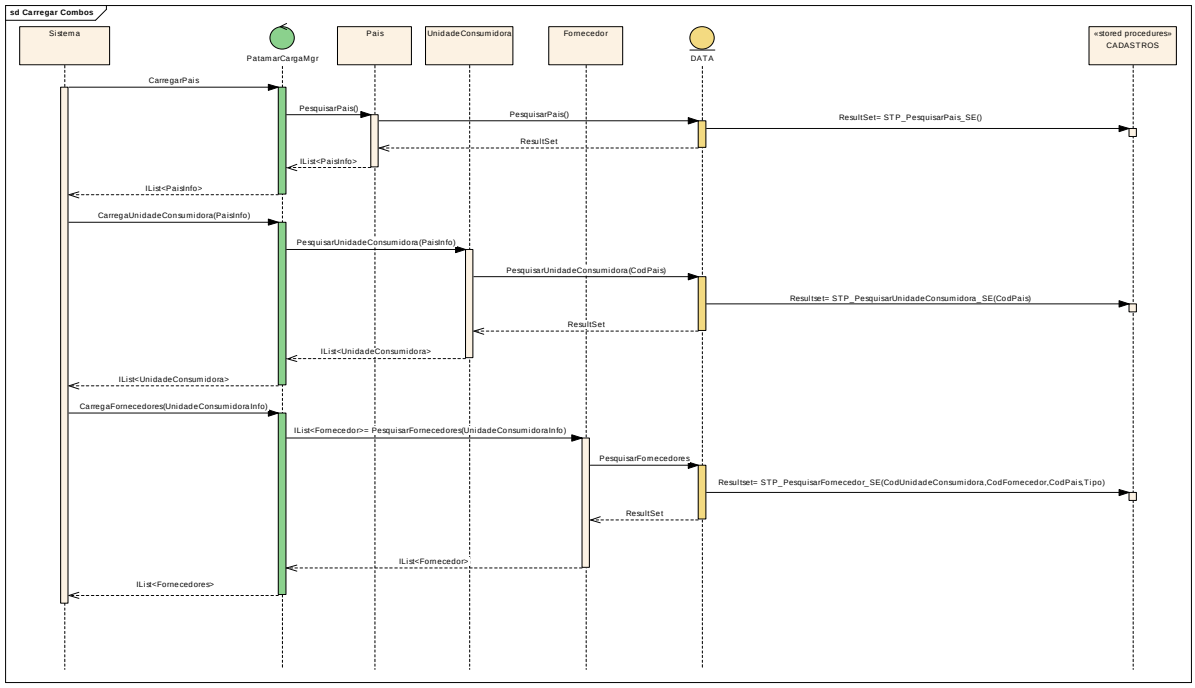

Figura 21: Diagrama de seqüência final

Outra preocupação de Larman quanto à qualidade, é o desenvolvimento dirigido por Teste e Refatoração, mais um conceito trazido das metodologias Ágeis, mais especificamente, XP (Extreme Programming). A refatoração consiste em “fazer o teste antes”, ou seja, antes de construir uma parte do software, já preparar um plano de teste aderente ao que propõe os requisitos, ajudando desta maneira então a controlar a qualidade funcional por meio de cenários de teste Caixa Branca. E um modelo de teste técnico por meio de testes unitários. (LARMAN, 2007)

4.6 – IMPORTÂNCIA DE UMA FERRAMENTA CASE

A esta altura, com vários esboços de caso de uso, modelo de domínio e diagramas de seqüência, se tornaria um trabalho árduo transformar tudo, exatamente igual, para código. Por isso então existem ferramentas case com a funcionalidade de engenharia reversa, com ela podemos manter as classes atualizadas para efeito de documentação ou alteração de design. Ferramentas como o Enterprise Manager possui essa função que possibilita tanto criar as classes em C# ou Java à partir de um diagrama, ou até mesmo importar as classes já programadas trazendo as classes para diagramação no EA (Enterprise Arquitect, a ferramenta usada para a diagramação neste trabalho).

Lembrando que segundo (LARMAN, 2007) para termos de agilidade, é muito mais importante comunicar que documentar, porque com a documentação e alta formalidade, perdemos flexibilidade nas alterações,

perdendo assim também com iteratividade entre as equipes, uma vez que a cada iteração o trabalho deve ser evoluído segundo o processo icremental.

Com uma ferramenta case, com engenharia reversa, ganhamos agilidade entre as equipes de design e implementação, uma vez que é mais ágil modificar um diagrama e depois converte-lo em código usando a ferramenta, que documentando este diagrama em um formato de texto e passando para o implementador a cada alteração que for notada como refino, seguindo o plano de ação proposto por Larman.

CAPÍTULO 5 – CONSIDERAÇÕES FINAIS

Neste trabalho foram apresentados os principais pontos de um processo de desenvolvimento de software iterativo, na visão do Analista de Sistema. Assim como a fundamentação teórica que levanta a necessidade de se ter um processo de desenvolvimento, e o porquê da iteratividade. E foram descritas também, algumas técnicas de organização do trabalho e modelagem de sistema.

Como pode ser visto no estudo, o processo de desenvolvimento de software é fundamental para se ter um software de qualidade, e como pontos centrais deste processo foram citados a iteratividade que é a maior responsável pela sintonia entre os profissionais da equipe e com o próprio cliente, ou usuário final. Propondo um modelo de processo procurei preencher as principais lacunas no desenrolar de um projeto de software na visão do Analista.

Por ser um assunto tão abrangente, que se analisarmos papel a papel do RUP, nota-se então que muito deve ser trabalhado dentro deste assunto, o Processo de Desenvolvimento de Software, que tem sido atuado mais seriamente nos últimos anos por conta do aumento de necessidade por softwares e estes por conta do aumento da necessidade de qualidade. E isso faz deste assunto, um assunto muito interessante que merece um cuidado especial, pois através deste processo podemos tornar uma empresa mais madura e competitiva na produção de softwares, e também, porque não, um aumento na qualidade de vida dos atuantes da área de TI com menos horas extras e mais planejamento.

REFERÊNCIAS BIBLIOGRAFICAS

MEIO IMPRESSO

RUP. The ***Rational Unified Process***. IBM Rational, versão 2002.

LARMAN, C. ***Utilizando UML e Padrões***. Porto Alegre: Bookman, 2007, 696 p.

COCKBURN, A. ***Escrevendo casos de Uso eficazes***, Porto Alegre: Bookman, 2005, 15-90 p.

COOCH, G. UML: ***Guia do Usuario***, Rio de Janeiro, 2005 , 474 p.

PRESSMAN, R., ***Software engineering: a practitioner's approach*** 5th ed , New York, 2001, 11-256 p.

BARY , B. **Anchoring *the Software Process***, IEEE Software, New York, 1996, pp. 73-82.

RATIONAL SOFTWARE. ***Best Practices for Software***. IBM Rational, Rev 2001.

IVAR JACOBSON, GRADY COOCH, and JIM RUMBAUGH, ***Unified Software Development Process***, Addison-Wesley,1999.

MEIO ELETRÔNICO

[1] **WIKIPEDIA**., ***Engenharia do Software***, Disponível em: <http://pt.wikipedia.org/wiki/Engenharia_do_software>. Acesso em: 28 out 2009.

[2] WIKIPEDIA., ***RUP***, Disponível em: <http://pt.wikipedia.org/wiki/IBM_Rational_Unified_Process>. Acesso em: 5 nov 2009.

[3] WIKIPEDIA., ***Orientação a Objetos***, Disponível em: <http://pt.wikipedia.org/wiki/Orienta%C3%A7%C3%A3o_a_objeto>. Acesso em: 1 out 2009.

[4] HO-WON JUNG and SEUNG-GWEON KIM, ***Measuring Software,Product Quality***: A Survey of ISO/IEC 9126, IEEE Software, Seul, 2004, p5