



Universidade Estadual de Maringá
Centro de Tecnologia
Departamento de Engenharia de Produção

**Uma Proposta de Melhoria no Processo de Software em uma
MPE**

Anderson dos Santos Lima

TCC-EP-09-2011

Universidade Estadual de Maringá
Centro de Tecnologia
Departamento de Engenharia de Produção

**Uma Proposta de Melhoria no Processo de Software em uma
MPE**

Anderson dos Santos Lima

TCC-EP-09-2011

Trabalho de Conclusão de Curso apresentado como requisito de avaliação no curso de graduação em Engenharia de Produção na Universidade Estadual de Maringá – UEM.

Orientador(a): Prof.^a: MSc. Gislaine Camila Lapasini Leal

**Maringá - Paraná
2011**

EPÍGRAFE

*Ser o homem mais rico do cemitério não me interessa.
Ir para a cama à noite dizendo que fizemos
algo maravilhoso, isso importa para mim...*
Steve Jobs

AGRADECIMENTOS

Primeiramente agradeço a Deus, pois Ele é a razão do meu viver e foi Quem me proporcionou todos os motivos pelos quais poderei agradecer às pessoas que, direta ou indiretamente, estiveram comigo nesta etapa de minha vida...

Agradeço à minha família pelo apoio dado desde o início – mesmo antes do ingresso na universidade – e por sempre apostarem em mim, confiando que eu alcançaria todos os meus objetivos. Obrigado mãe, pai e irmã...

Agradeço especialmente ao Jean, o melhor amigo [*meu irmão*] que encontrei na universidade e que sempre fará parte da minha vida. Obrigado pelos conselhos, pela amizade, pelo companheirismo, pelas lasanhas; obrigado simplesmente por estar ao meu lado sempre que precisei de um amigo verdadeiro. Simples palavras não podem expressar o seu valor para mim...

Agradeço ao Kleber, um dos melhores amigos que já fiz na vida e do qual usufruí de sua casa incontáveis vezes [*risos*]. Obrigado pelos ótimos momentos que passamos juntos, pela minha primeira viagem para fora do estado [*incrível, eu sei*], pelo irmão que você é pra mim...

Agradeço ao Douglas, em quem encontrei um companheiro fiel durante toda a jornada acadêmica, e que se mostrou um ótimo parceiro nos trabalhos em grupo e principalmente, agradeço por sua amizade sempre fiel e sincera...

Agradeço ao Renan por sua amizade, seu companheirismo e sinceridade. Obrigado por me emprestar as dependências de sua casa várias vezes [*risos*] e pelos ótimos momentos que passamos juntos... aquela música virou sua trilha sonora cara e toda vez que ouvi-la lembrarei de ti...

Agradeço ao Luiz Fernando pela amizade verdadeira ao longo destes 5 anos, nos quais sempre me escutou (é, na verdade ele escuta muito bem) e me deu bons conselhos.

Agradeço a todos os colegas de sala, tanto aqueles que estiveram comigo em algum destes cinco anos e não estão mais por algum motivo, como os que continuam juntos. Obrigado pelos ótimos momentos que passamos juntos...

Agradeço à professora Camila [*para mim particularmente, sempre será famosa*], minha orientadora deste trabalho [*e do estágio, e de muitos outros trabalhos*], que sempre acreditou em mim mesmo quando eu achava que não daria certo. Obrigado pela paciência, pelas dicas, pelos conselhos. Enfim, obrigado pela ótima conselheira e amiga que se tornou para mim...

Agradeço a todos os professores que de alguma maneira contribuíram para meu crescimento como estudante e como pessoa... Obrigado (especialmente) Welington, Camila Leal, Márcia Samed, Adriana, Yandre, Francielle...

Agradeço a todos os funcionários da PEN, a qual me proporcionou minha primeira experiência profissional. Obrigado Regina, Prof.^a. Alexandra, José Luiz e Cleverson...

Agradeço à M.K.I. Informática LTDA ME – na pessoa de seu Diretor Geral, Marcelo Siena – por ter me proporcionado todos os meios pelos quais eu pude desenvolver e concluir este trabalho.

Agradeço aos amigos e irmãos que fizeram parte da minha vida ao longo desses cinco anos, especialmente: Renato, Vinícius, Gobetti, Eduardo, Alexandre, Diego, Carlos Augusto, Paulo Henrique, Josef, Jonata, Raquel, Rebeca, Giovanna, Juliana, Duda, Hayla, Edilaine, Denise e Danieli.

RESUMO

O Desenvolvimento de Software se tornou uma atividade chave e crítica para muitas empresas. Um produto de software com qualidade requer um processo de desenvolvimento também com qualidade. Para isso, metodologias de desenvolvimento podem ser empregadas, como o RUP, o XP ou o AUP. A complexidade de algumas metodologias, entretanto, requer que elas sejam adaptadas à realidade da organização. A empresa M.K.I. Informática LTDA ME não dispunha de um processo de desenvolvimento formalizado, o que acarretava muitos problemas como estouro nos prazos, bugs constantes e insatisfação por parte dos clientes. O objetivo deste trabalho foi formalizar um processo de desenvolvimento, pautado em uma metodologia dentre as exploradas. O RUP foi escolhido como a base para o processo da empresa, sendo adaptado com os princípios ágeis do XP. O principal motivo de se formalizar um processo consiste na melhoria da qualidade do produto final. A implantação de um modelo de desenvolvimento requer, dentre outras coisas, comprometimento de toda equipe. A principal dificuldade consiste em convencer as pessoas que o maior tempo deve estar pautado no planejamento das atividades. O desenvolvimento ágil colabora para que somente atividades que agreguem valor ao produto sejam executadas, diminuindo, dessa maneira, tempo de desenvolvimento desnecessário.

Palavras-chave: Desenvolvimento de Software; ISO/IEC 12207; RUP – *Rational Unified Process*; XP – *Extreme Programming*; AUP-*Agile Unified Process*; MPS-BR; CMMI; Desenvolvimento Ágil, Melhoria Contínua.

ABSTRACT

The software development has become a key and critical activity for many companies. A software product with quality requires a development process also with quality. Thereunto, development methodologies can be used/applied, such as RUP, XP or AUP. The complexity of some methods, however, requires them to be adapted to the organization's reality. The company M.K.I. Informática LTDA ME didn't have a formalized development process, which caused many troubles such as term overflow, bugs and constant customers dissatisfaction. The objective of this study was to formalize a development process, based on a methodology among the exploited. The RUP was chosen as the business process base, being adapted to the agile principles of XP. The main reason to formalize a process is to improve final product's quality. A development model implementation requires, among other things, the involvement of the whole team. The main difficulty is to convince people that most of the time must be used in the activities planning. Agile development collaborates with only value-added activities to run, reducing in this way, unnecessary development time.

SUMÁRIO

EPÍGRAFE	iii
AGRADECIMENTOS	iv
LISTA DE ILUSTRAÇÕES	ix
LISTA DE ABREVIATURAS E SIGLAS	x
1. INTRODUÇÃO	1
1.1 Justificativa	1
1.2 Definição e delimitação do problema	2
1.3 Objetivos.....	3
1.3.1 <i>Objetivo geral</i>	3
1.3.2 <i>Objetivos específicos</i>	3
1.4 Metodologia.....	3
1.5 Organização do Trabalho.....	4
2. REVISÃO DE LITERATURA.....	5
2.1 Melhoria Contínua	5
2.2 Desenvolvimento de Software	7
2.3 Processo de Software	11
2.3.1 <i>RUP – Rational Unified Process</i>	12
2.3.2 <i>XP – Extreme Programming</i>	15
2.3.3 <i>AUP – Agile Unified Process</i>	23
2.4 Melhoria do Processo de Software	25
2.4.1 <i>MPS-BR</i>	26
2.4.2 <i>CMMI</i>	27
3. PROPOSTA DO PROCESSO.....	29
3.1 Caracterização da Empresa.....	29
3.2 Processo de Desenvolvimento	30
3.3 Diagnóstico de Problemas	32
3.4 Estruturação do Processo de Desenvolvimento	35
3.4.2 <i>Papéis</i>	36
3.4.2 <i>Disciplinas</i>	38
3.4.3 <i>Relação entre as fases do RUP e as disciplinas do modelo proposto</i>	68
4. CONSIDERAÇÕES FINAIS.....	70
4.1 Contribuições	70
4.2 Dificuldades e Limitação	71
4.3 Trabalhos Futuros	71
REFERÊNCIAS	72
APÊNDICE	77

LISTA DE ILUSTRAÇÕES

FIGURA 1 - PROCESSOS DA NORMA ISO/IEC 12207	8
FIGURA 2 - FASES E DISCIPLINAS DO AUP	24
FIGURA 3 - ORGANOGRAMA DA EMPRESA	30
FIGURA 4 - FLUXOGRAMA DA IMPLEMENTAÇÃO DE NOVOS MÓDULOS	31
FIGURA 5 - GRÁFICO DO Nº DE BUGS POR VERSÃO	34
FIGURA 6 - DIAGRAMA DE ATIVIDADES.....	39
FIGURA 7 - DIAGRAMA DE PACOTES: GERÊNCIA DE PROJETOS	40
FIGURA 8 - DIAGRAMA DE CASOS DE USO: GERÊNCIA DE PROJETOS	41
FIGURA 9 - DIAGRAMA DE CLASSES: GERÊNCIA DE PROJETOS	42
FIGURA 10 - DIAGRAMA DE ATIVIDADES: GERÊNCIA DE PROJETOS	42
FIGURA 11 - DIAGRAMA DE PACOTES: REQUISITOS.....	44
FIGURA 12 - DIAGRAMA DE CASOS DE USO: REQUISITOS	45
FIGURA 13 - DIAGRAMA DE CLASSES: REQUISITOS	46
FIGURA 14 - DIAGRAMA DE ATIVIDADES: REQUISITOS	47
FIGURA 15 - DIAGRAMA DE PACOTES: MODELAGEM DE NEGÓCIO	48
FIGURA 16 - DIAGRAMA DE CASOS DE USO: MODELAGEM DE NEGÓCIO.....	49
FIGURA 17 - DIAGRAMA DE CLASSES: MODELAGEM DE NEGÓCIO.....	49
FIGURA 18 - DIAGRAMA DE ATIVIDADES: MODELAGEM DE NEGÓCIO.....	50
FIGURA 19 - DIAGRAMA DE PACOTES: ANÁLISE E PROJETO.....	51
FIGURA 20 - DIAGRAMA DE CASOS DE USO: ANÁLISE E PROJETO	53
FIGURA 21 - DIAGRAMA DE CLASSES: ANÁLISE E PROJETO	54
FIGURA 22 - DIAGRAMA DE ATIVIDADES: ANÁLISE E PROJETO	55
FIGURA 23 - DIAGRAMA DE PACOTES: IMPLEMENTAÇÃO.....	56
FIGURA 24 - DIAGRAMA DE CASOS DE USO: IMPLEMENTAÇÃO	58
FIGURA 25 - DIAGRAMA DE CLASSES: IMPLEMENTAÇÃO	59
FIGURA 26 - DIAGRAMA DE ATIVIDADES: IMPLEMENTAÇÃO	59
FIGURA 27 - DIAGRAMA DE PACOTES: GERENCIAMENTO DE DOCUMENTAÇÃO PARA OS CLIENTES	60
FIGURA 28 - DIAGRAMA DE CASOS DE USO: GERENCIAMENTO DE DOCUMENTAÇÃO PARA OS CLIENTES	61
FIGURA 29 - DIAGRAMA DE CLASSES: GERENCIAMENTO DE DOCUMENTAÇÃO PARA OS CLIENTES	62
FIGURA 30 - DIAGRAMA DE ATIVIDADES: GERENCIAMENTO DE DOCUMENTAÇÃO PARA OS CLIENTES	62
FIGURA 31 - DIAGRAMA DE PACOTE: TESTE	63
FIGURA 32 - DIAGRAMA DE CASOS DE USO: TESTE.....	64
FIGURA 33 - DIAGRAMA DE CLASSES: TESTE.....	65
FIGURA 34 - DIAGRAMA DE ATIVIDADES: TESTE.....	65
FIGURA 35 - DIAGRAMA DE PACOTES: AMBIENTE	66
FIGURA 36 - DIAGRAMA DE CASOS DE USO: AMBIENTE	67
FIGURA 37 - DIAGRAMA DE ATIVIDADES: AMBIENTE	68
FIGURA 38 - FASES DO RUP E DISCIPLINAS DO MODELO PROPOSTO.....	69

LISTA DE ABREVIATURAS E SIGLAS

AMDD	<i>Agile Model Driven Development</i>
AUP	<i>Agile Unified Process</i>
CMMI	<i>Capability Maturity Model Integration</i>
ISO	<i>International Organization for Standardization</i>
MPS-BR	Melhoria de Processos do Software Brasileiro
RUP	<i>Rational Unified Process</i>
SOFTEX	Sociedade para Promoção da Excelência do Software Brasileiro
TQM	<i>Total Quality Management</i>
UML	<i>Unified Modeling Language</i>
XP	<i>Extreme Programming</i>

1. INTRODUÇÃO

O processo de software é definido como um conjunto ordenado de atividades para o gerenciamento, desenvolvimento e manutenção de software, e deve estar alinhado com as condições organizacionais (Fuggetta¹, 2000 *apud* Leal, 2010).

Conforme destaca Nascimento (2008), a demanda crescente por softwares nas mais variadas áreas tem feito com que as empresas se preocupem cada vez mais com o desenvolvimento de software com qualidade, e utilizando as novas tecnologias, a melhoria de processos, a capacitação dos colaboradores, dentre outros meios, elas conseguem alcançar este objetivo.

Nos últimos anos, tem sido observado um aumento considerável de empresas brasileiras envolvidas em programas de melhoria de processo de software. Estes programas têm sido planejados e executados pela necessidade de melhoria no desenvolvimento de software e pela demanda crescente no mercado por empresas reconhecidas em níveis de maturidade nos principais modelos existentes no mercado (HILGERT *et al.*, 2008).

Um sistema de qualidade requer um processo de desenvolvimento também de qualidade. Para isso, são necessários diversos esforços para minimização e correção dos erros que eventualmente possam surgir. Além disso, um processo formalizado e padronizado garante que todos que interagem no sistema tenham uma sequência lógica a ser seguida e entendam a relação de dependência das atividades.

Com base no exposto, o presente trabalho tem como tema a melhoria no processo de desenvolvimento de software em uma microempresa, que atua no ramo de sistemas voltados para provedores de internet.

1.1 Justificativa

A justificativa para este trabalho se dá pela necessidade de formalizar um processo de desenvolvimento para o produto da empresa, haja visto que atualmente isto não existe, o que acarreta muitas inconsistências e retrabalhos, muitos dos quais podem ser evitados com a

¹ Fuggetta, A. *Software process: a roadmap*. In: ICSE '00: Proceedings of the Conference on The Future of Software Engineering, New York, NY, USA: ACM, 2000, p. 25-34.

formalização de um processo. Um exemplo disso está na manutenção de módulos já existentes onde, muitas vezes, são criadas instabilidades em outros módulos que dependem do mesmo, pela alteração de variáveis, de escopo, dentre outros. Se houvesse um processo formalizado, todas as dependências entre módulos estariam documentadas e assim, seria mais fácil identificar todas as relações e necessidades de alterações em ambos.

Além disso, um processo formalizado contribui na comunicação e no entendimento de cada atividade por todos da empresa, sendo mais fácil a integração de novas pessoas em determinada equipe ou projeto – na equipe de desenvolvimento, por exemplo.

1.2 Definição e delimitação do problema

A empresa M.K.I. Informática LTDA ME tem como um de seus serviços chave, o desenvolvimento de um sistema de gerenciamento voltado para provedores de internet. Não focando seu negócio em “comercializar vários softwares”, no momento a empresa trabalha desenvolvendo novos módulos para o sistema já existente, bem como oferecendo treinamentos e suporte técnico do mesmo. As atividades do desenvolvimento se concentram basicamente em três: novos módulos, alterações em módulos existentes e correções em módulos existentes. Estas três atividades são realizadas pelo setor de Desenvolvimento. As demais atividades, como treinamento e suporte, são desenvolvidas pelo setor de Suporte Técnico e não fazem parte do escopo do trabalho em questão.

No momento, a empresa não dispõe de um processo formal de desenvolvimento. Dessa maneira, constantemente se observam problemas como: estouro nos prazos; lançamento de novas aplicações ou módulos sem os devidos testes, ocasionando inconsistências; atrasos em aplicações que deveriam ser prioritárias; perda de produtividade por mau uso do reuso (por exemplo, reaproveitamento de códigos de funções que implementam soluções similares); estes problemas impactam na qualidade e geram insatisfações por parte dos clientes.

Ao longo da história os clientes da empresa têm aprovado e indicado o sistema. Contudo, os erros acima mencionados – bem como outros – acabam gerando descontentamento nos clientes e ainda, uma perda de qualidade do software, o que não é almejado por empresa alguma.

1.3 Objetivos

1.3.1 Objetivo geral

O objetivo principal deste trabalho consiste em formalizar e melhorar um processo de desenvolvimento de software de uma microempresa de serviços que possui um sistema de gerenciamento voltado para provedores de internet.

1.3.2 Objetivos específicos

Como objetivos específicos, tem-se:

- Analisar como o processo de desenvolvimento da empresa acontece atualmente;
- Identificar os pontos críticos do processo;
- Selecionar um processo de desenvolvimento de software que possa ser adaptado à realidade da empresa;
- Formalizar um processo de desenvolvimento através da modelagem;
- Comunicar o processo à empresa.

1.4 Metodologia

Do ponto de vista de sua natureza, a pesquisa é considerada do tipo Aplicada, uma vez que objetiva gerar conhecimentos para aplicação prática, estando diretamente ligada à solução de um problema específico (Silva e Menezes, 2005). Do ponto de vista de seus objetivos, a pesquisa é do tipo Explicativa, uma vez que visa a identificação de fatores que contribuem para determinado fenômeno. Quanto aos procedimentos técnicos, ela é considerada uma Pesquisa-Ação, pois está diretamente relacionada com a solução de um problema específico e, além disso, os representantes da situação estão envolvidos de algum modo (GIL², 1991 *apud* SILVA e MENEZES, 2005).

As etapas que nortearam o trabalho foram:

- Revisão Bibliográfica sobre o tema proposto, que tratará basicamente de três assuntos: Melhoria Contínua, Processo de Software e Melhoria do Processo de Software;

² GIL, Antonio Carlos. *Como elaborar projetos de pesquisa*. São Paulo: Atlas, 1991.

- Caracterização da empresa, como forma a entender sua cultura e como ela está organizada;
- Caracterização do processo utilizado atualmente pela empresa;
- Levantamento dos pontos críticos, através da documentação disponível e entrevistas com os responsáveis pelo setor;
- Proposta de um processo compatível com as necessidades e características da empresa;
- Formalização do Processo;
- Comunicação do Processo.

1.5 Organização do Trabalho

Neste capítulo foram descritos os objetivos deste trabalho e, também, foram definidos e delimitados os problemas a serem estudados. Também, foram apresentados o contexto e a metodologia que nortearam o desenvolvimento do mesmo. O restante do trabalho está organizado da seguinte forma:

- Capítulo 2: apresenta os principais conceitos utilizados no desenvolvimento deste trabalho, a saber: Melhoria Contínua, Desenvolvimento de Software, Processo de Software e Melhoria do Processo de Software;
- Capítulo 3: descreve a empresa que é o objeto do estudo em questão, o processo de desenvolvimento atual e o diagnóstico de problemas. Aqui também é apresentado o modelo proposto, em termos de disciplinas, atividades, papéis e artefatos;
- Capítulo 4: discute as principais contribuições e limitações do trabalho e também apresenta os trabalhos futuros.

2. REVISÃO DE LITERATURA

Esta seção apresenta os conceitos que nortearão o desenvolvimento deste trabalho, sendo eles: Melhoria Contínua, Desenvolvimento de Software, Processo de Software e Melhoria do Processo de Software.

2.1 Melhoria Contínua

Conforme Caffyn³ e Bessant (1996, *apud* Valadao e Turrioni, 2010), a melhoria contínua é um processo que engloba toda empresa, focado na inovação incremental e contínua. Isso significa que ela não está relacionada apenas à alta direção da organização, mas sim a todos os setores, desde o chão de fábrica até os mais altos cargos.

Ela também pode ser entendida como pequenos passos que acontecem continuamente, devendo fazer parte da cultura da empresa e sendo executados de forma natural e frequente pelos funcionários que buscam melhorar suas práticas rotineiramente. Ela pode ser vista como um processo que apoia os demais processos de negócio, trazendo benefícios isoladamente e em curto prazo, mas que, cumulativamente, levam sensíveis melhorias às empresas (AGOSTINETTO, 2006).

Rodrigues *et al.* (2010) ressaltam a importância da melhoria contínua dos processos fazer parte da cultura da organização, uma vez que através dela é possível reduzir custos, evitar retrabalhos, aumentar a produtividade bem como a satisfação dos clientes e, também, a motivação da própria empresa como um todo. Além disso, quando o conhecimento está bem direcionado, ele pode aumentar a flexibilidade e a eficiência da organização.

A flexibilidade pode ser entendida como o resultado de todo o comprometimento e sabedoria acumulada ao longo do tempo e isso facilita a mudança nas estratégias e incentiva a busca pela melhoria contínua (UEDA e NOGUEIRA, 2010).

De uma forma abrangente, a melhoria contínua deve ser iniciada em cada setor, para alcançar toda a empresa, ao longo do tempo.

³ CAFFYN, S.; BESSANT, J. *A capability-based model for continuous improvement*. Proceedings of 3th International Conference of the EUROMA. London, 1996.

A melhoria contínua foi inicialmente demonstrada no Japão e posteriormente pelas empresas americanas e parte da Europa pelos programas TQM (*Total Quality Management*), razão pela qual está diretamente ligada à Qualidade (AGOSTINETTO, 2006). Enquanto Kume⁴ (1993, *apud* Oprime *et al.*, 2009) detalha a aplicação de modo sistemático (da melhoria contínua) como um conjunto de ferramentas e técnicas para solução de problemas, Imai⁵ (1986, *apud* Oprime *et al.*, 2009) introduz os conceitos japoneses de melhoria contínua através da filosofia do Kaizen, com eliminação de perdas e ênfase na melhoria incremental. Campos⁶ (1992, *apud* Soares, 2010) acrescenta que esta filosofia de trabalho tem como objetivo eliminar as causas fundamentais que ocasionam resultados indesejados e, a partir da introdução de novos conceitos e ideias, estabelecer novos níveis de controle para os processos.

Mas, como iniciar um processo de melhoria contínua? Paz (2011) destaca alguns desafios que os pequenos empresários têm que enfrentar quando desejam implantar a melhoria contínua em suas empresas, sendo eles:

- Desenvolver um modelo de Gerenciamento participativo, o que significa mais envolvimento e participação dos colaboradores da empresa, bem como maior comprometimento da equipe.
- Mostrar aos gestores da empresa que o planejamento é a porta principal do sucesso de toda e qualquer atividade. Dessa maneira, todas as ações devem ser primeiramente planejadas, evitando ações impensadas que, na maioria das vezes, geram altos custos, desperdícios e falhas.
- Garantir o aprimoramento dos processos e, por conseguinte, dos produtos e serviços da empresa. Para isso, é preciso que os ambientes interno e externo da empresa sejam monitorados e analisados, tomando-se medidas para sanar todos os erros, falhas e desperdícios.
- Deixar de lado a informalidade, o que configura passar a solucionar problemas e tomar decisões de modo sistemático. Ela prejudica a velocidade, a agilidade, a competência e a assertividade das decisões.

⁴ KUME, H. *Métodos estatísticos para melhoria da qualidade*: São Paulo. AOTS, Ed. Gente. 1993

⁵ IMAI, M., *Kaizen: The Key to Japanese Competitive Success*. Random House, New York. 1986.

⁶ CAMPOS, Vicente F. *TQC: Controle da qualidade total (no estilo japonês)*. 5. ed. Rio de Janeiro: QFCO, 1992.

Paz (2011) enfatiza que a maior dificuldade na implantação de um modelo de Gestão da Melhoria Contínua está na cultura dos empresários, que na grande maioria das vezes são resistentes à modelos de gestão que focam a efetiva participação dos colaboradores.

Observa-se que a implantação da melhoria contínua está ligada ao processo de mudança organizacional. Este processo não deve começar com discursos exortativos, muitas vezes apelando para a mão-de-obra. “Atitudes valem mais que palavras” é uma frase já muito difundida no mundo. Também não se deve iniciar um processo de mudança organizacional pelo estabelecimento de metas numéricas de melhoria. Não é fácil saber exatamente – e em termos numéricos – quanto e em que prazo é possível melhorar. Além disso, práticas como essa geralmente fazem com que os colaboradores só estejam interessados nos valores numéricos. O aumento de incentivos monetários também não é indicado como uma prática para se iniciar um processo de mudança organizacional. Ao fazer isso, a empresa estará formando “mercenários” que sempre irão se importar com a frase “o que eu vou ganhar se eu fizer isto?” (MOURA, 2011).

2.2 Desenvolvimento de Software

Um processo de desenvolvimento de software é um conjunto de atividades com o objetivo de analisar e desenvolver um produto de software (SOMMERVILLE, 2003).

É interessante observar, sob a ótica da Engenharia de Produção, a diferença entre os termos “produzir” e “desenvolver”. Conforme Silva *et al.* (2009) colocam, para produzir é necessária capacidade técnica, muitas vezes sobre algo que já se tem experiência e que pode ser realizado em segundos por processos automatizados. Por outro lado, desenvolver envolve aprendizado, análise, criatividade e conhecimento avançado.

Um processo de desenvolvimento de software geralmente é composto por várias atividades em fases distintas. Algumas etapas que podem ser citadas são: análise de requisitos, implementação, testes e manutenção.

Bueno (2008) afirma que a qualidade de software é um assunto muito discutido e pouco praticado, sendo que os problemas de qualidade se estendem desde o início dos tempos de desenvolvimento de software.

A qualidade de um software está diretamente ligada ao seu processo de desenvolvimento. De uma maneira ou de outra, a grande maioria das empresas diz possuir um processo bem definido de desenvolvimento. Chaves *et al.* (2010) ressaltam que, enquanto um processo pode ser imaturo e utilizar uma abordagem *ad-hoc* para organizar o trabalho, existem processos maduros que são caracterizados por uma metodologia padronizada e documentada, que já foi utilizada em projetos similares ou mesmo já é adotada em uma organização. Assim, é importante que o processo de desenvolvimento da empresa tenha um embasamento teórico em técnicas já testadas e comprovadas.

A norma ISO/IEC 12207 tem como objetivo ajudar os desenvolvedores de software - e não só os desenvolvedores, mas toda equipe envolvida - a definir seus papéis. Para isso, se utiliza de processos bem definidos, ajudando a organização a compreender as principais atividades e tarefas relacionadas ao desenvolvimento de software (NOGUEIRA, 2003).

A norma ISO/IEC 12207 propõe uma arquitetura de alto nível para o ciclo de vida de software por meio da definição de processos, atividades e tarefas que podem ser aplicadas na aquisição, no fornecimento, no desenvolvimento, na operação e na manutenção de software. Cada processo, por sua vez, é composto por atividades que, ainda, são compostas por tarefas. Os processos se classificam em: fundamentais, de apoio e organizacionais, como pode ser visto na Figura 1 (LEAL, 2010).



Figura 1 - Processos da Norma ISO/IEC 12207

Os processos fundamentais abordam o início do desenvolvimento, compreendendo, por exemplo, a contratação do fornecedor, execução do desenvolvimento, operação e manutenção de produtos de software. Nogueira (2003) destaca que muitas empresas – em comum acordo com os contratantes – consideram a documentação como perda de tempo e por isso, partem direto para o desenvolvimento (codificação), sendo levadas, posteriormente, a um processo de manutenção e correção intermináveis. Os processos fundamentais comandam os demais processos da norma, sendo em número de cinco (ARRUDA, 2006).

- **Aquisição:** seu início se dá com a identificação de uma necessidade do cliente, terminando com a aprovação do software pelo mesmo. Este processo possui quatro subprocessos: Preparação para Aquisição, Seleção de Fornecedor, Monitoração do Fornecedor e Aceitação pelo Cliente.
- **Fornecimento:** tem por objetivo formalizar uma proposta que atenda aos requisitos do cliente, constituindo-se a base para a execução dos processos de desenvolvimento do software. Possui cinco subprocessos: Proposta do Fornecedor, Acordo Contratual, Liberação do Produto e Suporte à Aceitação do Produto.
- **Desenvolvimento:** contém todas as atividades e tarefas para o desenvolvimento do software, desde a análise de requisitos, construção, testes, instalação e integração, sendo composto pelos seguintes subprocessos: Elicitação de Requisitos, Análise dos Requisitos do Sistema, Projeto da Arquitetura do Sistema, Análise dos Requisitos do Software, Projeto do Software, Construção do Software, Integração do Software, Teste do Software, Integração do Sistema, Teste de Sistema e Instalação do Software.
- **Operação:** contém as atividades e tarefas para a operação do software bem como suporte operacional aos usuários. Possui dois subprocessos: Uso Operacional e Suporte ao Cliente.
- **Manutenção:** é ativado quando é necessário realizar alguma mudança no software, seja uma correção de algum problema, atendimento a uma nova necessidade do cliente ou proposta de melhoria.

Os processos de apoio têm por objetivo garantir o sucesso e a qualidade do software. Eles serão usados oportunamente, em processos de verificação, controle de qualidade e gerência de configuração, por exemplo. Os oito processos de apoio são:

- **Documentação:** consiste no desenvolvimento de todos os documentos ligados ao projeto, desenvolvimento, produção, manutenção, edição e distribuição do software e

disponibilização destes documentos a todos os interessados, como gerentes, engenheiros e usuários do sistema.

- Gerência de Configuração: compreende o controle global de todos os itens que constituem o software, tais como modificações, consistência, correção, armazenamento, liberação, manipulação e distribuição dos itens.
- Garantia da Qualidade: consiste na definição de atividades que fornecerão a garantia adequada ao software, no que tange ao atendimento dos planos estabelecidos e atendimento às necessidades do cliente.
- Verificação: define as atividades que verificarão se um produto de software de uma atividade atende aos requisitos a ela imposta.
- Validação: tem como objetivo determinar se os produtos e o produto final atendem aos objetivos propostos.
- Revisão Conjunta: a revisão conjunta é realizada em níveis técnicos e de gerenciamento e tem por objetivo avaliar a situação de um produto de uma atividade do projeto.
- Auditoria: define as atividades para verificação da adequação aos requisitos, contratos e plano.
- Resolução de Problemas: tem por objetivo definir atividades que auxiliarão a solução de problemas de quaisquer naturezas que sejam encontrados em quaisquer fases do desenvolvimento.

Os processos organizacionais são empregados por uma organização para estabelecer e implementar uma estrutura constituída pelos processos de ciclo de vida e pelo pessoal envolvido no desenvolvimento do software (Leal, 2010) e muitas vezes não são empregados em um projeto específico, contribuindo diretamente para a melhoria da organização. Eles são em número de quatro:

- Gerência: compreende as atividades de gerenciamento como um todo (tarefas, atividades, projetos, etc.).
- Melhoria: medição, avaliação e controle de um processo de ciclo de vida de software.
- Infraestrutura: inclui hardware, software, ferramentas, técnicas e padrões necessários para manter qualquer processo.
- Treinamento: compreende as atividades necessárias para o treinamento de toda equipe envolvida, desde os desenvolvedores até os usuários.

O relacionamento dos processos estão baseados em dois conceitos principais (NOGUEIRA, 2003):

- Modularidade: todas as partes de um processo estão fortemente relacionadas, configurando processos com alta coesão e baixo acoplamento.
- Responsabilidade: cada processo pertence a uma “parte envolvida”. Estas “partes” podem ser de organizações diferentes.

2.3 Processo de Software

O processo de software é definido como a sequência de etapas executadas para realizar um determinado objetivo e que envolve métodos, ferramentas e pessoas e pode ser visto como um conjunto de atividades, métodos, práticas e transformações que as pessoas utilizam para desenvolver, manter e evoluir software e os artefatos associados (Humphrey⁷, 1989 *apud* Abreu, 2008); ou ainda, é a maneira pela qual o desenvolvimento e manutenção de software são organizados, gerenciados, medidos, apoiados e melhorados (Montangero⁸ *et al.*, 1999 *apud* Malheiros, 2010).

Miranda (2006) enfatiza que o processo é o elemento fundamental para se alcançar os objetivos do negócio da organização e por isso, ter um processo bem definido é fundamental para todas as empresas. Além disso, a definição do processo deverá permitir adaptações e possibilidade de expansão, principalmente por causa das inúmeras mudanças que ocorrem nas organizações e do avanço da tecnologia.

A área de melhoria de processos de software (MPS) tem sido investigada sistematicamente, dadas as evidências de que a qualidade do processo pode influenciar significativamente na qualidade do produto final (MALHEIROS, 2010).

É importante destacar que o sucesso de um projeto de software começa com a escolha de uma metodologia compatível com as características do mesmo e com um bom planejamento (VASCO *et al.*, 2008).

⁷ HUMPHREY, Watts S., *"Managing the Software Process"*, Addison-Wesley, 1989.

⁸ MONTAGERO, C.; DERNIAME, J.-C.; KABA, B.A.; WARBOYS, B. *The software process: Modelling and technology*. In: *Software Process: Principles, Methodology, Technology*, London, UK: Springer-Verlag, p. 1-14, 1999.

Leal (2010) apresenta uma análise comparativa entre alguns processos de desenvolvimento de software, destacando as principais características de cada um. Dentre esses, destacam-se:

- RUP: Possui 4 fases (início, elaboração, construção e transição) e é um processo iterativo, incremental e adaptável. A comunicação é feita através de artefatos e existe uma divisão bem definida de atividades;
- XP: sendo uma metodologia de desenvolvimento ágil, se baseia em 4 valores principais: comunicação, simplicidade, *feedback* e coragem. Ele possui 6 fases (exploração, planejamento, iterações para *release*, validação para produção, manutenção e morte) e tem por objetivo tornar o projeto mais flexível;
- AUP: é uma versão simplificada do RUP que utiliza técnicas e conceitos de métodos ágeis e assim, descreve o desenvolvimento do software de uma maneira simples e fácil;

É importante destacar que a escolha de um processo de desenvolvimento adequado a cada organização deve estar pautada em aspectos como o tamanho da organização, tamanho dos projetos, número de funcionários envolvidos, etc. (PARREIRAS e OLIVEIRA, 2004).

A seguir, cada um dos três processos de desenvolvimento de software serão descritos e analisados.

2.3.1 RUP – *Rational Unified Process*

De acordo com Delmas *et al.* (2007), o RUP é um processo proprietário de Engenharia de Software que fornece técnicas a serem seguidas pela equipe de desenvolvimento de software. É projetado e documentado através da UML e sua concepção é pautada na orientação a objetos.

Leal (2010) ainda ressalta que ele faz uma divisão bem definida de atividades e o divide em quatro fases, enquanto Delmas (2007) destaca a ênfase de cada uma delas:

- Início: determinação do escopo do desenvolvimento (ênfase no escopo do sistema);
- Elaboração: planejamento das atividades e dos recursos necessários (ênfase na arquitetura);

- Construção: implementação do código, propriamente dita (ênfase no desenvolvimento);
- Transição: disponibilização do produto (ênfase na implantação).

Na fase de Início há o contato com o cliente, onde são definidos os principais custos e prazos, são estabelecidas as prioridades e é feito o levantamento dos requisitos do sistema. De uma maneira genérica, é definido o escopo do projeto. A fase de Elaboração consiste numa análise mais detalhada do plano de projeto, com uma revisão dos prazos e riscos envolvidos, entendendo melhor o domínio do problema. Na fase de Construção são desenvolvidos os componentes de software, ou seja, ele é de fato construído. A fase de Transição consiste na entrega do produto ao cliente, bem como no treinamento dos usuários que utilizarão o software, garantindo que este seja completamente entendido por aqueles (MARTINEZ, 2011).

Cada fase possui *milestones* definidos, que têm por objetivo indicar o progresso do projeto e dar suporte às decisões de continuar, abortar ou alterar a direção do mesmo (VASCO *et al.*, 2008).

As melhores práticas de desenvolvimento de software moderno nos quais o RUP se baseia são (TAMAKI, 2008):

- Desenvolvimento de Software Iterativo: é um ciclo de vida formado por várias iterações e o software é desenvolvido incrementalmente ao longo das iterações;
- Gerenciamento de Requisitos: análise sistemática que identifica, organiza e documenta os requisitos do sistema;
- Arquitetura baseada em Componentes: reduzem a complexidade e o tamanho da solução, sendo mais robustas e flexíveis;
- Modelagem Visual: um modelo é uma visão simplificada do sistema, que mostra os elementos essenciais do mesmo de uma perspectiva específica. A utilização de modelos como a UML contribuem muito para a compreensão do sistema, permitindo a comparação de projetos arquiteturais, servindo de base para a implementação, ajudando na captura dos requisitos e na comunicação;
- Verificação Contínua da Qualidade: a qualidade é avaliada durante todo o projeto, não apenas no final do mesmo. Além disso, há planejamento de testes e revisão de artefatos;

- Gerenciamento de Mudanças: propicia controle das versões, dos *releases* e de uma maneira geral, um controle das mudanças nos requisitos, projeto, implementação e testes.

O RUP está dividido em nove disciplinas. Uma disciplina pode ser entendida como uma coleção de atividades relacionadas que compartilham um mesmo contexto de um projeto. Para Luiz (2009), enquanto a separação das atividades em disciplinas proporciona um melhor entendimento do projeto, isto acaba dificultando o planejamento das atividades. No entanto, Tamaki (2008) defende que o agrupamento das atividades em disciplinas facilita a compreensão do processo.

As disciplinas e suas principais definições são (LEAL, 2010):

- Gerência de Projetos: gerencia as metas levando em conta os custos e os prazos ao mesmo tempo em que gerencia os riscos no que tange à entrega de um produto que satisfaça as necessidades do cliente;
- Modelagem de Negócio: tem por objetivo entender a estrutura e a dinâmica da organização onde o software será inserido, bem como todas as regras de negócio relacionadas ao mesmo;
- Requisitos: entendimento das necessidades que o software deve atender, geralmente na forma de casos de uso;
- Análise e Projeto: consiste na especificação de como serão implementadas as soluções para os requisitos do cliente;
- Implementação: como o próprio nome sugere, nessa disciplina ocorre a implementação das classes e objetos, sendo responsável pela geração do código fonte do software;
- Teste: analisa se o produto atende aos requisitos do sistema, identificando erros, falhas e documentando-os;
- Configuração e Controle de Mudanças: tem por objetivo dar suporte às questões de controle de versão do projeto;
- Ambiente: organiza o projeto em termos de ferramentas, métodos e processos;
- Distribuição: se envolve com a distribuição, instalação, testes e treinamento dos usuários do sistema.

Cada disciplina possui seus papéis específicos. Basicamente, um papel define um conjunto de responsabilidades de um indivíduo ou equipe ao longo do processo de desenvolvimento. Cada papel, por sua vez, possui um conjunto de atividades, que devem ser realizadas por ele. As atividades possuem finalidade clara e produzem resultados significativos para o contexto do projeto. Por meio das atividades são gerados artefatos, definidos basicamente como um produto de trabalho do processo. Os papéis também podem fazer uso de artefatos para realizar as atividades (TAMAKI, 2008).

Broering (2011) aborda duas maneiras de adotar o RUP como metodologia de desenvolvimento. A primeira consiste na adoção dele à risca, ou seja, aplicando todos os métodos e processos exatamente como são propostos, o que pode se tornar inviável, visto que o RUP é bem complexo em sua íntegra, o que implicaria em treinamentos, projetos pilotos e um longo período de implantação. A segunda seria utilizar o RUP como referência para assuntos não abordados em outros modelos, sendo adequada, por exemplo, para empresas que já possuem algum tipo de metodologia que é inadequada em alguma área, como a documentação.

Enquanto Delmas *et al.* (2007) defendem que o RUP é preferencialmente aplicável a grandes equipes de desenvolvimento e a grandes projetos, Leal (2010) defende que por ser iterativo, incremental e adaptável, ele pode ser customizado para produtos e projetos de software dos mais variados tamanhos e tipos.

2.3.2 XP – *Extreme Programming*

O XP se difundiu e se tornou internacionalmente conhecido com a publicação do livro *Extreme Programming Explained: Embrace Change*, em 1999, por Kent Beck. Com a primeira versão voltada exclusivamente para pequenas equipes de desenvolvimento, na segunda versão houve uma reformulação completa do mesmo, onde ele passou a ser voltado para equipes de qualquer número de pessoas⁹. O uso da palavra *extreme*, de acordo com Silva (2007), tem a ver com o entendimento de que a técnica leva as melhores técnicas de programação ao extremo.

⁹ Além disso, também pode ser aplicado em equipes distribuídas.

O objetivo principal do XP é a excelência no processo de desenvolvimento de software, através de baixos custos, alta produtividade, poucos defeitos e alto retorno de investimento (SATO, 2007).

O XP está fundamentado em Valores, Princípios e Práticas. Os Valores são critérios universais e podem ser utilizados no julgamento de determinadas situações. Os Princípios têm por objetivo traduzir os Valores em Práticas. As Práticas, por sua vez, compreendem técnicas objetivas, diretas e específicas utilizadas no dia-a-dia de um processo de desenvolvimento.

2.3.2.1 Valores

O XP está baseado em cinco valores principais, sendo eles (BASSI, 2008):

- **Comunicação:** a comunicação é um fator muito importante quando se deseja construir um produto com qualidade e que atenda às necessidades do cliente. O XP favorece a comunicação da equipe de desenvolvimento através de atividades colaborativas. Quanto mais eficiente for a comunicação, mais facilmente poderão ser aproveitadas as contribuições de cada membro da equipe, bem como as críticas, criando melhoras nas soluções antes mesmo destas serem implementadas.
- **Simplicidade:** um sistema possui recursos essenciais e recursos “extras”. É preferível inicialmente implementações simples e das necessidades essenciais, que simplificam o código e demandam um prazo menor. É mais fácil criar algo simples e sofisticá-lo posteriormente do que já começar criando algo complexo, de difícil manutenção e mais propenso a erros. É por isso que os membros da equipe sempre estão à procura de soluções simples para todos os problemas.
- **Coragem:** no início de um processo de desenvolvimento há um planejamento das atividades e de tudo que será implementado. Com o passar do tempo e a evolução do projeto, o planejamento tem de ser revisto e decisões importantes como abrir mão de planos e trabalhos feitos têm de ser tomadas. Por isso é importante ter coragem, para mudanças e inovações, principalmente. Além disso, coragem está diretamente ligada às consequências que os atos trarão no futuro.
- **Feedback:** o *feedback* deve estar presente constantemente entre toda equipe de desenvolvimento. Ele permitirá que problemas e impedimentos sejam identificados mais cedo, fazendo com que se perca menos tempo com eles. Por isso é importante

que as avaliações do produto e do processo sejam frequentes. Os ciclos de feedback devem ser curtos, fazendo com que ele seja fornecido em minutos e até em segundos, dependendo do caso.

- Respeito: o respeito de todos para com todos é muito importante num processo de desenvolvimento. Sem ele, os demais valores não serão eficazes. Se não houver respeito, a sinergia da equipe será minada e a qualidade do produto será ruim. Respeito pode ser considerado, por exemplo, ao se assumir responsabilidades que poderão ser cumpridas e no comprometimento de que todos farão o melhor trabalho possível. O sucesso no desenvolvimento de software depende primeiramente das pessoas, não importando qual método ou técnica será utilizada. Por isso o respeito é tão importante.

2.3.2.2 Princípios

Para transformar os valores em práticas, existem alguns princípios fundamentais nos quais o XP está fundamentado, sendo eles: Humanidade, Economia, Benefício Mútuo, Auto semelhança, Melhoria, Diversidade, Reflexão, Fluxo, Oportunidade, Redundância, Falha, Qualidade, Passos pequenos e Aceitação da responsabilidade.

- Humanidade: diz respeito às pessoas que estão desenvolvendo o software. Pessoas têm necessidades e a principal dificuldade é aliar as necessidades delas com as necessidades de toda equipe.
- Economia: diz respeito à agregação de valor ao produto, priorizando os recursos mais importantes para o cliente, em primeiro lugar. Também diz respeito à flexibilidade do projeto.
- Benefício Mútuo: diz respeito aos benefícios que as atividades trazem aos membros da equipe, o que deve acontecer em todas as atividades. Um exemplo contrário a este princípio, é a escrita de documentos longos que não têm valor algum no momento, bem como planejamentos de longo prazo, que com certeza sofrerão muitas alterações.
- Auto semelhança: diz respeito à prática de se utilizar uma boa solução em outros projetos, podendo ser de diferentes conceitos e escalas.
- Melhoria: diz respeito à produção de algo “bom” hoje que poderá ser melhorado amanhã, se tornando algo “ótimo”.

- **Diversidade:** diz respeito às várias habilidades presentes na equipe, que propiciam diferentes perspectivas e opiniões, gerando um universo de várias ideias e soluções.
- **Reflexão:** diz respeito à análise de si mesmo por cada membro da equipe, aprendendo com as próprias falhas e identificando pontos que podem ser melhorados. A Melhoria Contínua deve sempre ser uma busca constante.
- **Fluxo:** diz respeito à criação de um fluxo de desenvolvimento de software, no qual o software é incrementado constantemente, com novas versões sendo implantadas em curtos períodos de tempo, fazendo com que a possibilidade de erros diminua, bem como o tamanho deles, sendo mais fácil corrigi-los.
- **Oportunidade:** diz respeito à equipe estar sempre atenta às possibilidades que surgem ao longo do processo de desenvolvimento. Problemas e mudanças bruscas também devem ser vistos como oportunidades de melhoria.
- **Redundância:** diz respeito à diminuição na probabilidade de defeitos. Para isso, os problemas mais difíceis e críticos devem ser resolvidos de várias maneiras diferentes. Algumas práticas de XP, como Programação Pareada (dois desenvolvedores trabalhando lado a lado em um computador) e Integração Contínua, são redundantes na tentativa de reduzir a quantidade de defeitos e aumentar a qualidade do software produzido (SATO, 2007).
- **Falha:** diz respeito à implementação paralela de várias soluções possíveis, mesmo sabendo que algumas delas irão falhar. Nesse caso, o importante é aprender com as falhas e reduzir tempo com discussões prolongadas sobre qual solução é a melhor.
- **Qualidade:** diz respeito ao projeto tratar a qualidade como uma atividade chave, sendo uma meta constante para a equipe. Ao contrário do que muitos possam pensar, comprometer a qualidade do projeto não contribui para um desenvolvimento mais rápido do mesmo.
- **Passos pequenos:** diz respeito à implementação em etapas pequenas, contribuindo (mais uma vez) para a diminuição dos possíveis erros e do tamanho destes.
- **Aceitação da responsabilidade:** diz respeito à aceitação das responsabilidades por cada membro da equipe.

2.3.2.3 Práticas

Inicialmente o XP foi criado seguindo uma abordagem de 12 práticas. Na sua segunda “versão”, as 12 práticas foram reformuladas e divididas em 13 práticas primárias e 11 práticas corolárias. As práticas Primárias são úteis e podem ser aplicadas de forma independente, não importando qual o contexto do projeto. As práticas corolárias, por sua vez, devem ser implementadas apenas quando houver muito domínio e experiência com as práticas primárias (Sato, 2007). A seguir serão apresentadas as 13 práticas primárias.

- **Sentar Junto:** as equipes devem dispor de um ambiente de trabalho no qual seja possível trabalharem juntos. Isso facilitará e aumentará a comunicação entre os membros da equipe, bem como a identificação de habilidades inerentes a cada membro.
- **Time Completo:** uma equipe de desenvolvimento deve possuir pessoas com diferentes habilidades, como testadores, analistas de projetos, analistas de negócios, clientes, etc. É importante que a equipe seja formada por pessoas qualificadas em todas as áreas necessárias.
- **Área de Trabalho Informativa:** o ambiente de trabalho deve refletir o projeto atual. É importante que as métricas estejam visíveis por todos, bem como gráficos mostrando o andamento do projeto, os cartões de história, etc. Para que isso seja possível, é necessário que um membro da equipe atualize esses dados constantemente.
- **Trabalho Energizado:** a equipe de desenvolvimento não deve trabalhar de forma que comprometa sua saúde ou sua vida pessoal. Elementos como hora extra devem ser raros e pouco utilizados. O cansaço e a indisposição diminuem a criatividade e a qualidade do código produzido. Para isso, o cronograma do projeto deve ser feito levando-se em conta cargas horárias realísticas com o ambiente de trabalho.
- **Programação Pareada:** os desenvolvedores devem trabalhar em pares. Enquanto um dos programadores digita, o outro pode revisar o código e sugerir melhorias (Bassi, 2008). Isto promove o trabalho colaborativo, unindo a equipe, melhorando a comunicação e disseminando o conhecimento, as habilidades e técnicas por toda a equipe. É importante que os pares sejam trocados frequentemente. Geralmente, eles são escolhidos de acordo com a tarefa que será realizada e a disponibilidade dos membros da equipe.

- Histórias: todo o planejamento do XP é feito com base em histórias escritas pelos clientes em cartões. Estes cartões possuem alguma funcionalidade requerida do software, geralmente requisitos funcionais. Os clientes também são responsáveis por dizer qual a prioridade de cada cartão de história. Estes também podem ter informações adicionais, como o nome do responsável pela implementação daquela atividade, o prazo estimado, etc. Porém, é importante que apenas algumas informações estejam no cartão, pois assim, a comunicação continuará sendo um valor chave ao longo de todo processo de desenvolvimento. Um exemplo de cartão de história pode ser visualizado no Quadro 1.

Quadro 1 - Exemplo de cartão de história

<p>Como um <usuário/papel> Eu gostaria de <funcionalidade> Para que <valor de negócio></p>
--

Fonte: (SATO, 2007)

- Ciclo Semanal: a equipe deve se reunir semanalmente, a fim de avaliar o progresso das atividades e do projeto como um todo, bem como fazer novos planejamentos de curto prazo. Nas reuniões, as histórias dos cartões são “quebradas” em atividades e elencadas a pares para serem implementadas.
- Ciclo Trimestral: tem como objetivo o planejamento das *releases*. As releases são planejadas de acordo com um tema específico. A equipe, então, deve escolher as histórias que mais se alinham ao tema, bem como identificar problemas e avaliar o ciclo anterior, sempre em busca do aprendizado e da melhoria.
- Folga: o planejamento de uma tarefa se baseia em estimativas de tempo que, por sua vez, se baseiam nas experiências pessoais dos membros da equipe. Por causa disto, as estimativas estão sujeitas a erros. É importante que os prazos fornecidos aos clientes sejam obedecidos. Por isso, é interessante a introdução de folgas no processo de desenvolvimento. Estas folgas podem ser colocadas como tarefas que, se necessário, podem ser descartadas sem comprometimento do projeto como um todo. Dessa maneira, garante-se que todas as tarefas principais serão realizadas, mesmo que haja um atraso nas estimativas.

- *Build* em 10 Minutos: o *build* consiste em todas as atividades necessárias para verificar a consistência do código, colocar o software em funcionamento e realizar todos os testes. É importante que o build não demore mais do que 10 minutos. Além disso, quanto mais rápido ele for, mais frequente será o *feedback* para a equipe.
- Integração Contínua: o código fonte do software deve ser mantido em um repositório e estar disponível a todos os membros da equipe. Após um novo código ser implementado e testado, ele deve ser integrado no repositório. A equipe diariamente irá recuperar e adicionar códigos neste repositório, o que garante que todos sempre estejam trabalhando com o código atualizado.
- Desenvolvimento Dirigido por Testes: esta prática constitui-se uma das mais importantes no XP. Os testes devem ser planejados antes do início da implementação do código fonte¹⁰. Além disso, a cada nova funcionalidade adicionada, os testes devem ser realizados novamente a fim de ter certeza que nada foi comprometido. Os testes devem ser escritos pelos programadores, mas devem considerar os conhecimentos dos clientes (as regras de negócio), comprovando que o software realiza o que realmente é preciso.
- Design Incremental: a equipe deve implementar o design mais simples necessário para suportar todas as necessidades inerentes ao software. Conforme o desenvolvimento evolui e as funcionalidades são incrementadas, o design deve ser melhorado também.

2.3.2.4 Documentação

A documentação no XP é minimalista e constitui-se de cartões de histórias e do próprio código fonte e radiadores de informação.

Os cartões de história estão explicados e definidos no item **2.3.2.3 Práticas**.

O código fonte não necessita de esforço de manutenção, está sempre atualizado e com alguns comentários constitui-se na melhor documentação que um software pode ter.

Os radiadores de informações têm como objetivo prover um ambiente auto informativo. Eles são cartazes espalhados por todo ambiente de desenvolvimento, mostrando informações

¹⁰ Na verdade isto não é uma regra, mas é preferível que todos os testes sejam planejados o quanto antes.

relevantes sobre o andamento do projeto. Conforme Bassi (2008), algumas informações que os radiadores de informações podem transmitir são: evolução dos testes, quantidade de cartões concluídos, etc.

2.3.2.5 Papéis

O XP possui vários papéis, porém nem todos existem nas equipes de desenvolvimento. Muitas vezes o papel de alguém é realizado por outro papel. Além disso, espera-se que não haja uma divisão muito clara entre cada pessoa, afim de que o conhecimento de todas as áreas seja disseminado e todos possam ter a mesma possibilidade de crescimento. Os principais papéis presentes no XP são (BASSI, 2008):

- Programadores, responsáveis principalmente pela implementação do software;
- Analistas de Negócios, que têm como função ajudar os clientes a definir e escrever os cartões de histórias;
- Analistas de Testes, responsáveis por criar os vários cenários de teste;
- Projetistas de Interação, responsáveis por avaliar a utilização do sistema pelos usuários;
- Arquitetos, responsáveis por analisar a estrutura do sistema e propor melhorias;
- Gerentes de Projeto, responsáveis por facilitar o trabalho dos programadores;
- Gerentes de Produto, responsáveis por definir os objetos do ciclo do projeto e priorizar histórias;
- Redatores Técnicos, responsáveis pela documentação;
- Executivos, responsáveis pelos objetivos de alto nível;
- Recursos Humanos, responsáveis pela solução de problemas burocráticos;
- Usuários, responsáveis por contribuições (opiniões, sugestões, críticas, etc.).
- *Coach*, responsável por verificar se os programadores estão seguindo padrões e metodologias propostas. O papel do *coach* geralmente é assumido por um programador com muita experiência.

Muitas vezes observa-se que os papéis de analista de testes e programadores são realizados pela mesma pessoa.

O cliente, embora não pertença ao processo de desenvolvimento em si, é um papel fundamental para o projeto, uma vez que ele é o conhecedor das principais regras de negócio (na maioria das vezes), bem como dos ambientes de testes e deve ser capaz de tirar eventuais dúvidas que a equipe possa ter.

2.3.2.6 Adaptações e Adoção do XP

Adotar um método ágil como o XP é uma tarefa difícil, trabalhosa e que demanda muito tempo. Conforme Silva (2007), XP é uma metodologia que requer muita disciplina e também mudança social. Por este motivo, adaptações são permitidas, a fim de que a equipe se familiarize com o método aos poucos e melhore cada vez mais. O importante é focar na busca pela melhoria contínua das práticas. Por isso, é possível que a equipe altere algumas práticas, crie novas ou mesmo utilize práticas de outros métodos de desenvolvimento ágil.

Para a adoção do XP, a filosofia da empresa tem que estar alinhada com a filosofia do método. Assim, em empresas que dão valor aos segredos, à timidez, à complexidade e ao isolamento, o XP provavelmente não irá funcionar. Por isso é importante que haja sinergia entre os valores do XP e os valores da equipe. O importante hoje não é “como” o XP funciona, mas sim “por quê” ele funciona. E a resposta a esta pergunta está pautada nos seus valores e princípios por trás das práticas (SATO, 2007).

2.3.3 AUP – Agile Unified Process

O AUP está baseado no RUP (2.3.1 RUP – *Rational Unified Process*) e incorpora processos ágeis, sendo composto por sete disciplinas e quatro fases, sendo estas (Santos *et al*, 2007).

- **Concepção:** esta fase compreende o entendimento dos objetivos do *stakeholder* (análise de requisitos), a análise da viabilidade do projeto (em termos de custos, negócios, etc.), a análise dos riscos envolvidos e o que será necessário para o ambiente de desenvolvimento (em termos de hardware, software, ferramentas, etc.). Uma divisão clara da fase de concepção evidencia quatro atividades-chave: Definir o Escopo do Projeto, Definir os Riscos, Definir viabilidade do Projeto e Preparar o Ambiente de Desenvolvimento do Projeto.

- **Elaboração:** os principais objetivos desta fase consistem na identificação, validação e início da construção da arquitetura do sistema que será desenvolvido. Nesta fase geralmente é construído um protótipo, chamado de “protótipo arquitetural”.
- **Construção:** nesta fase, o software será de fato desenvolvido. Ela compreende atividades de modelagem, construção e testes. Além disso, nesta fase também é feita a documentação de suporte do software, constituindo um artefato muito importante que deve ser mantido.
- **Transição:** esta fase consiste na entrega do software para o cliente (ainda que seja uma versão Beta) e inclui atividades de testes do software, testes de usuário, adaptações no software e aceitação pelo usuário. A documentação deve ser finalizada e possíveis bugs devem ser encontrados e corrigidos.

As quatro fases do AUP, bem como a ênfase dada em cada disciplina das mesmas, podem ser vistas na Figura 2 (GONÇALES *et al.*, 2010).

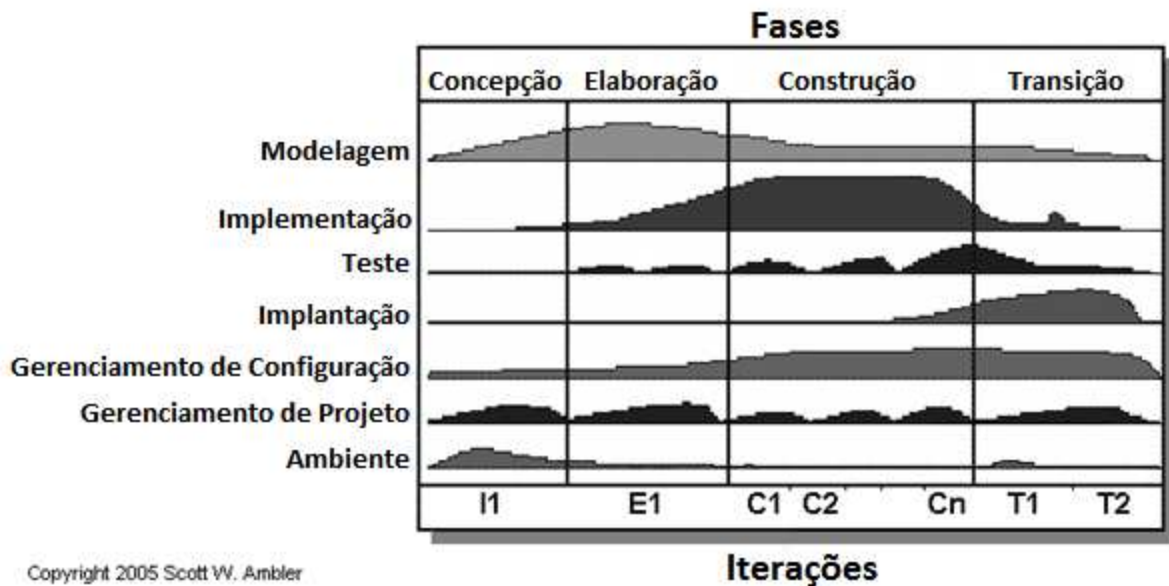


Figura 2 - Fases e Disciplinas do AUP

De acordo com Viggiano (2008), a filosofia do AUP está apoiada nos seguintes princípios: A equipe sabe o que está fazendo; Simplicidade; Agilidade; Foco em atividades de alto valor; O processo será personalizável, de acordo com as necessidades de quem o usa.

Pode-se dizer que o AUP é uma forma mais leve e objetiva de tratar os processos dos projetos de desenvolvimento de software. Considerando que o AUP é uma forma simplificada do RUP, ele adota um conceito de ciclo de vida de desenvolvimento orientado à modelagem ágil (AMDD – *Ágil Model Driven Development*), o que se traduz em um desenvolvimento minimalista. Isso significa que apenas o que for realmente essencial deve ser modelado e de uma forma bem objetiva (Gonçales *et al.*, 2010). É importante observar que isto não significa deixar de lado a documentação, mas sim focar naquilo que tem mais relevância no presente e também terá no futuro. Viggiano (2008) ainda destaca que isto não significa que os modelos devam ser ruins nem que o trabalho deva ser negligenciado, mas sim que as atividades devem ser executadas até o ponto em que otimizem o esforço.

Gonçales *et al.* (2010) apontam que a modelagem inicial no AUP muitas vezes é feita através de rascunhos em quadros brancos ou folhas de papel. Esta prática é justificada principalmente pelo nível de complexidade, que geralmente não é alto (no início). Os rascunhos conseguem explicitar muito bem o que se quer modelar e, posteriormente, podem ser incorporados na documentação final do sistema. É importante salientar que esta prática não substitui a utilização de ferramentas de modelagem mais sofisticadas.

2.4 Melhoria do Processo de Software

Conforme Malheiros (2010), a Melhoria de Processo de Software se trata de uma disciplina que gere, melhora e apoia o uso dos processos de desenvolvimento de software em uma organização e, através da disciplina, é possível entender o processo atual, propor mudanças e melhorar a qualidade do produto de software, reduzindo custos e tempo de desenvolvimento.

Para Hilgert (2008), a implantação de um Programa de Melhoria de Processo de Software demanda planejamento, dedicação, orientação de profissionais capacitados e, antes de tudo, trabalho em equipe. Além disso, programas desta natureza envolvem modificações na organização como um todo que, se não previstas e gerenciadas adequadamente, podem resultar no fracasso do programa.

Um Programa de Melhoria de Processo de Software geralmente tem como meta a implantação de um modelo de qualidade de software. Estes modelos de qualidade, por sua vez, têm como

objetivo analisar o processo de desenvolvimento atualmente utilizado e melhorá-lo, através de padrões e boas práticas de desenvolvimento de software já conhecidas e difundidas.

Como principais problemas na implantação de um Programa de Melhoria de Processo de Software, têm-se os altos investimentos que darão retorno apenas em longo prazo, bem como a necessidade de mudanças radicais na cultura da organização.

A seguir serão mostrados dois modelos de qualidade de software, também chamados de modelos de maturidade, sendo: o MPS-BR e o CMMI.

2.4.1 MPS-BR

Criado no final de 2003 através da participação de diversas universidades, instituições, grupos de pesquisa, empresas de software, etc. (Weber *et al*, 2006), e sob a coordenação da SOFTEX - Sociedade para Promoção da Excelência do Software Brasileiro, o Programa de Melhoria de Processo do Software Brasileiro - MPS.BR tem por objetivo a melhoria de processos de software em empresas brasileiras, a um custo acessível, voltado principalmente para as micro, pequenas e médias empresas.

Programas de melhoria como o CMMI são caros e, na maioria das vezes, inacessíveis às micro, pequenas e médias empresas que correspondem à maior parcela de empresas que desenvolvem software no país.

O MPS-BR é composto por 3 componentes básicos (LIEBMAN, 2006):

- Modelo de Referência para Melhoria de Processo de Software: contém os requisitos necessários que as empresas precisam ter para estarem em conformidade com o modelo, as definições dos níveis de maturidade, da capacidade de processos e dos processos em si.
- Método de Avaliação para Melhoria de Processo de Software: contém os requisitos para os avaliadores, os requisitos para averiguação da conformidade ao modelo e o processo de avaliação.
- Modelo de Negócio para Melhoria de Processo de Software: contém a descrição das regras para a implementação do modelo por empresas de consultoria, de software e de avaliação.

O MPS-BR é composto por sete níveis de maturidade (começando no G e terminando no A), através dos quais fica fácil acompanhar a evolução do processo e visualizar os resultados em um curto espaço de tempo. Os sete níveis são:

- A – Em Otimização
- B – Gerenciado Quantitativamente
- C – Definido
- D – Largamente Definido
- E – Parcialmente Definido
- F – Gerenciado
- G – Parcialmente Gerenciado

Para mudar de nível de maturidade a organização deve atender a todos os requisitos básicos do nível em que se encontra e dos anteriores.

2.4.2 CMMI

O CMMI - *Capability Maturity Model Integration* constitui-se de um conjunto de práticas para gerenciamento e melhoria da qualidade a serem aplicadas num processo de desenvolvimento de software. Foi criado em 1987 por Watts Humphrey, tendo sua primeira versão oficialmente publicada no ano de 2000 (SCHNEIDER, 2009).

O CMMI é dividido em cinco níveis de maturidade, sendo eles:

- Nível 1 - Estruturado: processo imprevisível e pobremente controlado (CITS, 2011), capaz tão somente de orientar as macro tarefas no nível operacional (CTGI, 2004).
- Nível 2 - Gerenciado: processo caracterizado para projetos, sendo frequentemente reativo (CITS, 2011). A maioria das empresas brasileiras está buscando sua certificação neste nível (CTGI, 2004).
- Nível 3 - Definido: processo caracterizado para a organização, sendo frequentemente proativo (CITS, 2011). Além disso, gerencia os aspectos organizacionais, técnicos e de integração de equipes e fornecedores (CTGI, 2004).
- Nível 4 - Gestão Quantitativa: neste processo o foco está na análise dos indicadores do processo (CITS, 2011). Ele consegue obter previsibilidade, através das comparações (CTGI, 2004).

- Nível 5 - Otimização: neste nível, o foco está na Melhoria Contínua do processo (CITS, 2011).

Schneider (2009) destaca, também, as características que uma organização madura em seus processos e projetos deve possuir: (i) processos definidos, documentados e utilizados; (ii) planos de gestão são desenvolvidos, monitorados e comunicados; (iii) papéis e responsabilidades claros; (iv) produto e processo medidos; (v) Qualidade, custo e prazo previsíveis; (vi) a tecnologia, além de ser utilizada com eficácia, é previamente planejada; (vii) Melhoria Contínua é um estilo de vida.

O CMMI ajuda na integração das funções organizacionais que tradicionalmente estão separadas, bem como no estabelecimento de um processo de melhoria nos objetivos e prioridades, no fornecimento de um guia para processos de qualidade e no fornecimento de um ponto de referência para instruir processos atuais (BERNARDO, 2010).

Souza (2011) destaca algumas áreas de processo presentes no CMMI que podem ajudar a equipe de desenvolvimento. A Solução Técnica ajuda na escolha do melhor método, ferramentas, arquiteturas e componentes a utilizar. A Integração de Produto é uma área que possui atividades de integração de produtos, geração de versão e liberação para ambiente de produção, por exemplo. A Validação oferece vários elementos que ajudam na criação de um ambiente para validação, teste e comunicação dos resultados.

Não se pode esquecer que o CMMI não é uma metodologia de desenvolvimento, – como o XP, o RUP e o AUP, por exemplo - o que significa que não existe um passo-a-passo de como implementá-lo. Ele contém guias e recomendações que ajudam a determinar o que um processo deve ter, bem como analisar a qualidade do mesmo (SOUZA, 2011).

3. PROPOSTA DO PROCESSO

Este capítulo descreve a formalização do Processo de Desenvolvimento de Software na empresa M.K.I. Informática LTDA ME, com base no exposto ao longo deste trabalho.

3.1 Caracterização da Empresa

A iSUPER (nome fantasia da M.K.I. Informática LTDA ME) nasceu da junção dos provedores de internet USOnline e Netmarr, bem como a Usina do Software, com o objetivo de atender às demandas crescentes de serviços de internet. Pioneira em serviços de Radiocomunicação, soluções VoIP e acesso à Internet, a iSUPER foi o primeiro provedor a iniciar suas operações de tecnologia à cabo e à rádio em Marialva e região, contribuindo para o desenvolvimento tecnológico da cidade bem como de toda região, fazendo a verdadeira Inclusão Digital. Esta constitui sua missão.

A iSUPER sempre optou por investimento de primeira linha na montagem de sua estrutura técnico/operacional para oferecer acesso de qualidade à rede internet, e através dos anos esta estrutura vem sendo ampliada e modernizada, objetivando atender seus clientes com precisão e respeito, o que sempre fez e continua fazendo parte de sua filosofia empresarial.

Operando com sistemas baseados em Linux, servidores de última geração e links de Internet exclusivos de fibras ópticas de alta capacidade redundantes, ela garante a qualidade do acesso, mantendo-a sempre um passo à frente, inclusive de empresas de grandes centros.

Além de banda larga de qualidade, um dos produtos da empresa é um sistema gestor para provedores de internet: o RouterBox. Ele é um sistema especializado e com diferenciais próprios, comercializado em todas as partes do país.

Atualmente a empresa conta com 17 colaboradores e dois sócios, sendo uma Prestadora de Serviços. O organograma de setores pode ser observado na Figura 3.

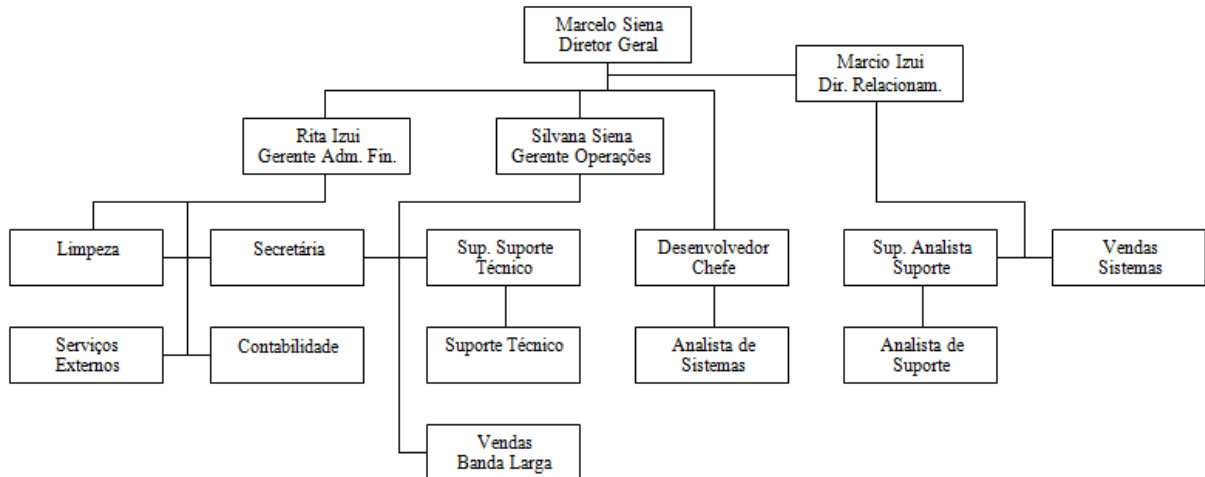


Figura 3 - Organograma da Empresa

3.2 Processo de Desenvolvimento

O Processo de Desenvolvimento da Empresa não está formalizado, mas existe. Por mais abstrato que seja, existem etapas e tarefas que são desenvolvidas seguindo uma sequência lógica.

A primeira tarefa consiste em identificar o tipo de desenvolvimento, que pode ser:

- Implementação de novos módulos;
- Manutenção em módulos existentes.

O processo de desenvolvimento possui tarefas similares de acordo com o tipo de implementação e tarefas específicas para cada tipo.

A Implementação de novos módulos (Figura 4) requer, inicialmente, um estudo das regras de negócio relacionadas aos novos módulos. Esta tarefa consiste na justificativa de implementação – que pode ser uma necessidade imposta por um cliente ou uma necessidade identificada pela Equipe de Desenvolvimento – bem como na análise de requisitos, identificando-se todos os elementos que serão necessários para a construção do módulo (ferramentas, bibliotecas, linguagens, manuais, etc.). Aqui também devem ser modeladas as tabelas necessárias do banco de dados – geralmente no Excel –, bem como um esboço da interface que o módulo deverá ter.

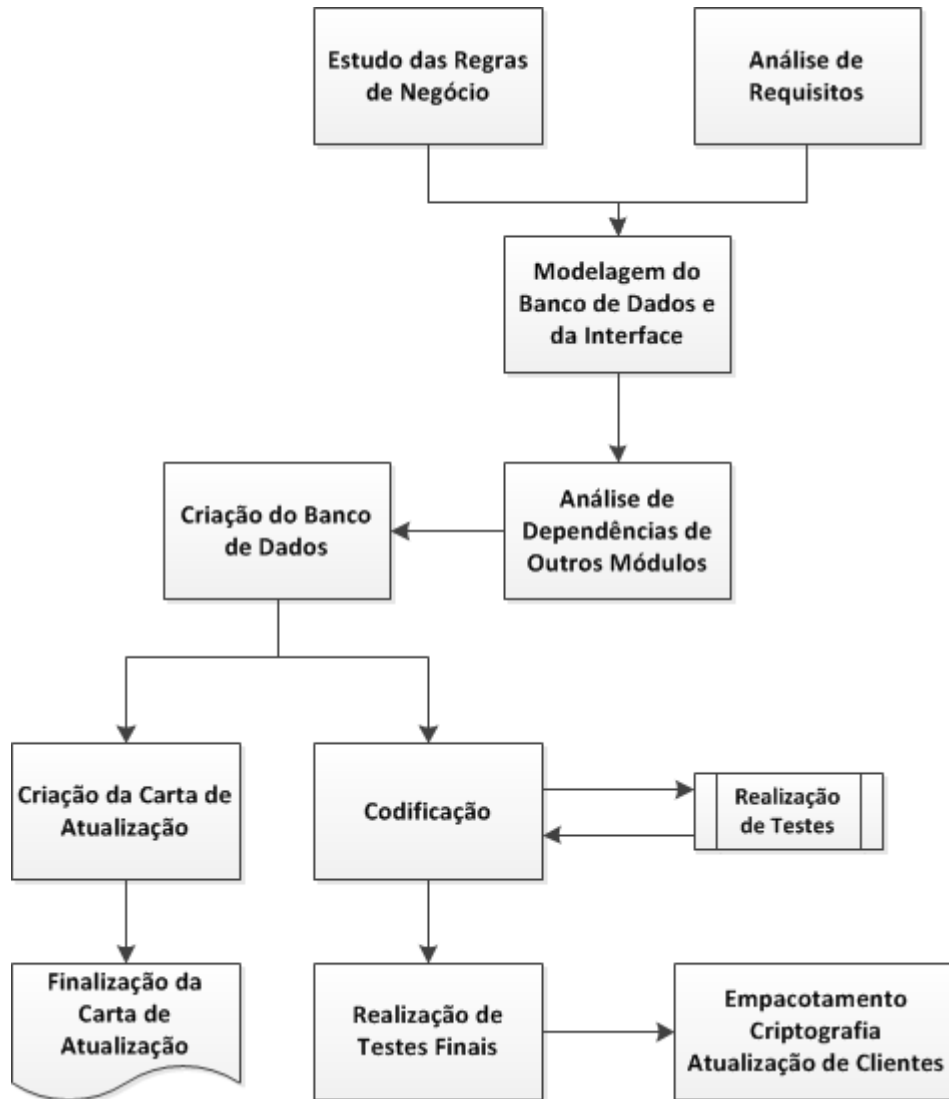


Figura 4 - Fluxograma da Implementação de Novos Módulos

A próxima tarefa consiste na integração dos novos módulos com os já existentes. Aqui são analisadas todas as alterações que deverão ser realizadas em módulos que já existem, em termos de layout gráfico, regras de negócio e banco de dados.

Com as duas tarefas anteriores concluídas, procede-se à criação das tabelas no banco de dados, de acordo com o modelo construído anteriormente. Após a construção das tabelas, dá-se início à codificação do módulo. Testes são realizados constantemente durante a implementação do código, pelo próprio desenvolvedor.

A Documentação atual que o sistema possui constitui-se de Cartas, criadas a cada nova atualização do mesmo pelo próprio desenvolvedor. A construção da Carta normalmente

ocorre paralelamente à implementação do código. Os rascunhos elaborados na primeira etapa também são todos arquivados.

Ao final da implementação dos códigos de todos os novos módulos, um teste geral e mais acurado é realizado, buscando-se identificar erros e inconsistências. Depois de tudo testado e a Carta finalizada, procede-se às tarefas específicas de atualização nos clientes – como empacotamento das aplicações, criptografia, etc. – que não fazem parte do escopo em questão e, por este motivo, não serão detalhadas.

A Manutenção em módulos existentes pode ocorrer pela adição de funcionalidades extras, pela descoberta de um bug que precisa ser corrigido ou por melhorias significativas. A primeira tarefa de uma manutenção consiste na análise de quais módulos possuem dependência com o módulo que será alterado. Em seguida, procede-se à implementação do código e as demais etapas são iguais às etapas de Implementação de novos módulos.

Geralmente as Manutenções são realizadas paralelamente às Implementações novas, estando estas como prioridade e aquelas em segundo plano.

3.3 Diagnóstico de Problemas

A falta de um processo formalizado acarreta muitos problemas e dificuldades em todas as fases do desenvolvimento, o que impacta sobre a produtividade.

O diagnóstico foi realizado por meio de observações *in loco*, entrevistas com a Equipe de Desenvolvimento e análise de documentos que a Empresa possui.

O primeiro problema que pode ser identificado são os erros que surgem quando um módulo que depende de outros é alterado. Este problema decorre da falta de documentação. Atualmente o sistema possui muitos módulos e muitas aplicações, grande parte interligados e que se comunicam constantemente (alto acoplamento). Durante a manutenção, os módulos dependentes estão apenas no conhecimento do desenvolvedor, pois não há nenhum documento relacionando todas as dependências que existem (por exemplo, diagrama de entidade-relacionamento). Assim, não é raro que aplicações necessárias sejam deixadas de lado durante a manutenção de determinado módulo. Conforme relato do principal

desenvolvedor da equipe, em determinada situação uma simples adição de um campo em determinada tabela do banco de dados foi responsável pela “parada” de dois módulos financeiros muito importantes. Este problema não teria ocorrido se houvesse uma documentação específica sobre aquele módulo.

O segundo problema que ocorre constantemente é o estouro nos prazos estabelecidos. O negócio da empresa não consiste em determinar um prazo fechado em que novas atualizações serão lançadas. Contudo, a empresa tem em sua política que o sistema não deve ficar mais de um mês sem atualização. Em contrapartida, geralmente o tempo para lançamento de uma atualização pode chegar a quatro vezes o estipulado. Isso gera muita insatisfação por parte dos clientes. O lançamento da versão 2.0.000 do sistema ocorreu no dia 26 de agosto de 2010. Um *release* foi lançado nos dias seguintes, para corrigir bugs inesperados que foram encontrados. Após este *release*, a próxima versão do sistema demorou em torno de 4 meses para ser lançada. A Tabela 2 apresenta as datas de lançamento das últimas versões do sistema.

Tabela 2 - Datas de Lançamento das Últimas Versões do Sistema

Versão	Lançamento
2.0.000	26/08/2010
2.0.001 (<i>release</i> de correções)	30/08/2010
2.0.002	29/12/2010
2.0.003	29/01/2011
2.0.004 (<i>release</i> de correções)	01/02/2011
2.0.005	24/02/2011

Outro problema que pode ser identificado é a falta de priorização entre as novas implementações e as manutenções constantes. Várias vezes a Equipe de Desenvolvimento para sua atividade para dar suporte e manutenção ao problema de determinado cliente, o que contribui ainda mais no problema anteriormente descrito, que são os estouros nos prazos.

A falta de documentação também é um problema grave. Os módulos do sistema possuem alto grau de acoplamento. No início do desenvolvimento perde-se muito tempo identificando todas as relações entre módulos. Além disso, problemas e dúvidas “comuns” – ou seja, que já foram identificados e resolvidos em outros momentos – requerem o mesmo esforço para encontrar uma solução. Estas duas situações seriam facilmente resolvidas se houvesse uma documentação mínima.

Pode-se apontar ainda a falta de um gerenciamento de testes adequado. Atualmente os poucos testes realizados não são documentados e muitas vezes cenários importantes deixam de ser testados, fazendo com que bugs não sejam identificados enquanto a nova versão ainda não foi lançada. A Figura 5 apresenta a quantidade de erros corrigidos nas últimas versões do sistema. Pode-se constatar que as atualizações lançadas com maior atraso – versões 2.0.002 e 2.0.006 respectivamente – são as que possuem maior quantidade de erros a serem corrigidos. A contagem foi realizada através das Cartas de Atualização, uma vez que são a única documentação disponível no momento.

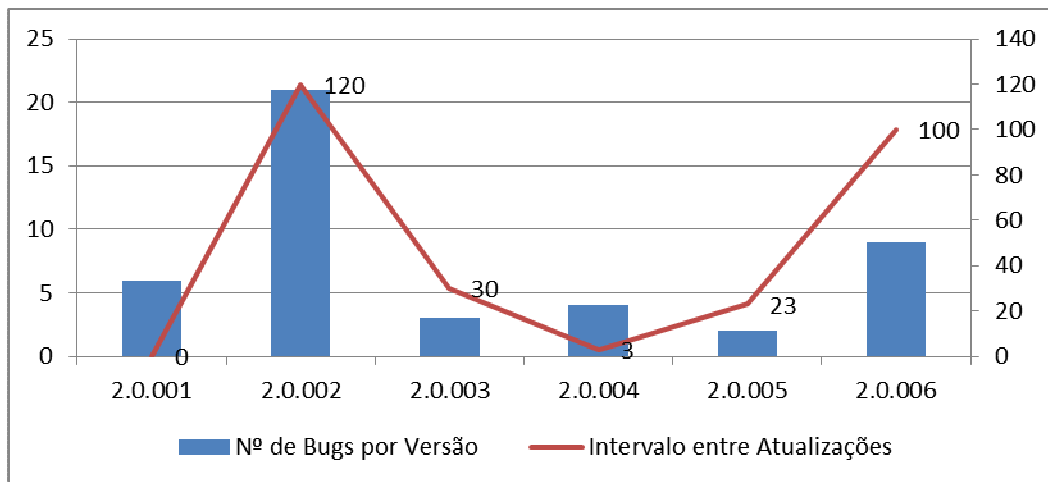


Figura 5 - Gráfico do Nº de Bugs por Versão

Por fim, a falta de um processo formalizado não possibilita um acompanhamento e avaliação da produtividade da Equipe de Desenvolvimento.

3.4 Estruturação do Processo de Desenvolvimento

O primeiro item observado na estruturação do Processo de Desenvolvimento foi a determinação em se deixar de lado toda e qualquer informalidade, de modo que a solução de problemas possa ser realizada de uma maneira sistemática. Por isso, o processo em si foi pautado no registro de todas as ações e resultados das mesmas, como forma de análise e avaliação do processo como um todo.

O início deu-se com o contato entre a Equipe de Desenvolvimento e os responsáveis pelo setor, uma vez que a gerência de nível superior tem que estar a par de tudo o que está acontecendo. O planejamento foi mostrado como um dos principais meios através do qual o setor de desenvolvimento conseguirá atingir seus objetivos de maneira eficiente, eficaz e com qualidade. Por este motivo, a própria gerência chegou à conclusão que tempo gasto com planejamento, é tempo economizado com manutenções futuras.

O âmbito do setor de desenvolvimento requer práticas consolidadas, aprovadas e testadas. Além disso, a gerência tem papel fundamental no que tange à validação e aceitação das práticas desenvolvidas, analisando e identificando pontos que precisam ser melhorados.

As várias atividades realizadas pela Equipe de Desenvolvimento envolvem análises, testes, implementações, implantações, distribuição, manutenções, etc. A estruturação do Processo de Desenvolvimento se pautará nos seguintes processos da norma ISO/IEC 12207:

- Desenvolvimento (Processos Fundamentais);
- Documentação e Verificação (Processos de Apoio);
- Melhoria Contínua (Processos Organizacionais).

A estruturação do Processo de Desenvolvimento será realizada tomando como base a metodologia RUP e agregando práticas da metodologia XP que podem contribuir positivamente para o resultado final. O processo existente hoje será trabalhado, dividido, especificado e se tornará o modelo padrão de desenvolvimento para a empresa.

É importante lembrar que o processo proposto é focado no projeto de novos *releases* para um sistema já existente. Assim, ele é voltado para a criação de novas aplicações, manutenções e melhorias no sistema.

O modelo proposto levou em consideração algumas práticas de desenvolvimento do RUP, a saber:

- O desenvolvimento do software será iterativo e incremental, através de um ciclo de vida pautado em disciplinas que delineiam as diretrizes do desenvolvimento.
- O gerenciamento de requisitos será realizado por um papel específico, e constituirá a primeira atividade no desenvolvimento de um novo *release*.
- A modelagem visual será utilizada na forma de modelos UML, principalmente diagramas de casos de uso, contribuindo na compreensão do sistema e na comunicação entre quem está passando os requisitos e quem está recebendo.
- A verificação contínua da qualidade se dará através de planejamento de testes, revisões e verificações durante todo o desenvolvimento e não apenas ao final deste.

Dois valores do XP em especial também foram levados em consideração: **comunicação** e **simplicidade**. O primeiro é elementar no desenvolvimento de qualquer atividade da qual participem várias pessoas. Em termos de desenvolvimento, a comunicação deve ser constante e perfeitamente compreensível. Especialmente a análise de requisitos demanda uma comunicação eficaz e objetiva. A simplicidade diz respeito a soluções simples para problemas complexos, uma vez que o aprimoramento de um determinado recurso geralmente é mais prático e vantajoso do que o desenvolvimento inicial de um recurso extremo.

Além dos valores, algumas disciplinas agregam práticas do XP, a fim de garantir que o modelo seja desenvolvido com práticas já comprovadas e testadas.

3.4.2 Papéis

Os papéis têm como objetivo definir responsabilidades e deveres e são dez, ao todo, detalhados a seguir.

- Analista de Requisitos: é o responsável pela definição dos recursos que o *release* deverá possuir e opcionalmente, da maneira com que a solução será implementada. O Analista de Requisitos deve possuir ótima capacidade de comunicação oral e relacionamento, porque muitas vezes será ele que estará em contato direto com o cliente.

- Gerente de Projetos: é o responsável pela delimitação dos prazos para conclusão do *release*. O Gerente de Projetos deve possuir ampla experiência, pois este papel é o principal responsável pelo sucesso ou fracasso na entrega do produto final dentro do prazo estabelecido com o cliente.
- Programador: constitui o elemento principal da Equipe de Desenvolvimento. Seu objetivo primário é a implementação das aplicações. Portanto, o Programador deve possuir amplo conhecimento em várias linguagens de programação e ótimo conhecimento em lógica computacional. Além disso, ele deve conhecer o sistema no qual está envolvido, do nível conceitual ao operacional. Não obstante, o Programador deve estar hábil a seguir padrões e cumprir prazos.
- Programador Chefe: é o responsável pela Equipe de Desenvolvimento. Além de desenvolver, ele gerencia a equipe, avalia os resultados e define diretrizes. Está em constante contato com o Analista de Requisitos.
- Projetista de Banco de Dados: é o responsável por definir a estrutura final do banco de dados. Para tanto, a pessoa que desempenha este papel deve possuir amplo conhecimento do banco de dados do sistema, estando a par das suas restrições e possibilidades.
- Programador de Testes: é o responsável pela criação de modelos de testes para serem utilizados nas atividades de teste pela Equipe de Desenvolvimento. O Programador de Testes também participa das atividades de programação.
- Projetista de Interfaces: é o responsável por criar a interface de interação entre a aplicação e o usuário da mesma. O Projetista de Interfaces deve ter ótima capacidade em interpretar funcionalidades em botões, campos e informações, de forma que o usuário consiga utilizar a aplicação de maneira independente. Além disso, deve possuir amplo conhecimento de todo o sistema, a fim de conseguir criar soluções padrões.
- Redator Técnico: é o responsável pela criação da documentação do *release* para o usuário final. O Redator Técnico deve ter ótimos domínios de gramática e da língua portuguesa. Além disso, é desejável que ele possua conhecimentos em outras línguas, ainda que não seja o responsável pela tradução dos documentos.
- Revisor de Documentação: é o responsável pela revisão de documentos voltados aos usuários do sistema (geralmente os documentos com as novidades do *release* e manuais do usuário). O Revisor de Documentação deve possuir amplo conhecimento

da língua portuguesa, uma vez que ele será o último a avaliar a documentação e qualquer erro não identificado será de sua responsabilidade.

- Tradutor: é o responsável por traduzir os documentos voltados aos usuários do sistema (geralmente os documentos com as novidades do *release* e manuais do usuário) para outras línguas que forem necessárias. Para tanto, ele deve possuir amplo conhecimento nestas línguas, uma vez que será o único a traduzir e revisar os documentos. Atualmente, o sistema é desenvolvido apenas na língua portuguesa mas a gerência já possui planos de estendê-lo para outras línguas, inclusive com alguns possíveis clientes já em vista.

3.4.2 Disciplinas

As Disciplinas do modelo são importantes para dividir as atividades e tarefas e também para proporcionar um controle melhor de cada artefato e de cada papel.

Das nove disciplinas do RUP, oito foram mantidas e adaptadas à realidade do modelo. Por isso, seus nomes também foram alterados. A disciplina de Distribuição não fará parte do modelo, uma vez que este se concentra nas atividades de desenvolvimento do sistema.

Os diagramas que representaram as disciplinas usam as definições abaixo para mostrar o relacionamento entre as atividades e os papéis:

- <<perform>>: representa o papel que executa a atividade efetivamente.
- <<assist>>: representa o papel que auxilia na execução da atividade.

A Figura 6 apresenta o fluxo de informações entre todas as disciplinas.

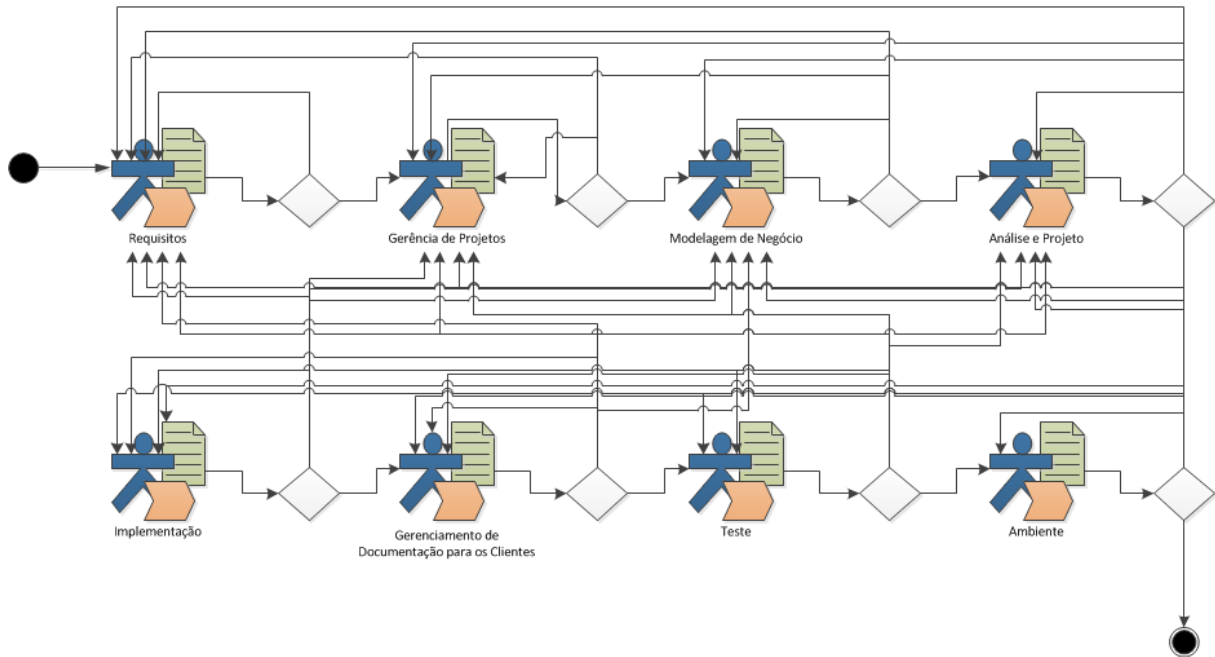


Figura 6 - Diagrama de Atividades

3.4.2.1 Gerência de Projetos

O principal objetivo desta disciplina – análoga à disciplina de mesmo nome do RUP – é gerenciar as metas conforme os prazos estabelecidos. O gerenciamento dos projetos requer que todas as tarefas do mesmo sejam especificadas, possuam prazos iniciais e finais, tenham marcos definidos e sejam passíveis de avaliação, pela Equipe de Desenvolvimento e pela Gerência.

Aqui são adotadas duas práticas do XP: o “Ciclo Trimestral” e a “Folga”. A primeira diz respeito ao planejamento dos *releases* conforme um tema específico. Por exemplo, se for decidido que o próximo *release* será financeiro, não faz sentido incluir uma aplicação contábil. Esta prática também consiste na análise e avaliação dos *releases* anteriores, sempre buscando a melhoria. A segunda prática a ser adotada – a Folga – tem a ver com as estimativas de erro nos planejamentos realizados. Na delimitação de prazos, é importante que sejam adicionadas folgas no processo de desenvolvimento. Estas folgas consistem em tarefas que, se necessário, podem ser descartadas sem que comprometam o projeto como um todo.

A Disciplina Gerência de Projetos é detalhada no Diagrama de Pacotes representado na Figura 7, na qual é possível visualizar os elementos (papéis, atividades e artefatos) que a constituem.

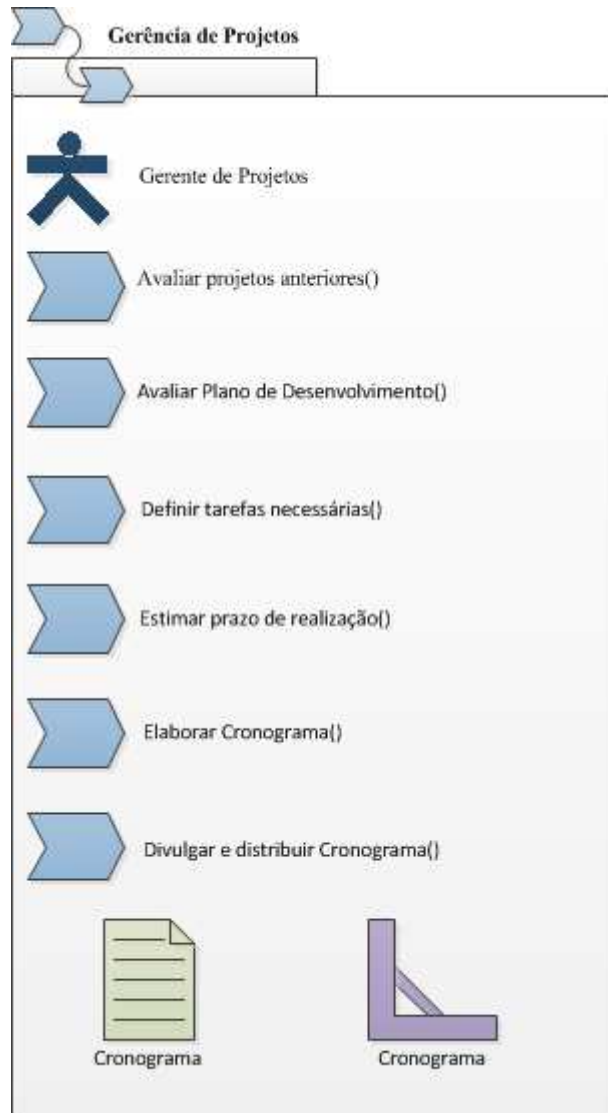


Figura 7 - Diagrama de Pacotes: Gerência de Projetos

A Disciplina Gerência de Projetos é constituída pelas seguintes atividades:

- Avaliar projetos anteriores, através dos quais será possível realizar uma estimativa mais próxima da realidade;
- Avaliar o Plano de Desenvolvimento, no qual estarão elencados todos os requisitos do novo *release*;
- Definir tarefas necessárias para cada membro da equipe;
- Estimar um prazo de realização para cada tarefa;
- Elaborar cronograma, geralmente através de uma ferramenta específica, como o Microsoft Project, por exemplo. Algumas empresas – como a do estudo em questão – possuem um gerenciador de projetos próprio, através do qual fazem a programação das atividades;

- Divulgar e Distribuir cronograma a todos os membros da equipe.

Nesta Disciplina há o Gerente de Projetos, responsável pela elaboração do cronograma, visto que ele possui a experiência necessária para este fim. O Programador Chefe também participa dando suporte a algumas atividades. O relacionamento entre as atividades e os papéis é ilustrado no Diagrama de Casos de Uso da Figura 8.

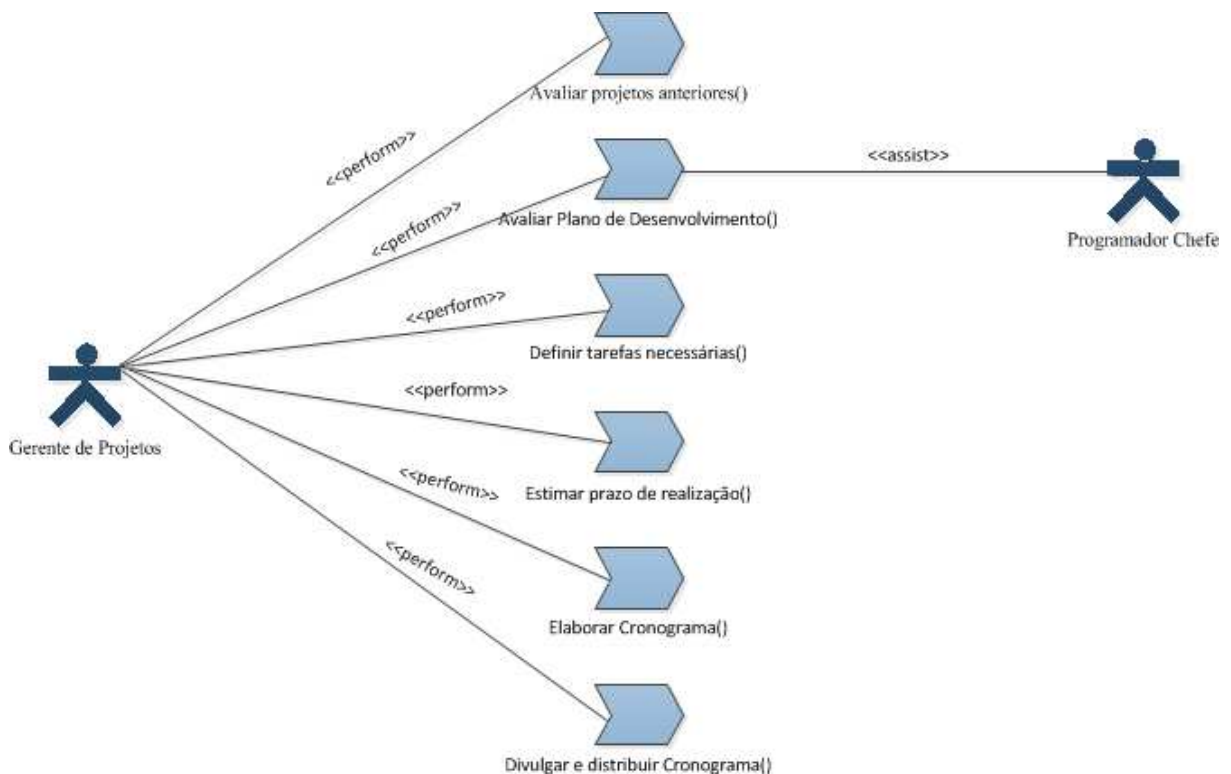


Figura 8 - Diagrama de Casos de Uso: Gerência de Projetos

O artefato gerado nesta Disciplina é o Cronograma do Projeto, que implementa as tarefas e prazos necessários para a execução de cada uma. Ele é criado com base no Plano de Desenvolvimento, gerado na Disciplina Requisitos. Muitas vezes são criados vários cronogramas, pois novos requisitos constantemente são incrementados no desenvolvimento do *release*. Um *template* do cronograma pode ser visto no Apêndice A.

O Diagrama de Classes da Figura 9 ilustra o relacionamento dos papéis e artefatos envolvidos, definindo assim, os responsáveis pela geração do mesmo.



Figura 9 - Diagrama de Classes: Gerência de Projetos

O Diagrama de Atividades da Figura 10 representa o fluxo das atividades, mostrando a precedência das mesmas.

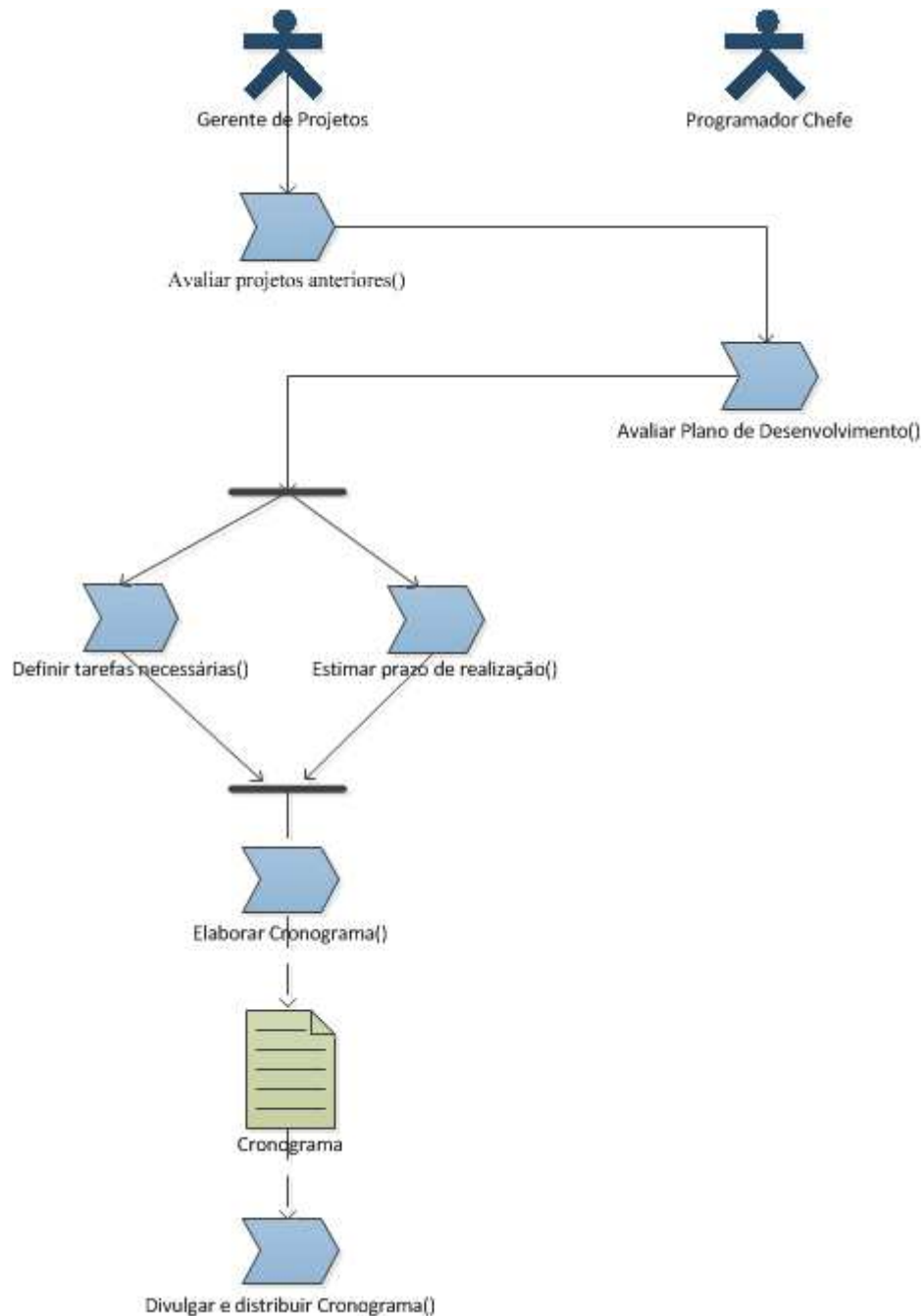


Figura 10 - Diagrama de Atividades: Gerência de Projetos

3.4.2.2 Requisitos

A disciplina Requisitos seleciona, avalia e define os requisitos do sistema, principalmente no que tange às especificações do cliente. O termo “cliente” aqui pode ser entendido como sendo o responsável pelos itens solicitados (novas aplicações, por exemplo). No trabalho proposto, em cerca de 90% das vezes o cliente é a própria Gerência da Empresa que, através de estudos e análises aprofundadas, define os requisitos básicos de cada novo *release* que será lançado. O cliente do sistema, por sua vez, raramente é o responsável pelas solicitações, mas há casos nos quais isto também ocorre.

As atividades desta disciplina são realizadas em forma de reuniões entre as pessoas envolvidas, para que todas as dúvidas sejam sanadas e não haja diferentes entendimentos de um mesmo requisito. As reuniões devem ser realizadas em um ambiente tranquilo e livre de interrupções, uma vez que elas requerem uma análise aprofundada de cada requisito. É interessante o uso de papéis para anotações.

A Disciplina Requisitos é detalhada no Diagrama de Pacotes representado na Figura 11, na qual é possível visualizar os elementos (papéis, atividades e artefatos) que a constituem.



Figura 11 - Diagrama de Pacotes: Requisitos

A Disciplina Requisitos é constituída pelas seguintes atividades:

- Definir todos os itens que farão parte do *release*, sejam novos recursos, correções de erros ou alterações em aplicações existentes, sendo apresentada uma visão superficial sobre cada item.
- Detalhar cada item apontado, explicando a importância do mesmo, o motivo dele estar sendo incluído no *release* e as principais alterações que isto causará no sistema como um todo. Muitas vezes esta atividade não conterà todos os itens do *release*, sendo

necessário realizá-la novamente posteriormente, quantas vezes forem necessárias, até que todos os itens sejam completamente definidos;

- Sugerir e propor melhorias;
- Elaborar Plano de Desenvolvimento;
- Criar ou Atualizar Diagramas de Casos de Uso. A atualização ocorrerá quando o item elencado for uma adição ou alteração em um recurso/aplicação já existente no sistema, que acarrete uma mudança significativa no mesmo.
- Avaliar Diagramas de Casos de Uso;

Nesta Disciplina há o Analista de Requisitos, responsável pela definição de todos os requisitos que farão parte do novo release. O Programador Chefe, por sua vez, avaliará todos os requisitos no âmbito do desenvolvimento, através da sua experiência em projetos anteriores, uma vez que ele será o responsável em passar as tarefas para os programadores iniciarem a implementação.

O relacionamento entre as atividades e os papéis é ilustrado no Diagrama de Casos de Uso da Figura 12.

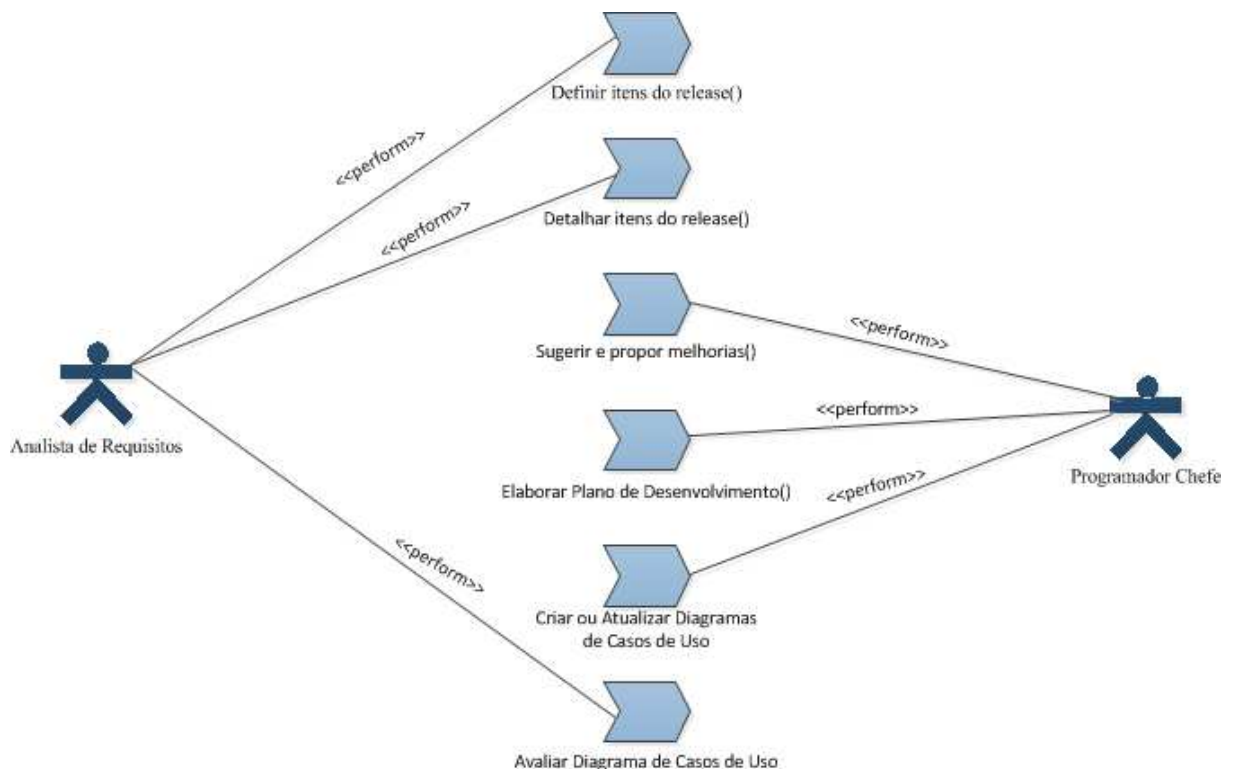


Figura 12 - Diagrama de Casos de Uso: Requisitos

O primeiro artefato a ser gerado é o Plano de Desenvolvimento, que consiste num documento contendo todas as tarefas, em nível de desenvolvimento, que deverão ser realizadas para a conclusão do *release*. Um *template* do Plano de Desenvolvimento pode ser visto no Apêndice B. Também serão gerados os Diagramas de Casos de Uso necessários, a fim de confirmar o entendimento dos requisitos definidos e facilitar a implementação dos mesmos. Um *template* de Caso de Uso está disponível no Apêndice C.

O Diagrama de Classes da Figura 13 ilustra o relacionamento dos papéis e artefatos envolvidos, definindo assim, os responsáveis pela geração dos mesmos.

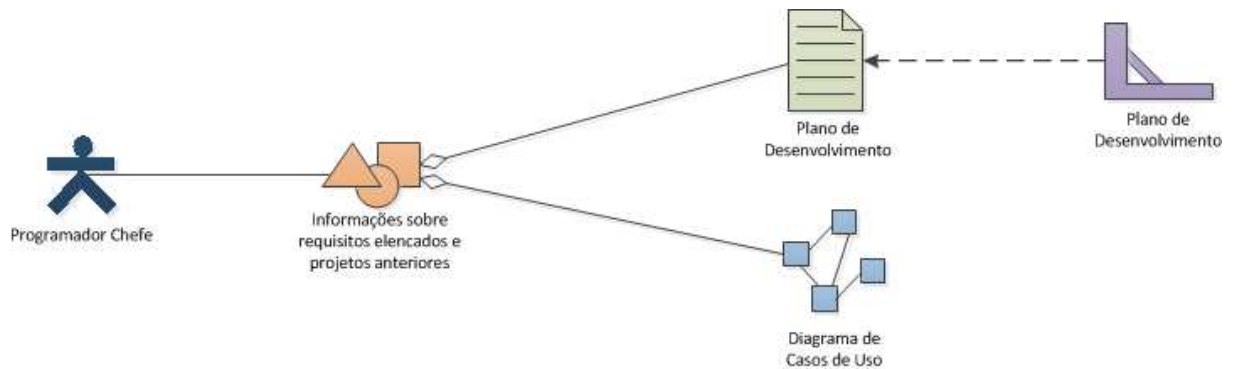


Figura 13 - Diagrama de Classes: Requisitos

O Diagrama de Atividades da Figura 14 representa o fluxo das atividades, mostrando a precedência das mesmas.

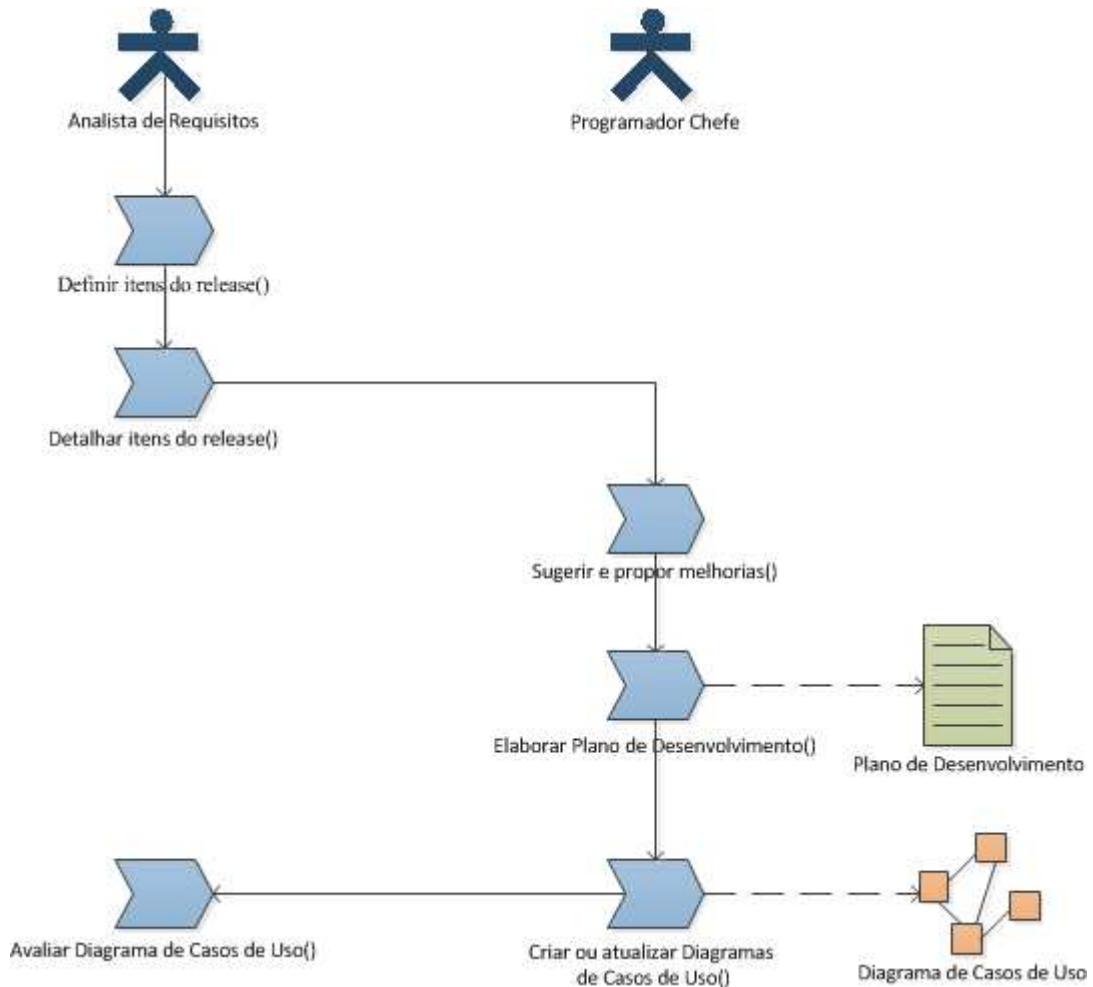


Figura 14 - Diagrama de Atividades: Requisitos

3.4.2.3 Modelagem de Negócio

Esta disciplina tem o objetivo de entender as regras de negócio das novas aplicações do *release*. É muito comum que, para se desenvolver alguma aplicação, sejam necessários estudos aprofundados em determinadas áreas, como contabilidade, finanças, redes, etc. A disciplina agrega novos conhecimentos e dá suporte ao desenvolvimento de uma maneira eficiente. A falta deste conhecimento pode provocar inconsistências e, em um estágio mais avançado do desenvolvimento, problemas que deverão ser solucionados a um custo muito maior do que se tivessem sido evitados no início.

A Disciplina Modelagem de Negócio é detalhada no Diagrama de Pacotes representado na Figura 15, na qual é possível visualizar os elementos (papéis, atividades e artefatos) que a constituem.

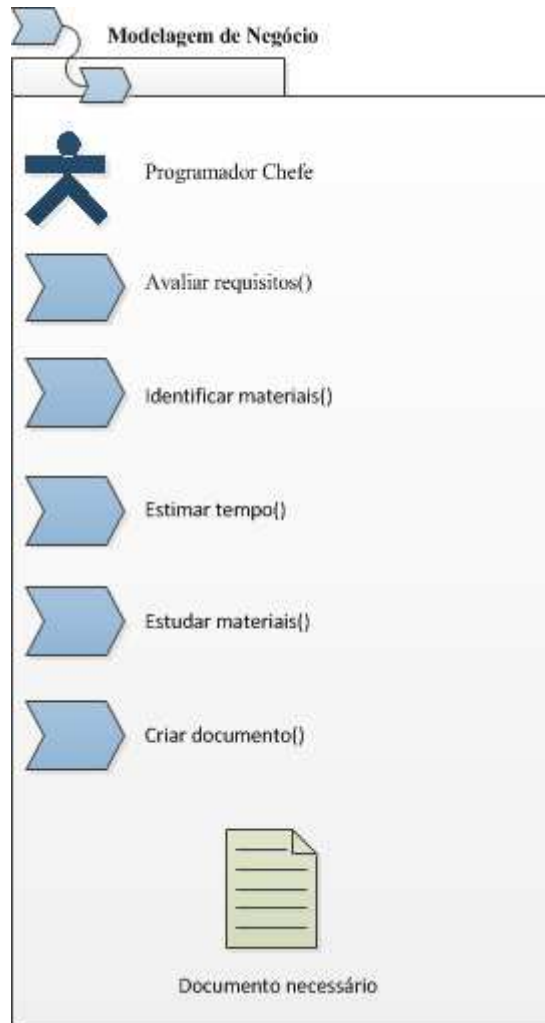


Figura 15 - Diagrama de Pacotes: Modelagem de Negócio

A Disciplina Modelagem de Negócio é constituída pelas seguintes atividades:

- Avaliar requisitos que necessitem de um estudo aprofundado sobre determinado assunto;
- Identificar materiais necessários – geralmente manuais, apostilas, artigos, etc. – que darão suporte ao entendimento de todas as regras de negócio necessárias;
- Estimar tempo necessário para entendimento das regras de negócio;
- Estudar materiais identificados. Esta atividade pode durar bastante tempo, dependendo do nível de estudo requerido ou da complexidade do assunto;
- Criar documento específico, quando necessário. A criação deste documento depende da regra de negócio que está sendo estudada e não é obrigatória. Uma tabela relacionando intervalos de valores por atributos – para nota fiscal eletrônica, por exemplo – é um exemplo de documento que pode ser gerado. Há casos em que não é gerado documento algum.

Nesta Disciplina há o Programador Chefe, responsável pelo entendimento das regras de negócio a fim de direcionar o desenvolvimento de maneira eficaz e eficiente. O relacionamento entre as atividades e os papéis é ilustrado no Diagrama de Casos de Uso da Figura 16.

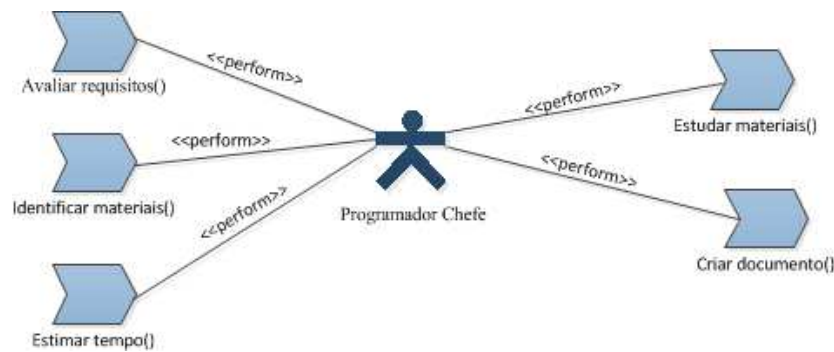


Figura 16 - Diagrama de Casos de Uso: Modelagem de Negócio

Com relação aos artefatos gerados, pode ser que não haja necessidade de gerar nenhum. Quando são necessários, o Programador Chefe é o responsável pela elaboração dos mesmos.

O Diagrama de Classes da Figura 17 ilustra o relacionamento dos papéis e artefatos envolvidos, definindo assim, os responsáveis pela geração dos mesmos.

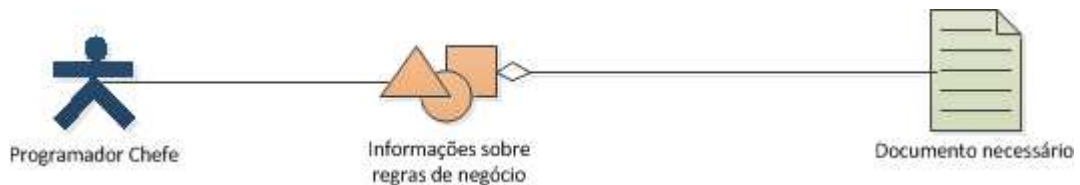


Figura 17 - Diagrama de Classes: Modelagem de Negócio

O Diagrama de Atividades da Figura 18 representa o fluxo das atividades, mostrando a precedência das mesmas.

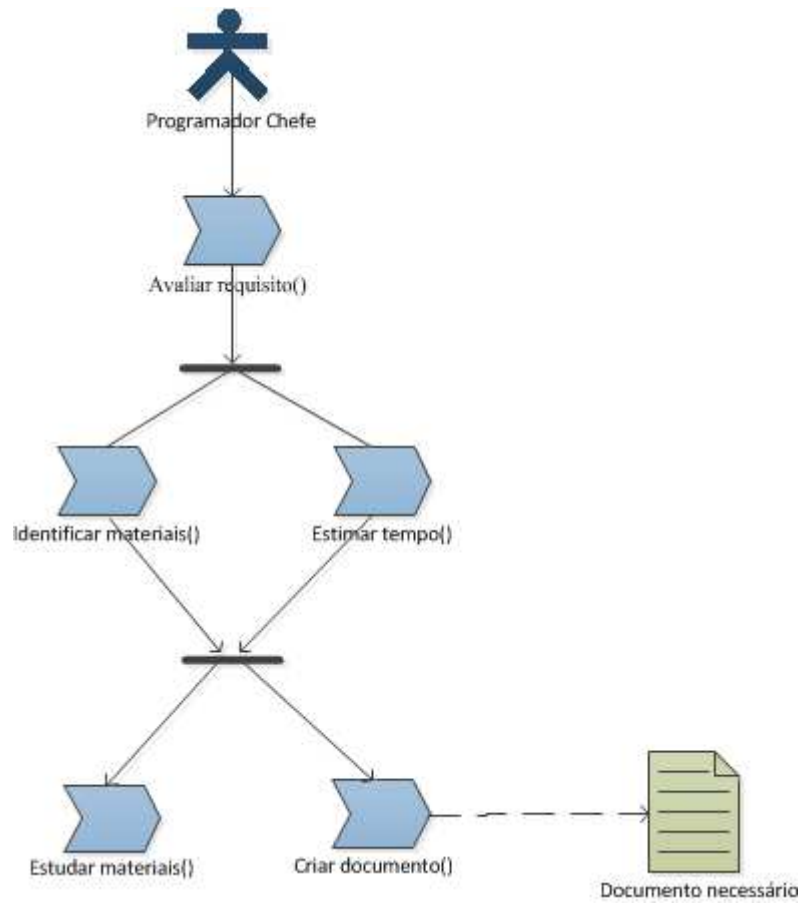


Figura 18 - Diagrama de Atividades: Modelagem de Negócio

3.4.2.4 Análise e Projeto

Recursos aparentemente simples, como a adição de um novo campo para guardar alguma informação ou o repasse de informações para outra aplicação, podem desencadear uma série de complicações conceituais e estruturais em um sistema. A disciplina de Análise e Projeto visa encontrar soluções para todos os requisitos que deverão fazer parte do *release*.

Em um sistema relativamente grande, a quantidade de aplicações, ligações e dependências entre elas pode se tornar um quebra-cabeça muito complicado de se resolver. Por isso, a Análise e Projeto avalia o sistema da maneira como está hoje e como estará futuramente, levando em consideração aspectos como número de clientes e número de informações armazenadas pelo banco de dados.

Aqui é adotada uma prática do XP, chamada de “Design Incremental”, que tem por objetivo a implementação de um design simples que suporte todas as necessidades da aplicação. Com o

incremento das funcionalidades e a evolução da aplicação e do sistema, o design deve ser evoluído e melhorado.

A Disciplina Análise e Projeto é detalhada no Diagrama de Pacotes representado na Figura 19, na qual é possível visualizar os elementos (papéis, atividades e artefatos) que a constituem.

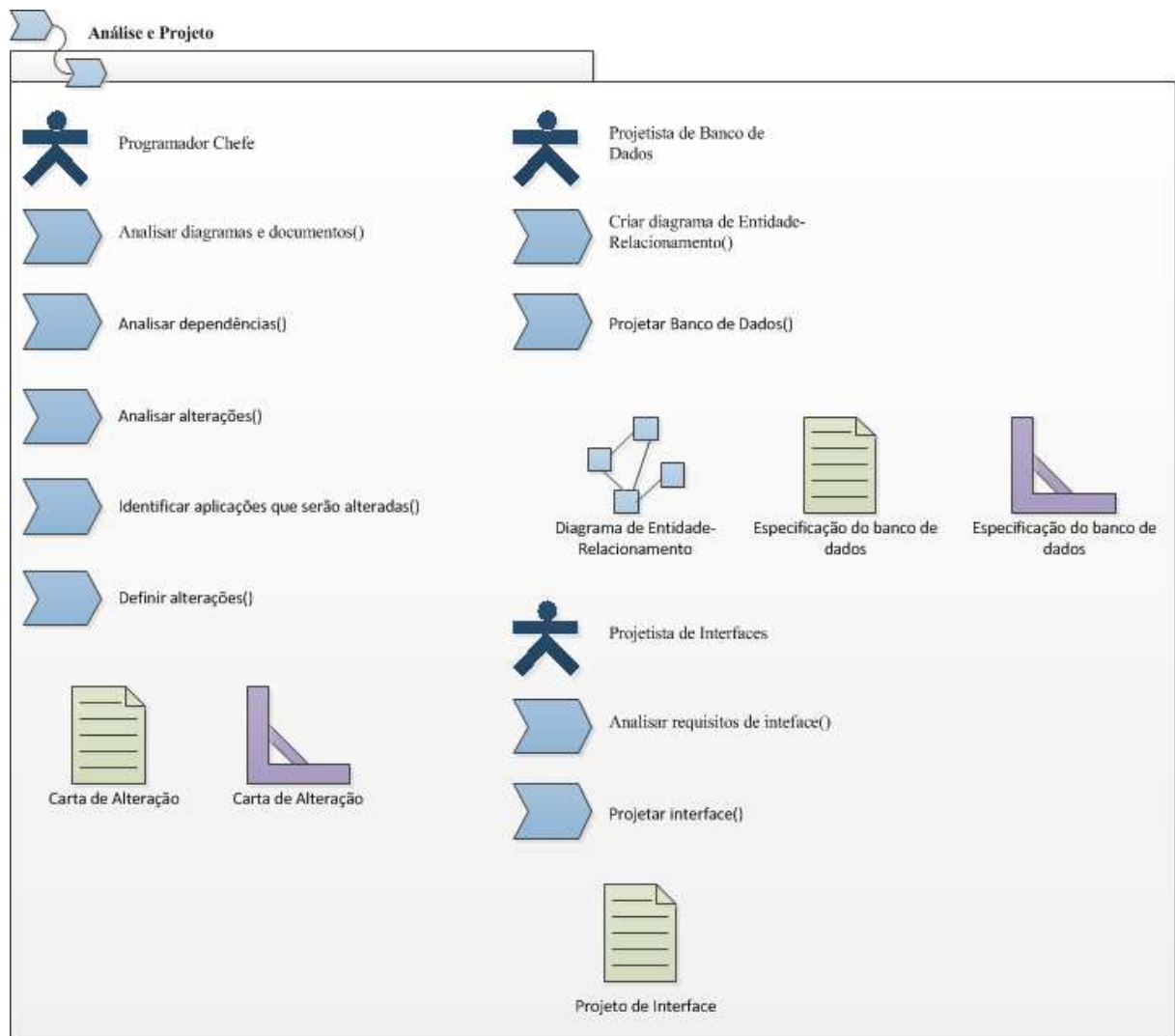


Figura 19 - Diagrama de Pacotes: Análise e Projeto

A Disciplina Análise e Projeto é constituída pelas seguintes atividades:

- Analisar Diagramas de Casos de Uso e documentos referentes às Regras de Negócio;
- Criar Diagrama de Entidade-Relacionamento. Esta atividade pode ser realizada com a ajuda de um software como o Microsoft Visio;
- Projetar Banco de Dados;

- Analisar requisitos de interface;
- Projetar Interface. Esta modelagem geralmente é realizada à mão, em uma folha de papel, conforme os requisitos do *release*;
- Analisar dependências entre módulos existentes. O projeto de um *release* muitas vezes demanda o incremento de funcionalidades em aplicações já existentes, por isto esta análise é fundamental;
- Analisar alterações de Banco de Dados e Interface. Muitas vezes novos recursos requerem alterações importantes em aplicações já existentes, como a criação de novos campos, alteração do banco de dados ou mesmo alterações de layout;
- Definir aplicações que sofrerão alterações e de quais tipos;
- Definir todas as alterações identificadas nas duas atividades anteriores. Esta atividade é realizada através de reuniões, uma vez que as alterações identificadas podem necessitar de conhecimentos de vários papéis diferentes.

Nesta Disciplina há três papéis principais. O Projetista de Banco de Dados é o responsável pela modelagem do banco de dados, incluindo aqui todas as alterações que se façam necessárias. O Projetista de Interfaces tem como função modelar a interface das aplicações, ou seja, nome dos campos, botões de ajuda, cores, posições dos elementos, etc. ele também é o responsável por alterar as interfaces existentes, quando necessário. O Programador Chefe é o responsável por analisar as dependências entre os módulos, com base nos novos requisitos do *release*. Assim, esta análise consiste em verificar alterações necessárias de interfaces, banco de dados ou regras de negócio. Um quarto papel, o do Analista de Requisitos, pode ser presente dando suporte em determinadas atividades.

O relacionamento entre as atividades e os papéis é ilustrado no Diagrama de Casos de Uso da Figura 20.



Figura 20 - Diagrama de Casos de Uso: Análise e Projeto

Esta atividade gera quatro artefatos. O primeiro é o Diagrama de Entidade-Relacionamento, que tem por finalidade servir de base para a modelagem do banco de dados. Através do Diagrama de Entidade-Relacionamento, a especificação do banco de dados é criada em uma planilha do Excel com todas as tabelas e informações úteis para a criação do mesmo. Um *template* desta especificação pode ser visto no Apêndice D. Esta especificação do banco de dados também pode ser criada com a ajuda de uma ferramenta própria para isso, ficando a critério da equipe escolher. O terceiro artefato a ser criado é o Projeto de Interface, que consiste num desenho mostrando como deve ser o layout final das aplicações. Podem ser gerados vários Projetos de Interface diferentes, um para cada aplicação. Um *template* deste artefato pode ser visto no Apêndice E. O quarto artefato a ser gerado é a Carta de Alteração,

que consiste num documento contendo todas as alterações que determinado módulo/aplicação deverá ter. Além disso, informações como impacto das alterações no sistema como um todo e quaisquer outras informações necessárias à Equipe de Desenvolvimento devem estar presentes neste documento. Podem ser geradas várias Cartas de Alteração diferentes, uma para cada aplicação. Um *template* deste artefato pode ser visto no Apêndice G.

O Diagrama de Classes da Figura 21 ilustra o relacionamento dos papéis e artefatos envolvidos, definindo assim, os responsáveis pela geração dos mesmos.

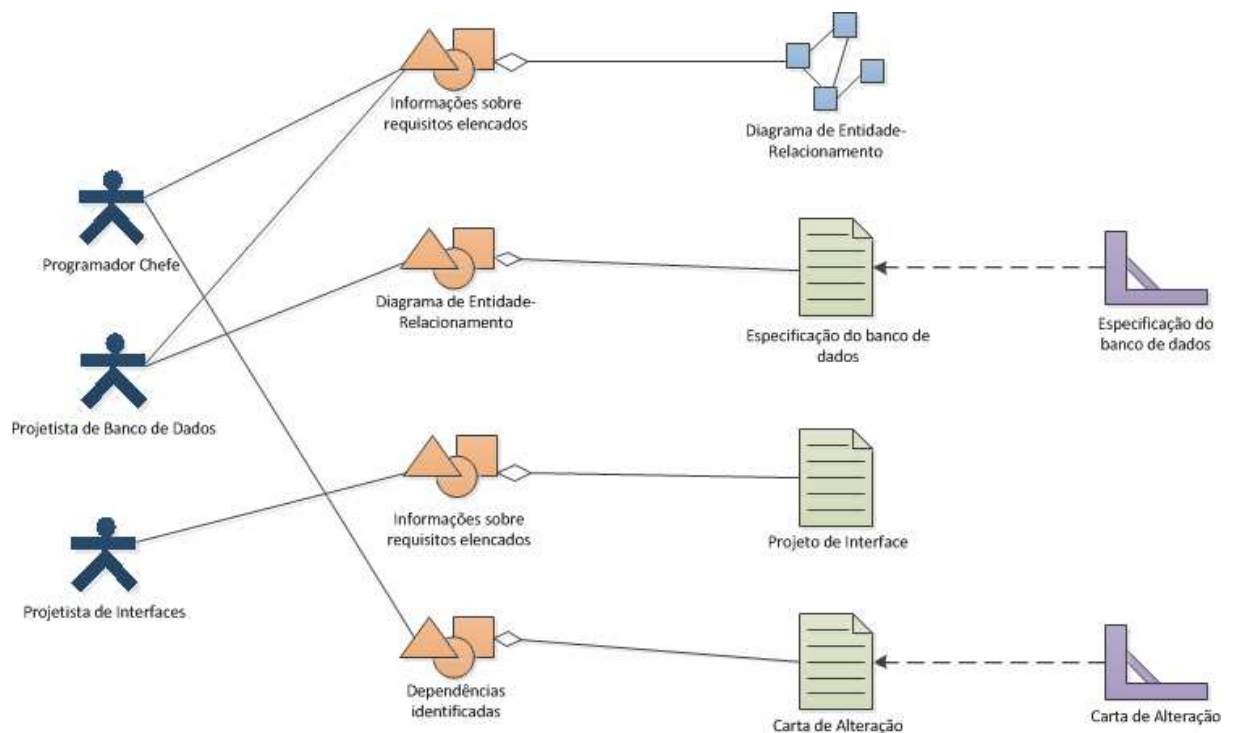


Figura 21 - Diagrama de Classes: Análise e Projeto

O Diagrama de Atividades da Figura 22 representa o fluxo das atividades, mostrando a precedência das mesmas.

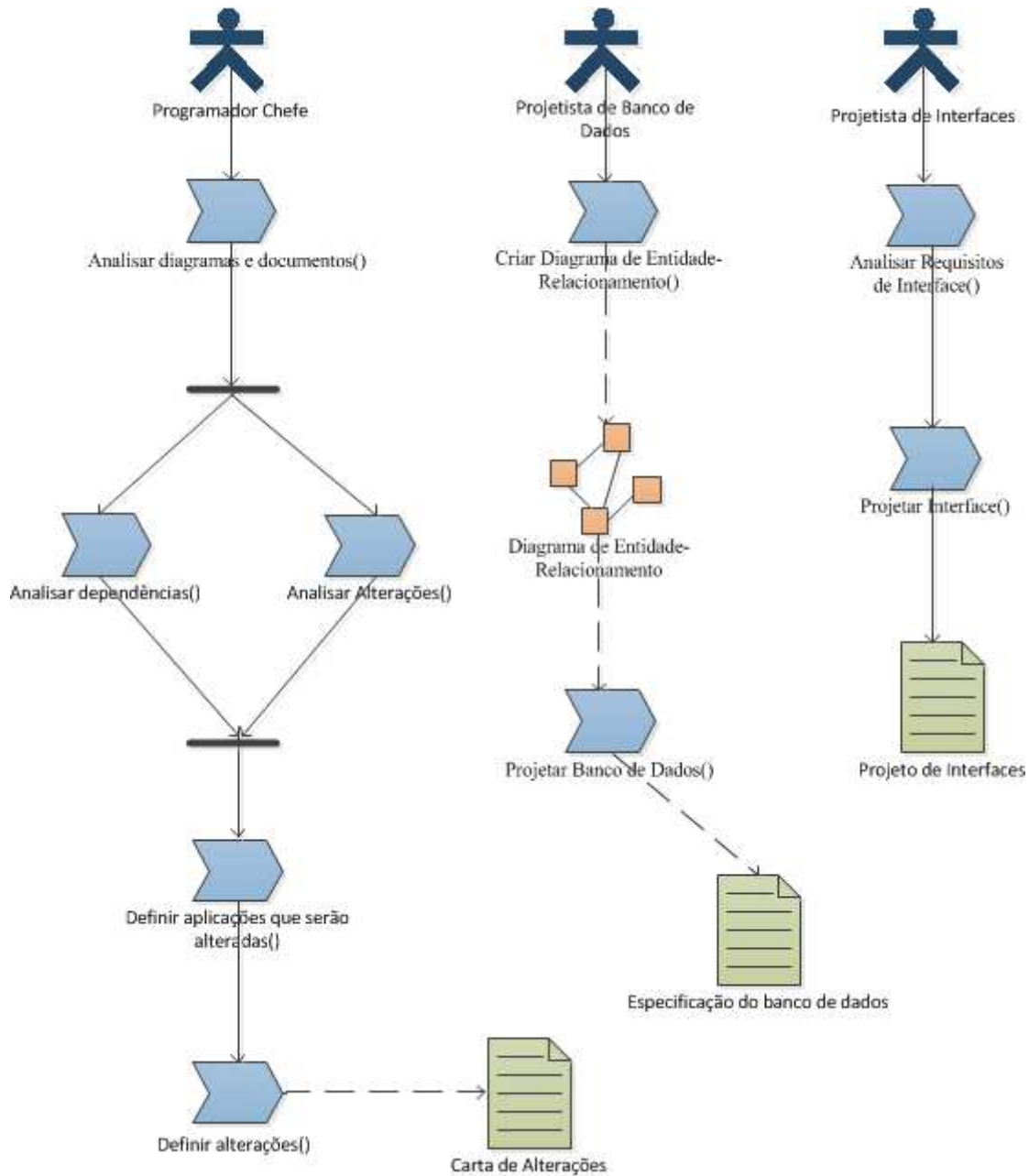


Figura 22 - Diagrama de Atividades: Análise e Projeto

3.4.2.6 Implementação

A disciplina de Implementação envolve a construção do *release*, através da geração dos códigos-fonte das aplicações. Ela envolve as atividades mais demoradas e trabalhosas. Além disso, aqui há um aumento na quantidade de pessoas – geralmente programadores – gerando certa complicação no gerenciamento da equipe. Os artefatos gerados na Disciplina Análise e Projeto constituem a base para realizar as atividades desta disciplina.

Uma prática do XP a ser adotada aqui é o “Sentar Junto”, que consiste em um ambiente de trabalho no qual os membros da equipe possam trabalhar juntos. Além de melhorar a comunicação – um dos valores elencados como fundamentais na elaboração do modelo – entre os membros da equipe, esta prática viabilizará a identificação de habilidades inerentes a cada membro. Conforme a equipe cresça, talvez seja interessante estudar a possibilidade de dividir os membros em áreas de especificidades.

Na maioria dos casos, as atividades pertencentes a esta disciplina são as que demandam mais tempo para serem realizadas.

A Disciplina Implementação é detalhada no Diagrama de Pacotes representado na Figura 23, na qual é possível visualizar os elementos (papéis, atividades e artefatos) que a constituem.

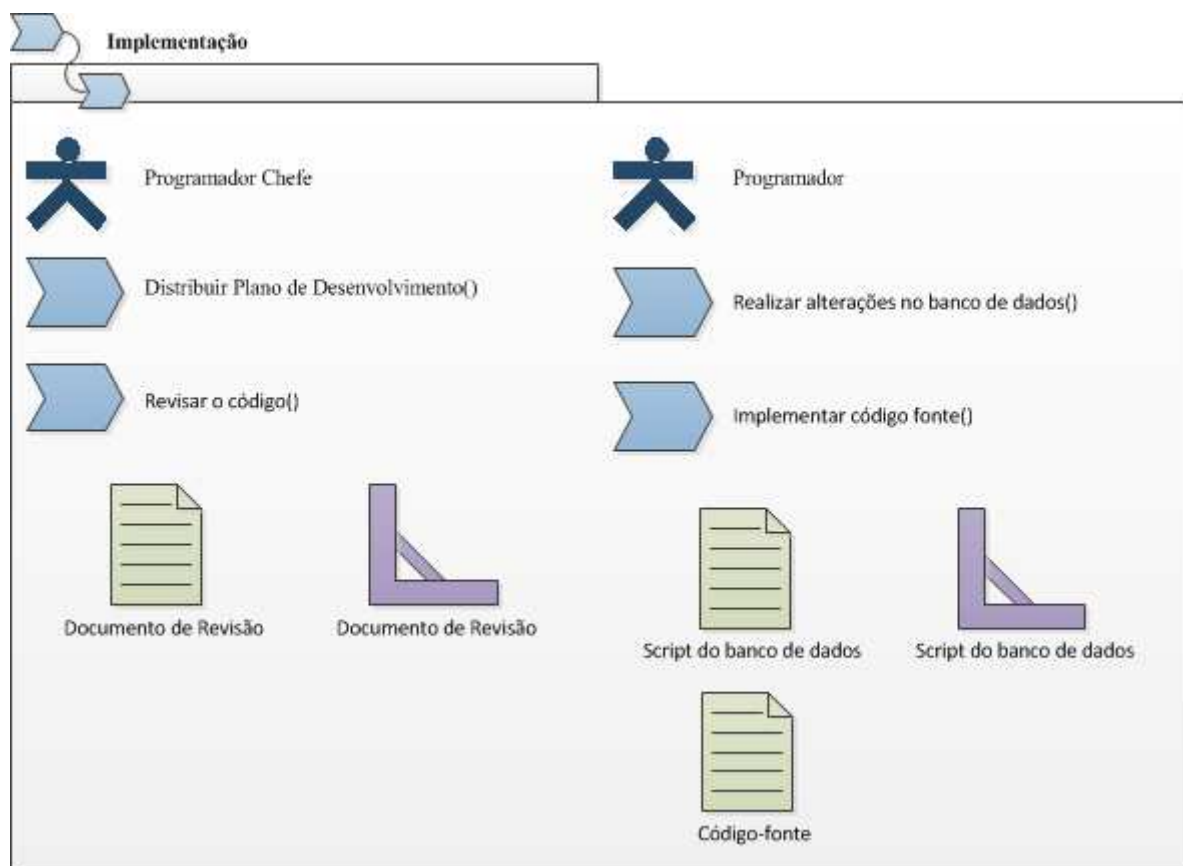


Figura 23 - Diagrama de Pacotes: Implementação

A Disciplina Implementação é constituída pelas seguintes atividades:

- Distribuir Plano de Desenvolvimento para a equipe e sanar possíveis dúvidas;

- Realizar alterações no banco de dados, como criação de novas tabelas, alteração em tabelas já existentes, etc;
- Implementar código-fonte. Um item importante nesta atividade é a necessidade do estabelecimento de padrões para a implementação. Levando-se em conta que cada Programador tem uma personalidade própria e, portanto, uma maneira própria de programar, o não estabelecimento de um padrão pode acarretar na construção de um sistema codificado de maneira totalmente heterogênea. Uma possível manutenção futura se transformará numa tarefa árdua e demorada. Além disso, o estabelecimento de uma padronização garante que todos conseguirão fazer análises de qualquer aplicação, mesmo que não seja o responsável pela codificação da mesma. Esta atividade requer total atenção do Programador e por isso, o ambiente físico no qual se encontra a Equipe de Desenvolvimento deve ser o mais bem preparado possível;
- Revisar o código. Esta atividade tem por objetivo identificar inconsistências e erros na codificação. Além disso, podem ser sugeridas melhorias nos códigos, por exemplo, deixando a aplicação mais leve;

Nesta atividade há dois papéis principais. O Programador Chefe é o responsável por distribuir e explicar o Plano de Desenvolvimento a toda Equipe de Desenvolvimento. O Programador é o responsável pela implementação dos códigos-fonte, que serão revisados, posteriormente, pelo Programador Chefe.

O relacionamento entre as atividades e os papéis é ilustrado no Diagrama de Casos de Uso da Figura 24.

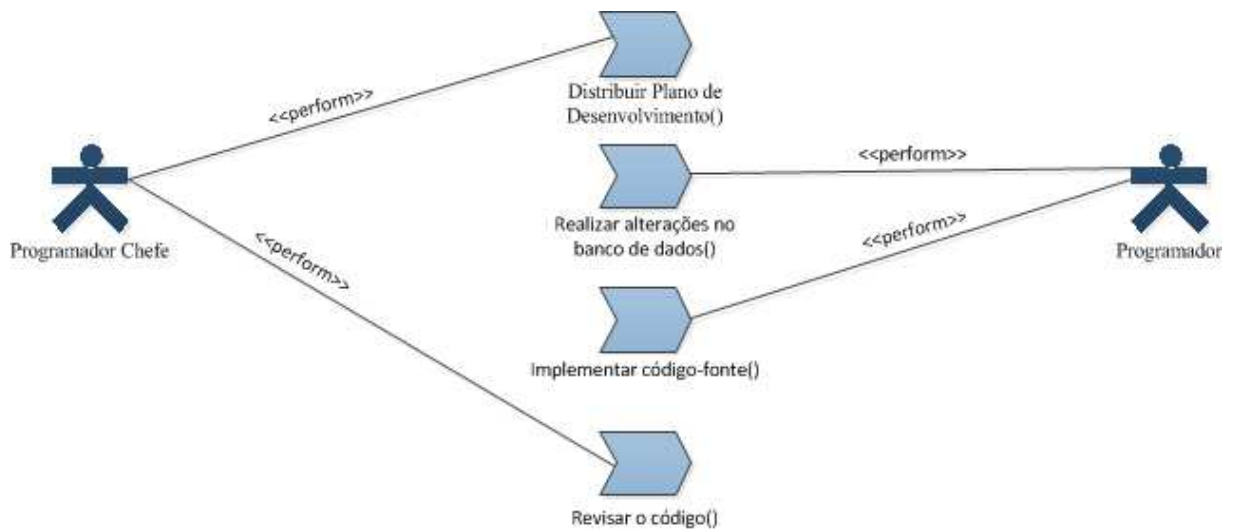


Figura 24 - Diagrama de Casos de Uso: Implementação

Esta disciplina gera três artefatos. O primeiro deles é o script de alterações no banco de dados, que consistem num arquivo de texto com as linhas de código para manipulação do banco. Um *template* deste artefato pode ser visualizado no Apêndice H. Ao final da programação os códigos-fonte constituem o segundo artefato criado, sendo a principal base de documentação para a Equipe de Desenvolvimento. É importante que a implementação tenha comentários, explicando cada parte do código. Os comentários ajudam na interpretação das linhas de comando e economizam tempo em análises mais aprofundadas. Um *template* deste artefato pode ser visto no Apêndice I. O terceiro artefato a ser criado é chamado Documento de Revisão, que contém alterações que necessitem ser feitas nos códigos-fonte. Cada Programador recebe um documento distinto, após a revisão realizada pelo Programador Chefe. Este documento também serve para concluir a atividade do Programador, caso não seja necessário efetuar nenhuma alteração no código. O *template* do Documento de Revisão pode ser visto no Apêndice J.

O Diagrama de Classes da Figura 25 ilustra o relacionamento dos papéis e artefatos envolvidos, definindo assim, os responsáveis pela geração dos mesmos.

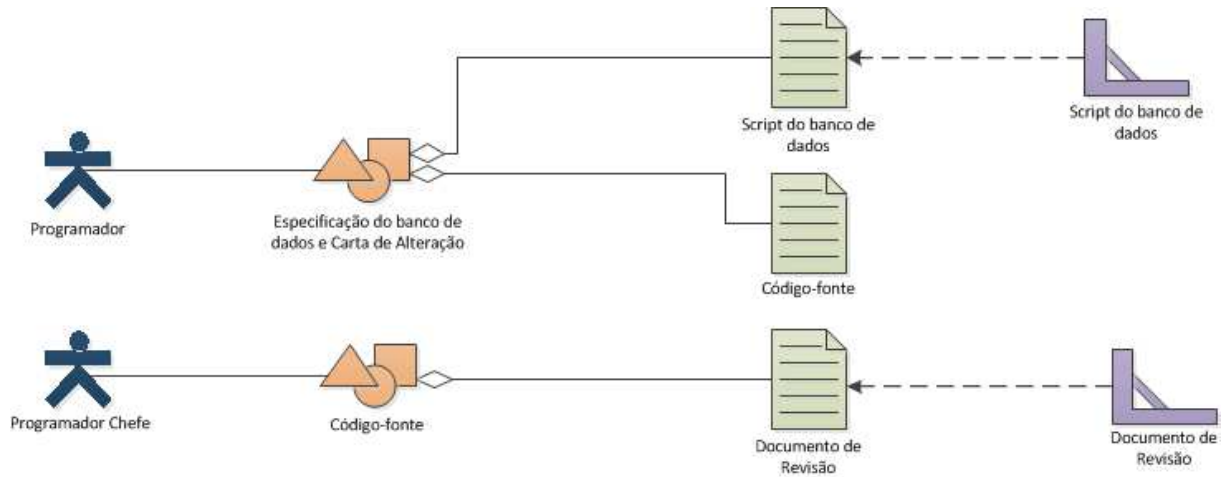


Figura 25 - Diagrama de Classes: Implementação

O Diagrama de Atividades da Figura 26 representa o fluxo das atividades, mostrando a precedência das mesmas.

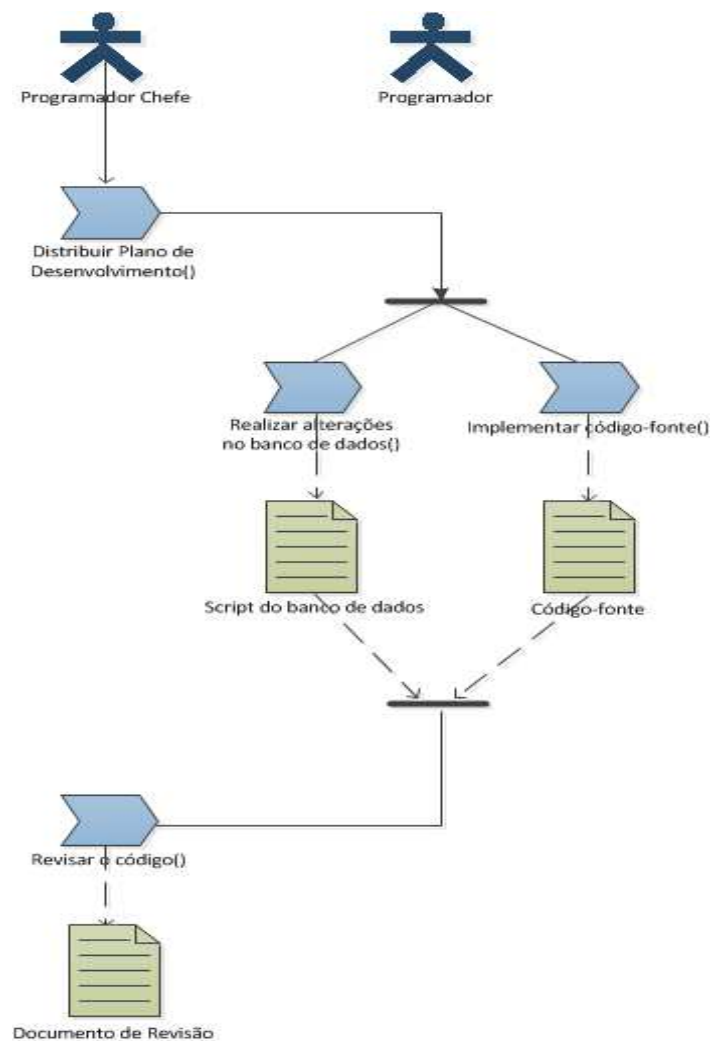


Figura 26 - Diagrama de Atividades: Implementação

3.4.2.7 Gerenciamento de Documentação para os Clientes

Esta disciplina é análoga à Configuração e Controle de Mudanças do RUP. Ela foi alterada para lidar com a documentação para os clientes. Assim, manuais do usuário e *changelogs* fazem parte de seu escopo de estudo. A disciplina no RUP gerenciava o controle de versões do projeto mas este item foi retirado da mesma – neste modelo – uma vez que o modelo proposto já é focado no desenvolvimento de versões de um sistema, ou seja, o controle das mesmas pode ser considerado “automático”.

A Disciplina Gerenciamento de Documentação para os Clientes é detalhada no Diagrama de Pacotes representado na Figura 27, na qual é possível visualizar os elementos (papéis, atividades e artefatos) que a constituem.

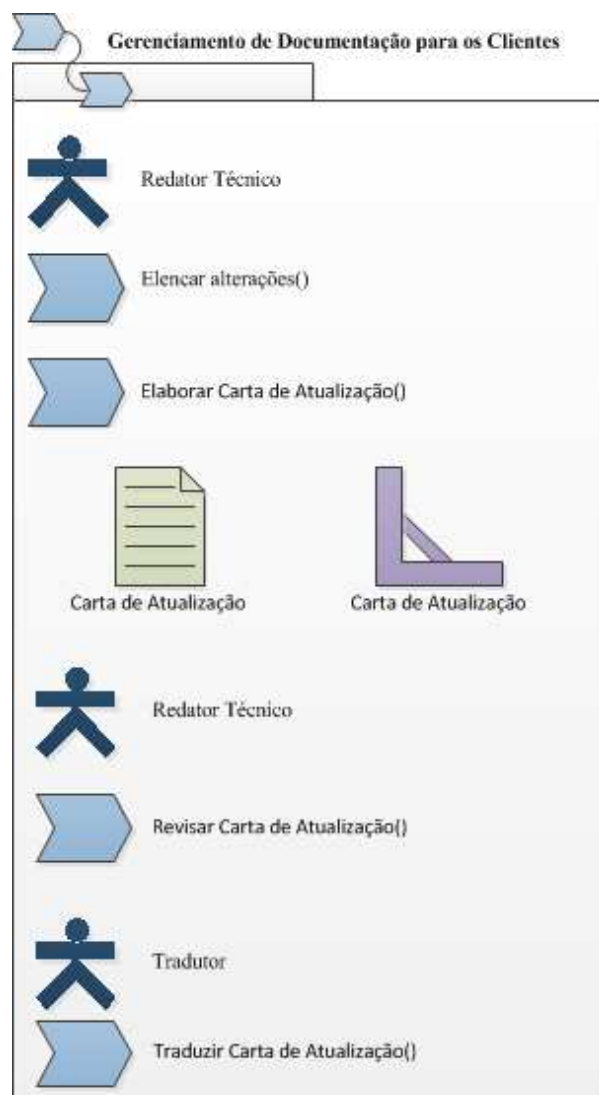


Figura 27 - Diagrama de Pacotes: Gerenciamento de Documentação para os Clientes

A Disciplina Gerenciamento de Documentação para os Clientes é constituída pelas seguintes atividades:

- Elencar todas as alterações, novos recursos e correções de erros do *release*;
- Elaborar Carta de Atualização;
- Revisar Carta de Atualização;
- Traduzir Carta de Atualização, quando necessário;

O Redator Técnico é o responsável por redigir a Carta de Atualização. Quando esta estiver finalizada, o Revisor de Documentação irá revisá-la, identificando e corrigindo erros gramaticais, bem como adequando a linguagem a determinados contextos, quando necessário. Ele também é o responsável por deixar a Carta de Atualização no seu formato padrão. Quando for necessário a tradução deste documento para outra língua, o Tradutor será o encarregado desta atividade.

O relacionamento entre as atividades e os papéis é ilustrado no Diagrama de Casos de Uso da Figura 28.

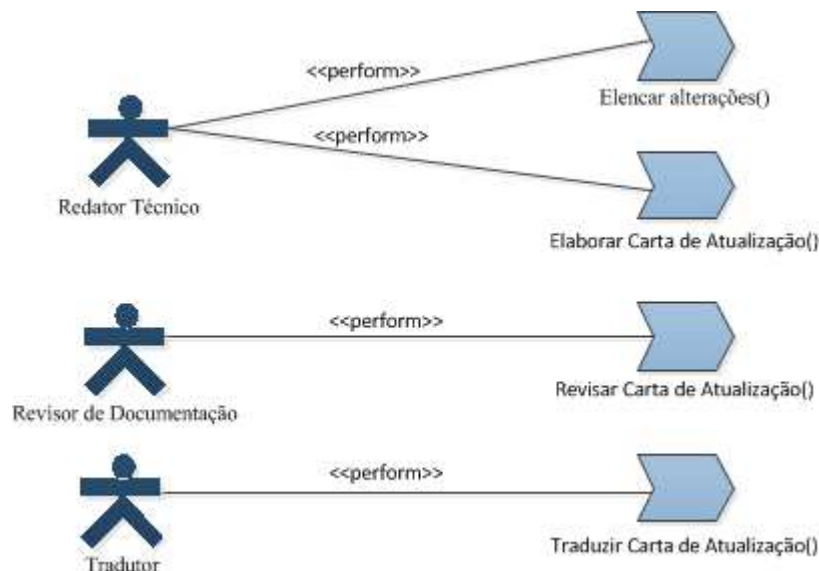


Figura 28 - Diagrama de Casos de Uso: Gerenciamento de Documentação para os Clientes

Esta disciplina gera apenas um artefato, a Carta de Atualização. Além de todas as alterações realizadas no sistema, ela também deve constar as novas especificações (quando houver) e toda informação necessária para que a atualização do *release* ocorra da maneira desejada (por

exemplo, mudanças necessárias de hardware e software). O *template* da Carta de Atualização pode ser visualizado no Apêndice K.

O Diagrama de Classes da Figura 29 ilustra o relacionamento dos papéis e artefatos envolvidos, definindo assim, os responsáveis pela geração dos mesmos.

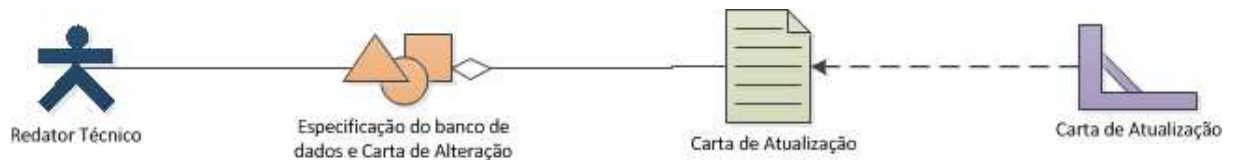


Figura 29 - Diagrama de Classes: Gerenciamento de Documentação para os Clientes

O Diagrama de Atividades da Figura 30 representa o fluxo das atividades, mostrando a precedência das mesmas.

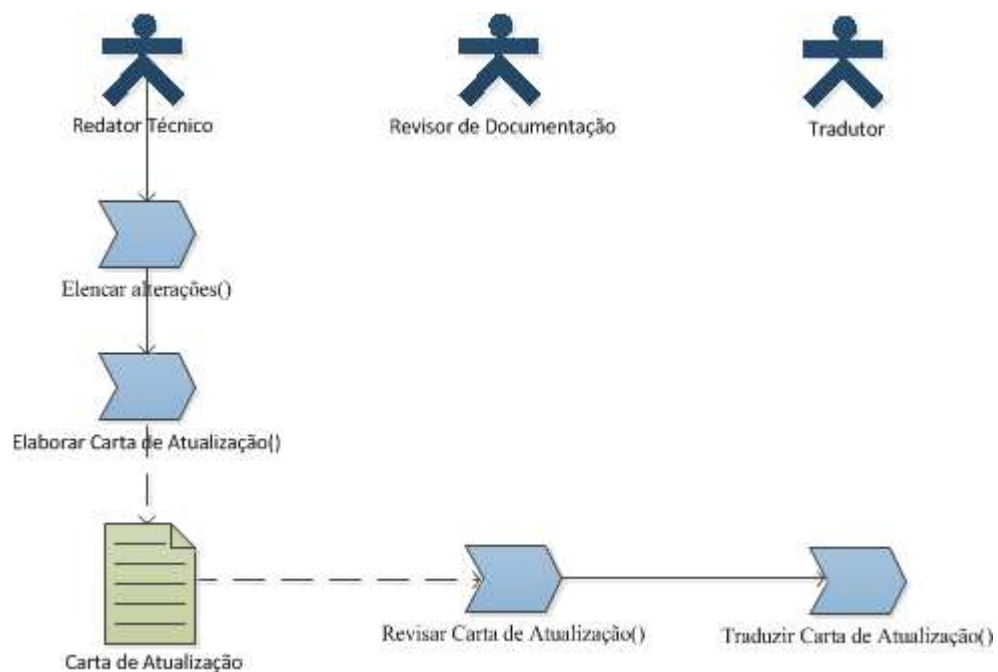


Figura 30 - Diagrama de Atividades: Gerenciamento de Documentação para os Clientes

3.4.2.8 Teste

Antigamente, entregar um sistema com pequenos *bugs* era aceitável. Na verdade, era raro encontrar algum que não tivesse erro algum. Contudo, hoje em dia a realidade é bem diferente. A disseminação da internet propiciou vários campos e especificidades no

desenvolvimento de sistemas, de forma que mesmo aqueles iniciantes no assunto, com a ajuda de apostilas e sites especializados, conseguem criar sistemas relativamente complexos e de qualidade.

A qualidade deixou de ser um diferencial para ser uma característica fundamental dos sistemas. Aliado a isto, recursos complexos surgem a cada dia, fazendo com que o desenvolvimento de software se torne uma tarefa cada vez mais complexa. Quando o nível de complexidade dos sistemas aumenta, a possibilidade do surgimento de erros também aumenta. Por isso, antes que um sistema seja entregue ao cliente, ele deve ser testado ante todas as frentes possíveis de erros e inconsistências.

A disciplina de Teste tem o objetivo de propiciar ambientes úteis de teste para a verificação e análise de todos os itens de uma aplicação.

A Disciplina Teste é detalhada no Diagrama de Pacotes representado na Figura 31, na qual é possível visualizar os elementos (papéis, atividades e artefatos) que a constituem.

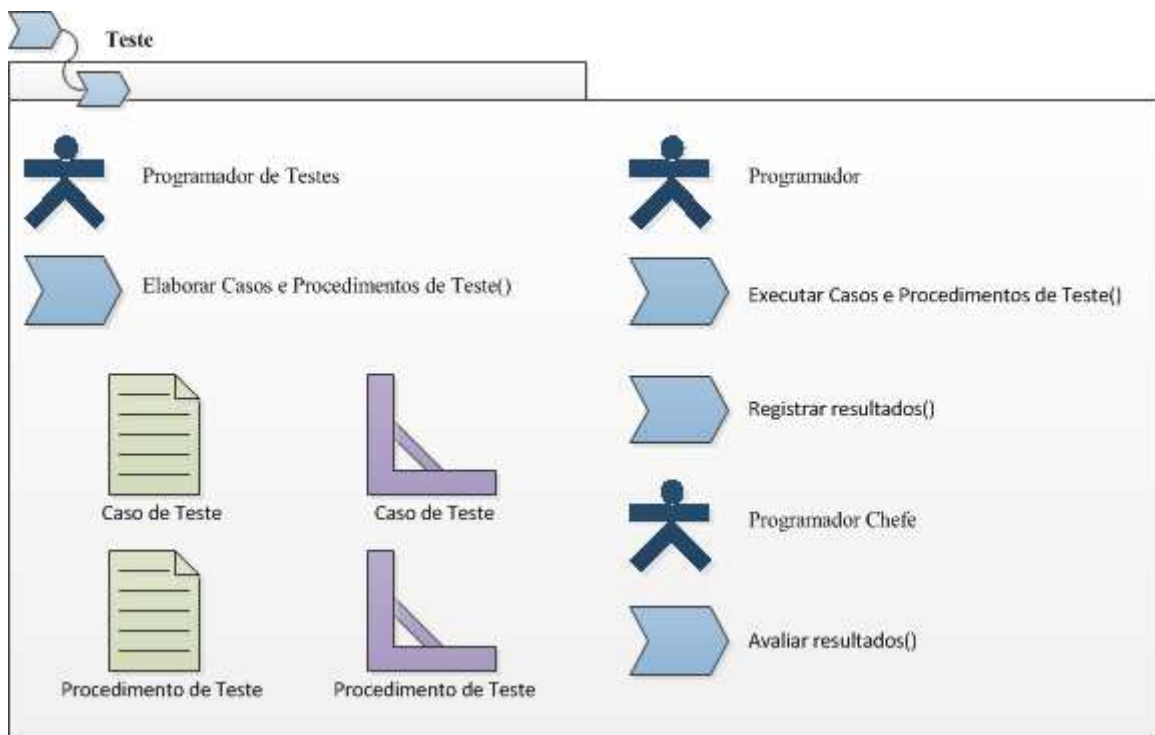


Figura 31 - Diagrama de Pacote: Teste

A Disciplina Teste é constituída pelas seguintes atividades:

- Elaborar Casos e Procedimentos de Teste;
- Executar Casos e Procedimentos de Teste. Os testes devem ser realizados durante e após a implementação do código-fonte. Quando houver um protótipo os testes já podem ser iniciados;
- Registrar resultados dos testes;
- Avaliar resultados dos testes. Quando algum teste identificar um problema a ser corrigido, a solução deve ser implementada e, depois, os testes devem ser refeitos, constituindo um ciclo;

Nesta disciplina há três papéis. O Programador de Testes é o responsável pela criação dos Casos e Procedimentos de Teste. O Programador é o responsável por executar os testes. O Programador Chefe é o responsável por avaliar a execução dos testes.

O relacionamento entre as atividades e os papéis é ilustrado no Diagrama de Casos de Uso da Figura 32.

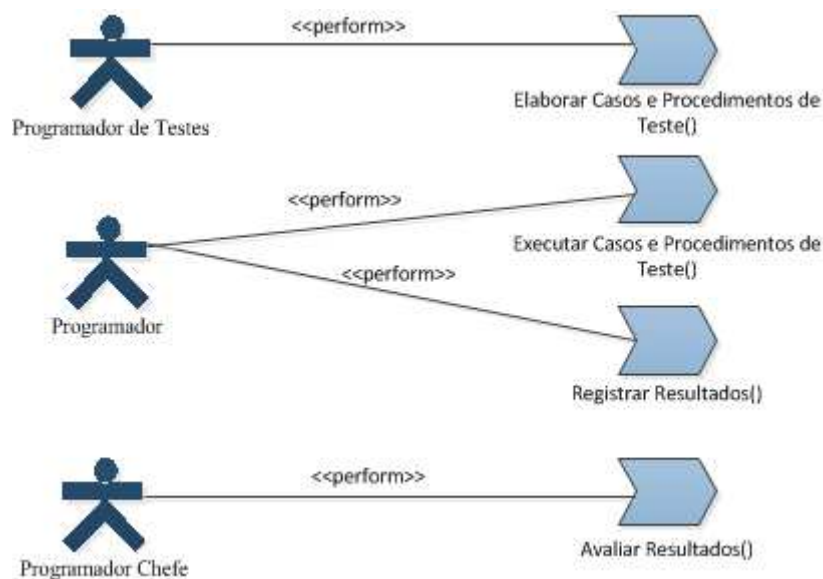


Figura 32 - Diagrama de Casos de Uso: Teste

Dois artefatos são gerados nesta disciplina. O primeiro deles é Caso de Teste, que consiste num documento contendo os principais itens dos testes (o que será testado, quais as dependências, etc.). O segundo artefato é o Procedimento de Teste, que consiste num roteiro de execução do teste. Cada Caso e Procedimento de Teste deve ser nominal para cada

Programador. Estes, por sua vez, devem registrar os resultados dos testes. Os *templates* dos dois artefatos podem ser visualizados no Apêndice L.

O Diagrama de Classes da Figura 33 ilustra o relacionamento dos papéis e artefatos envolvidos, definindo assim, os responsáveis pela geração dos mesmos.

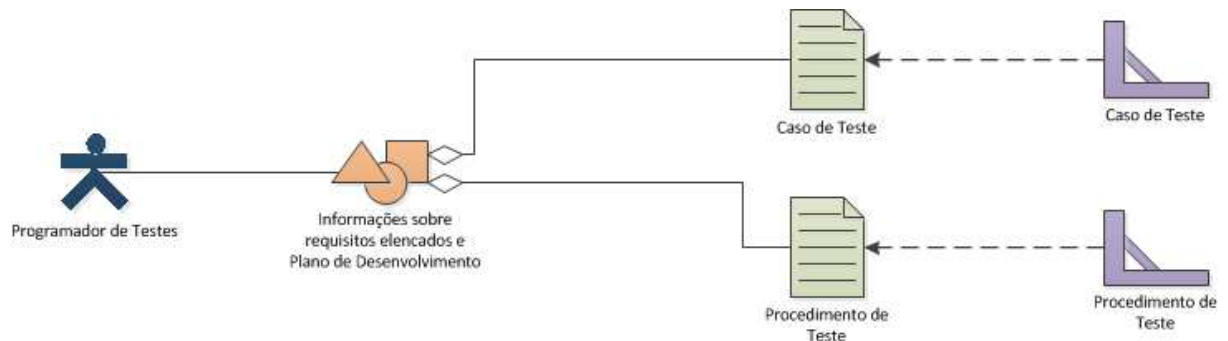


Figura 33 - Diagrama de Classes: Teste

O Diagrama de Atividades da Figura 34 representa o fluxo das atividades, mostrando a precedência das mesmas.

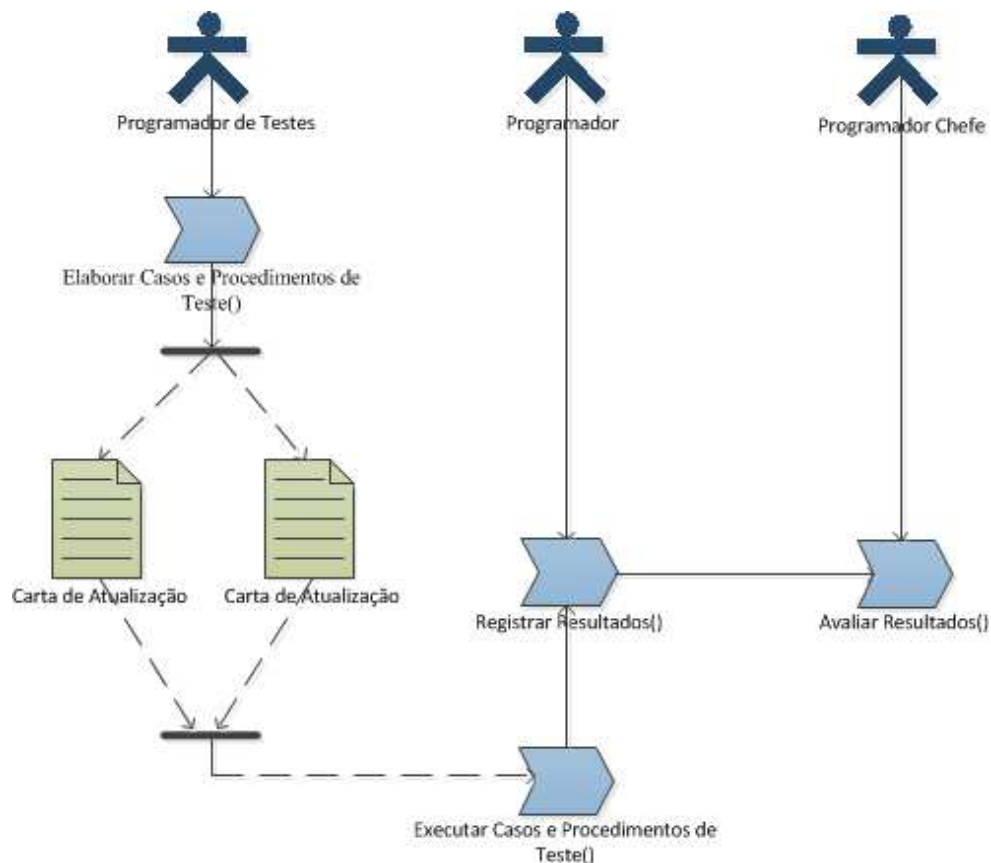


Figura 34 - Diagrama de Atividades: Teste

3.4.2.9 Ambiente

Grandes projetos possuem o que é chamado de **projeto piloto**, que consiste na aplicação de um novo projeto em um ambiente para estudo e análise. Ele é importante porque na maioria das vezes o ambiente de desenvolvimento não possui todos os recursos e limitadores que o ambiente de produção terá.

Esta disciplina tem como objetivo avaliar o *release* finalizado ante um ambiente real de produção. Embora tenha o mesmo nome de uma disciplina do RUP, seu conceito é completamente diferente. De uma certa maneira, ela pode ser vista como o último e mais importante teste. Também é importante salientar que as atividades desta disciplina são específicas para cada empresa.

A Disciplina Ambiente é detalhada no Diagrama de Pacotes representado na Figura 35, na qual é possível visualizar os elementos (papéis, atividades e artefatos) que a constituem.

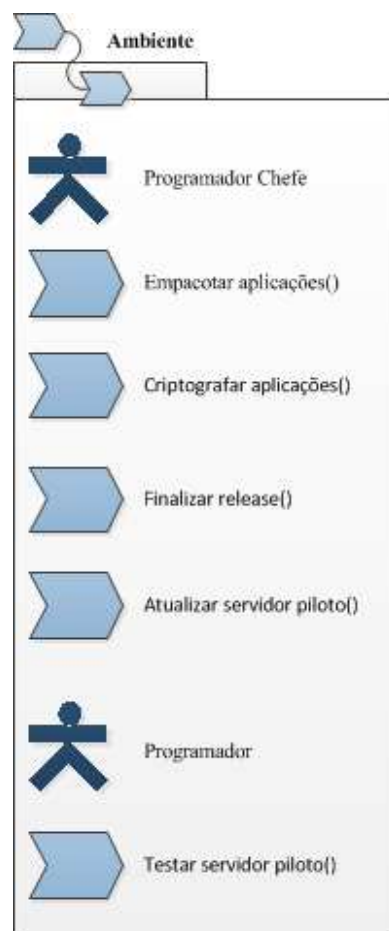


Figura 35 - Diagrama de Pacotes: Ambiente

A Disciplina Ambiente é constituída pelas seguintes atividades

- Empacotar aplicações. Esta atividade consiste em agrupar todos os módulos do *release* em um mesmo local;
- Criptografar aplicações, através de um software específico para este fim. A criptografia protege a empresa contra comercialização não autorizada do sistema;
- Finalizar todos os arquivos do *release*;
- Atualizar servidor piloto, de maneira análoga à que será feita nos clientes;
- Testar servidor piloto. É importante lembrar que qualquer problema identificado nesta fase é complexo, uma vez que todas as aplicações já foram testadas e finalizadas. Erros identificados aqui, geralmente comprometem muito o prazo estimado anteriormente;

Nesta disciplina há dois papéis. O Programador Chefe é o responsável pela preparação e atualização piloto do *release*. O Programador é o responsável por testar o servidor já atualizado, em busca de erros.

O relacionamento entre as atividades e os papéis é ilustrado no Diagrama de Casos de Uso da Figura 36.

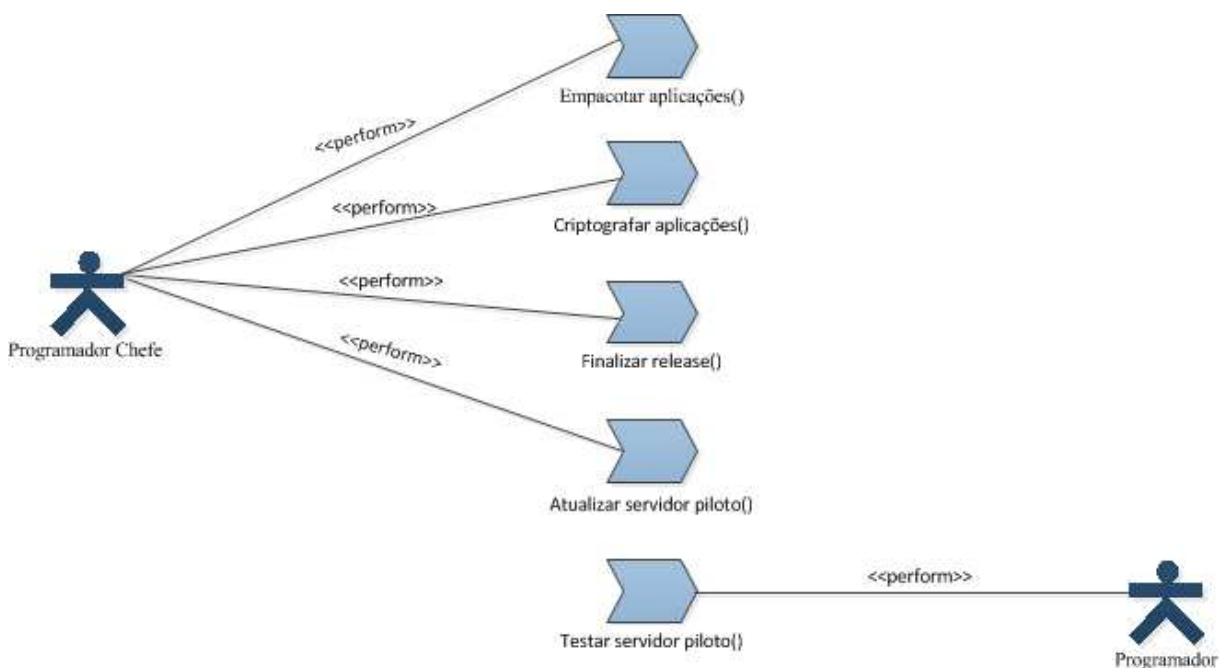


Figura 36 - Diagrama de Casos de Uso: Ambiente

Nesta disciplina não é gerado nenhum artefato.

O Diagrama de Atividades da Figura 37 representa o fluxo das atividades, mostrando a precedência das mesmas.

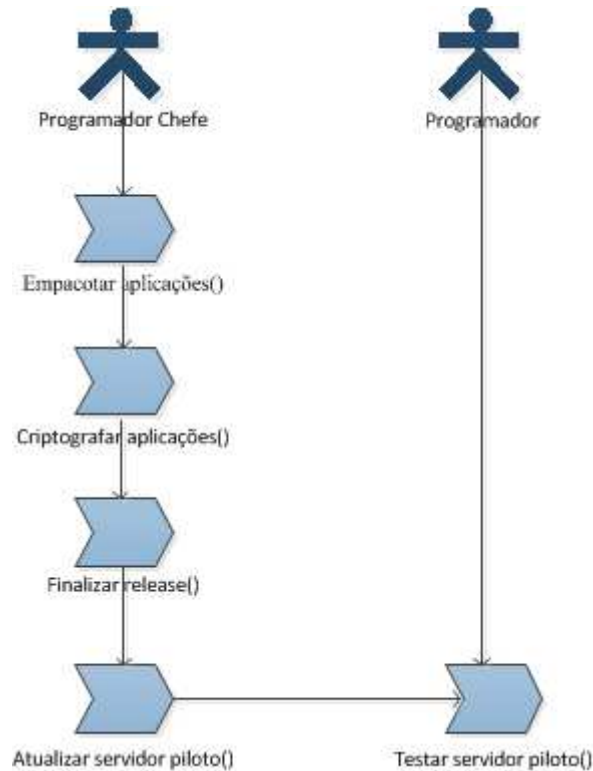


Figura 37 - Diagrama de Atividades: Ambiente

3.4.3 Relação entre as fases do RUP e as disciplinas do modelo proposto

As disciplinas do modelo proposto se adequam às fases do RUP (Início, Elaboração, Construção e Transição), conforme a Figura 15.

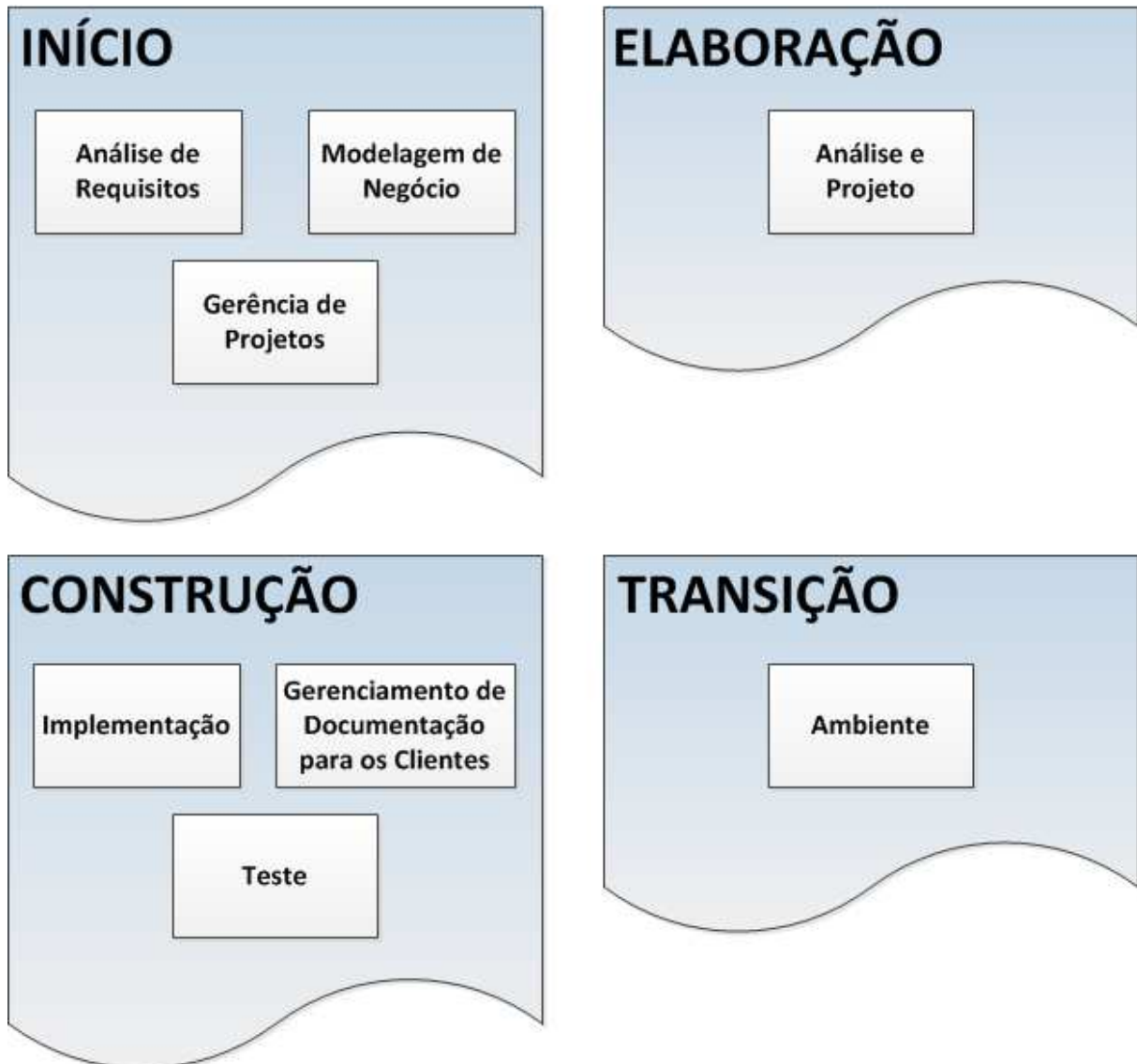


Figura 38 - Fases do RUP e Disciplinas do Modelo Proposto

A fase de Início tem por objetivo delinear o escopo do desenvolvimento e por este motivo, as disciplinas que a compõem se referem à modelagem, gerenciamento e análise. Na fase de Elaboração, como o próprio nome diz, as principais estruturas do sistema são elaboradas e por isso, a disciplina de Análise e Projeto se encontra nesta fase. A fase de Construção é onde o sistema é implementado. Ela contém as disciplinas de Implementação, Gerenciamento de Documentação para os Clientes e Teste. A fase de Transição tem por objetivo a transição do ambiente de desenvolvimento para o ambiente de produção. Ela possui a disciplina de Ambiente.

4. CONSIDERAÇÕES FINAIS

4.1 Contribuições

A formalização e documentação de um processo é o primeiro passo na implantação de um modelo de melhoria de software, seja o CMMI ou o MPS.BR. Assim, este trabalho pode contribuir para que a empresa do estudo esteja mais próxima de implantar um destes modelos.

Lembrando que a qualidade de um software está diretamente ligada ao seu processo de desenvolvimento, é importante que este tenha sido criado de forma a maximizar a utilização de todos os recursos envolvidos e, ainda, permitir que o desenvolvimento se dê com a maior eficiência possível. A padronização e sistematização das atividades contribuem significativamente para isto.

Ainda, a formalização de um modelo permite que sejam identificados gargalos no processo, etapas que podem ser realizadas em paralelo – diminuindo, assim, o tempo de desenvolvimento –, requisitos necessários para cada um que realizará cada atividade e, principalmente, a identificação mais fácil de erros que poderão ser corrigidos sem comprometer o planejamento realizado.

Por fim, formalizar um processo permite que todas as etapas do desenvolvimento sejam devidamente documentadas. Isto colabora facilitando as atividades de manutenção e adaptação do software bem como a integração de novos membros à equipe.

Considerando que as micro, pequenas e médias empresas somam a maior parte entre todas as empresas que desenvolvem software no país, é importante que elas tenham um processo bem formalizado e documentado contribuindo, assim, para uma melhoria de seu produto final. Neste âmbito, este trabalho pode contribuir para a redução de erros e atrasos no processo de desenvolvimento da iSUPER, melhorando a qualidade de seu software e aumentando a eficiência de toda equipe.

4.2 Dificuldades e Limitação

A primeira dificuldade encontrada na realização deste trabalho foi o tempo disponível, através do qual não foi possível avaliar a eficiência do modelo proposto, em campo. A aplicação deste modelo dependerá da disponibilidade dos gestores da empresa e, de certa forma, de uma reestruturação em toda Equipe de Desenvolvimento, atividade esta que deve ser realizada com a devida cautela.

Além disso, os projetos em andamento não podem simplesmente ser paralisados para implantação do modelo. Desta maneira, é necessário que ele seja implantado paralelamente às atividades que já estão sendo desenvolvidas. Assim, é necessária a execução do processo em um projeto piloto.

4.3 Trabalhos Futuros

A maturidade das empresas de desenvolvimento de software é avaliada por meio de programas, tais como o MPS-BR e o CMMI. Futuramente, a empresa poderá implantar um programa de melhoria como os citados acima, melhorando ainda mais o seu processo de desenvolvimento.

Além disto, também podem ser realizados estudos a respeito da eficiência da implantação do modelo, através de uma comparação entre antes e depois. Alguns parâmetros que podem ser comparados são:

- Tempo médio de lançamento dos *releases*;
- Número de erros do *release* anterior, corrigidos no *release* atual;

Ainda, é possível utilizar indicadores de desempenho de forma a analisar o desempenho de cada membro da equipe, principalmente através dos documentos que registram os erros e alterações realizadas por cada membro. Alguns indicadores que podem ser utilizados são:

- Produtividade, relacionando o tamanho do *release* com o tempo necessário para desenvolvê-lo;
- Qualidade, relacionando a quantidade de defeitos com o tamanho do *release*;

REFERÊNCIAS

ABREU, C.B.B. **Método para aplicação de modelos de melhoria de avaliação do processo de desenvolvimento de software em sistemas críticos de segurança.** 2008. 167f. Dissertação (Mestrado em Engenharia) – Escola Politécnica da Universidade de São Paulo, São Paulo, 2008. Disponível em: <http://www.teses.usp.br/teses/disponiveis/3/3141/tde-05112008-112625/publico/Dissertacao_rev_Becker.pdf>. Acesso em 21 mar. 2011.

AGOSTINETTO, J.S. **Sistematização do processo de desenvolvimento de produtos, melhoria contínua e desempenho: o caso de uma empresa de autopeças.** 2006. 121f. Dissertação (Mestrado em Engenharia de Produção) – Escola de Engenharia de São Carlos da Universidade de São Paulo, São Carlos, 2006. Disponível em: <http://www.teses.usp.br/teses/disponiveis/18/18140/tde-23042007-091901/publico/Juliana_Silva_Agostinetti-VF.pdf>. Acesso em 21 mar. 2011.

ARRUDA, S. **ISO/IEC 12207 Processos Fundamentais.** Publicado em 27 nov. 2006. Disponível em: <<http://www.jera.com.br/artigos/isoiec-12207-processos-fundamentais>>. Acesso em: 11 mai. 2011.

BASSI FILHO, D. L. **Experiências com desenvolvimento ágil.** 2008. 170f. Dissertação (Mestrado em Ciências) – Instituto de Matemática e Estatística da Universidade de São Paulo, São Paulo, 2008. Disponível em: <http://www.teses.usp.br/teses/disponiveis/45/45134/tde-06072008-203515/publico/Dissertacao_Metodos_Ageis_Dairton_Bassi.pdf>. Acesso em 06 abr. 2011.

BERNARDO, E. **CMMI e XP: em software, só programação não basta.** Publicado em 01 fev. 2010. Disponível em: <<http://webinsider.uol.com.br/2010/02/01/software-so-programacao-nao-basta>>. Acesso em: 11 mai. 2011.

BROERING, E. **RUP - Rational Unified Process.** Disponível em: <<http://www.devmedia.com.br/articles/viewcomp.asp?comp=4574>>. Acesso em: 05 abr. 2011.

BUENO, P.R. **Contribuição para um modelo de processo de software para pequenos grupos de desenvolvimento.** 2008. 128f. Dissertação (Mestrado em Informática Industrial) – Universidade Tecnológica Federal do Paraná, Curitiba, 2008. Disponível em: <http://arquivos.cpgei.ct.utfpr.edu.br/Ano_2008/dissertacoes/Dissertacao_481_2008.pdf>. Acesso em 21 mar. 2011.

CHAVES, A.P.; LEAL, G.C.L.; FERRANTI, J.S.; POZZA, R.S. EPIMA: Uma abordagem para desenvolvimento de sistemas acadêmicos. In: Conferência Latino americana de Informática, XXXVI, 2010. **Anais...** Assunção, Paraguai: 2010. Disponível em: <<http://www.anachaves.pro.br/arquivos/2010clei.pdf>>. Acesso em 07 mar. 2011.

CTGI, CMMI: CMMI (CAPABILITY MATURITY MODEL INTEGRATION). Disponível em: <<http://www.cits.br/cmmi.do>>. Acesso em: 11 mai. 2011.

DELMAS, C.; PENAFORTE, D.; GONÇALVES, G.; PAULO, H.; RODRIGUES, T. **RUPinho - Processos de desenvolvimento de software focado em pequenas empresas**. 2007. Centro de Informática, Universidade Federal de Pernambuco, Recife, 2007.

GONÇALES, M.A.D.; SANTORO, F.M.; MONTE, L.C.M. **Modelando Interfaces de Usuários Utilizando um Processo Ágil**. 003/2010 Rio de Janeiro: Universidade Federal Do Estado Do Rio De Janeiro, 2010. 12 p. Editor: Prof. Sean W. M. Siqueira. Disponível em: <<http://www.seer.unirio.br/index.php/monografiasppgi/article/viewFile/948/722> >. Acesso em: 11 mai. 2011.

HILGERT, F.P.; MOREIRA, L.S.R.; PRIKLADNICKI, R.; BOSSLE, R.; MÓRA, M.C.; BACK, R. Gestão de Mudança em Melhoria de Processo de Software: Um Relato de Integração entre RH e SEPG na Tlantic SI. In: Workshop Um Olhar Sociotécnico sobre a Engenharia de Software, IV, 2008. **Anais...** Florianópolis, 2008. Disponível em: <<http://www.cos.ufrj.br/%7Ehandrade/woses/woses2008/?download=027.pdf>>. Acesso em 21 mar. 2011.

LEAL, G.C.L. **Uma abordagem integrada de desenvolvimento e teste de software para equipes distribuídas**. 2010. 169f. Dissertação (Mestrado em Ciência da Computação) – Universidade Estadual de Maringá, Maringá, 2010. Disponível em <<http://www.din.uem.br/pos-graduacao/mestrado-em-ciencia-da-computacao/arquivos/dissertacoes-1/Gislaine%20Camila%20Lapasini%20Leal.pdf>>. Aceso em 07 mar. 2011.

LIEBMAN, A. **Melhoria no Processo de Software: Implantação do MPS.BR Nível G em uma Empresa de Pequeno Porte**. 2006. 56f. Monografia (Bacharelado em Ciência da Computação) – Departamento de Ciência da Computação da Universidade Federal de Lavras, Lavras, 2006. Disponível em: <http://www.bcc.ufla.br/monografias/2005/Melhoria_no_processo_de_software_implantacao_do_MPS_BR_nivel_G_em_uma_empresa_de_pequeno_porte.pdf>. Acesso em 20 abr. 2011.

LUIZ, R.R.V. **Obtendo Qualidade de Software com o RUP**. Publicado em 17 ago. 2009. Disponível em: <<http://javafree.uol.com.br/artigo/871455/Obtendo-Qualidade-de-Software-com-o-RUP.html>>. Acesso em: 05 abr. 2011.

MALHEIROS, V. **Uma contribuição para a melhoria colaborativa e distribuída de processos de software**. 2010. 218f. Tese (Doutorado em Ciências) – Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo, São Carlos, 2010. Disponível em: <http://www.teses.usp.br/teses/disponiveis/55/55134/tde-18082010-143052/publico/Tese_Viviane_Dias_Malheiros_de_Pinho.pdf>. Acesso em 21 mar. 2011.

MARTINEZ, M. **RUP**. Publicado em 19 mar. 2010. Disponível em: <<http://www.infoescola.com/engenharia-de-software/rup>>. Acesso em: 05 abr. 2011.

MIRANDA, J.M. **Analisis de Estructuras para la Definición de Procesos**. Universidad Politécnica de Madrid, Faculdade de Informática, 2006. **Trabalho de Pesquisa Tutelado**. Disponível em: <http://www.dlsiis.fi.upm.es/docto_lsiis/Trabajos20062007/Mejia.pdf>. Acesso em: 11 mai. 2011.

MOURA, E.C. **Melhoria organizacional e mentalidade empresarial**. Disponível em: <<http://www.qualiplus.com.br/blog-q/artigos/91-melhoria-organizacional-e-mentalidade-empresarial.html>>. Acesso em: 12 mai. 2011.

NASCIMENTO, H.A.A. **Melhoria do Processo de Software e a Avaliação da Maturidade no Modelo MPS.BR em uma Pequena Empresa**. 2008. 61f. Trabalho de Conclusão de Curso (Graduação em Ciência da Computação) – Universidade Estadual de Londrina, Londrina, 2008.

NOGUEIRA, M. **Qual a Importância da Adoção da Norma ISO 12207 nas Empresas de Desenvolvimento de Software?** In: Simpósio de Engenharia de Produção, X, 2003. **Anais...** Bauru: 2003. Disponível em: <http://www.simpep.feb.unesp.br/anais_simpep_aux.php?e=10>. Acesso em 20 abr. 2011.

OPRIME, P.C.; DONADONE, J.C.; MONSANTO, R.R. Estudo da operacionalização do processo de melhoria contínua na abordagem do TPM: um estudo de campo das empresas brasileiras. In: Encontro Nacional de Engenharia de Produção, XXIX, 2009. **Anais...** Salvador: 2009. Disponível em: <http://www.abepro.org.br/biblioteca/enegep2009_TN_STO_092_625_12874.pdf>. Acesso em 21 mar. 2011.

PARREIRAS, F.S.; OLIVEIRA, G.S. Análise comparativa de processos de desenvolvimento de software sob a luz da gestão do conhecimento: um estudo de caso de empresas mineiras. In: Simpósio Brasileiro de Qualidade de Software, III, 2004. **Anais...** Brasília: 2004. Disponível em: <<http://bibliotecadigital.sbc.org.br/download.php?paper=267>>. Acesso em 21 mar. 2011.

PAZ, S. **Os Desafios da Melhoria Contínua nas Pequenas Empresas Prestadoras de Serviços**. Disponível em: <http://www.ibco.org.br/index.php?option=com_content&view=article&id=665&catid=15:artigos-publicados&Itemid=644>. Acesso em: 12 mai. 2011.

RODRIGUES, K.F.C.; FARIA, A.F.; SUZUKI, J.A. Implantação da gestão por processos em uma empresa nascente de base tecnológica. In: Simpósio de Engenharia de Produção, XVII, 2010. **Anais...** Bauru: 2010. Disponível em: <http://www.simpep.feb.unesp.br/anais_simpep.php?e=5>. Acesso em 21 mar. 2011.

SANTOS, W.C.; TAVARES, E.; VIEIRA, O. **Processo de Desenvolvimento de Software: AUP – Agile Unified Process**. In: Seminário sobre Processo de desenvolvimento de Software, 2007, Rio Grande do Norte. Disponível em:

<<http://opentechnology.files.wordpress.com/2007/10/aup-agile-unified-process.pdf>>. Acesso em 09 mai. 2011.

SATO, D. T. **Uso eficaz de métricas em métodos ágeis de desenvolvimento de software**. 2007. 155f. Dissertação (Mestrado em Ciências) – Instituto de Matemática e Estatística da Universidade de São Paulo, São Paulo, 2007. Disponível em: <<http://www.teses.usp.br/teses/disponiveis/45/45134/tde-06092007-225914/publico/dissertacao.pdf>>. Acesso em: 06 abr. 2011.

SCHNEIDER, G. **CMMI: do desenvolvimento à gestão de projetos**. Publicado em 03 mar. 2009. Disponível em: <<http://webinsider.uol.com.br/2009/03/03/cmmi-do-desenvolvimento-a-gestao-de-projetos>>. Acesso em: 11 mai. 2011.

SILVA, A.F. **Reflexões sobre o ensino de metodologias ágeis na academia, na indústria e no governo**. 2007. 151f. Dissertação (Mestrado em Ciências) – Instituto de Matemática e Estatística da Universidade de São Paulo, São Paulo, 2007. Disponível em: <<http://www.teses.usp.br/teses/disponiveis/45/45134/tde-17122007-175223/publico/dissertacaoAleFreirelowfiPraWeb.pdf>>. Acesso em 06 abr. 2011.

SILVA, E.L.; MENEZES, E.M. **Metodologia da Pesquisa e Elaboração de Dissertação**. Florianópolis, 2005. 139f. Disponível em: <<http://moodlep.uem.br/mod/resource/view.php?id=2395>>. Acesso em 21 mar. 2011.

SILVA, F.L.C.; SALES, D.S.; CARVALHO, R.A. Desenvolvimento e manutenção de software ágil: Um novo processo. In: Simpósio de Engenharia de Produção, XVII, 2009. **Anais...** Bauru: 2009. Disponível em: <http://www.simpep.feb.unesp.br/anais_simpep.php?e=4>. Acesso em 21 mar. 2011.

SOARES, J.M. **Melhoria contínua da qualidade do produto e do processo industrial de uma empresa de confecção**. 2010. 98f. Trabalho de Conclusão de Curso (Graduação em Engenharia de Produção) – Universidade Estadual de Maringá, Maringá, 2010.

SOMMERVILLE, Ian. **Engenharia de Software**. 6. ed. São Paulo: Addison Wesley, 2003. 585 p. Tradução de: André Maurício de Andrade.

SOUZA, W. **O CMMI parece ser focado em Gerentes de Projetos. Tem algo para desenvolvedores?** Publicado em 21 mar. 2011. Disponível em: <<http://www.blogcmmi.com.br/engenharia/o-cmmi-parece-ser-focado-em-gerentes-de-projetos-tem-algo-para-desenvolvedores>>. Acesso em: 11 mai. 2011.

TAMAKI, P.A.O. **Uma Extensão do RUP com Ênfase no Gerenciamento de Projetos do PMBOK Baseada em *Process Patterns***. 2008. 211f. Dissertação (Mestrado em Engenharia) – Escola Politécnica da Universidade de São Paulo, São Paulo, 2007. Disponível em: <<http://www.teses.usp.br/teses/disponiveis/3/3141/tde-08012008-113326/publico/Dissertacao.pdf>>. Acesso em 06 abr. 2011.

UEDA, R.M.; NOGUEIRA, E. Melhoria contínua em uma empresa de alimentos. In: Simpósio de Engenharia de Produção, XVII, 2010. **Anais...** Bauru: 2010. Disponível em: <http://www.simpep.feb.unesp.br/anais_simpep.php?e=5>. Acesso em 21 mar. 2011.

CTGI, UMA VISÃO Geral do CMMI. Publicado em 06/2004. Disponível em: <<http://www.ctgi.com.br/rep/cmmi.pdf>>. Acesso em: 11 mai. 2011.

VALADAO, A.F.C.; TURRIONI, J.B. A influência da cultura organizacional no processo de melhoria contínua. In: Simpósio de Engenharia de Produção, XVII, 2010. **Anais...** Bauru: 2010. Disponível em: <http://www.simpep.feb.unesp.br/anais_simpep.php?e=5>. Acesso em 21 mar. 2011.

VASCO, C.G.; Vithoft, M.H.; Estante, P.R.C. **Comparação entre Metodologias RUP e XP**. 2008. Programa de Pós Graduação em Informática Aplicada (Dissertação), Pontifícia Universidade Católica do Paraná, Curitiba. Disponível em: <http://www.edilms.eti.br/uploads/file/bd/RUPvsXP_draft.pdf>. Acesso em 04 abr. 2011.

VIGGIANO, E. **Arquitetura Ágil com o AUP**. Publicado em 20 nov. 2008. Disponível em: <<http://blog.arkhi.com.br/2008/11/20/arquitetura-agil-com-o-aup>>. Acesso em: 11 mai. 2011.

WEBER, K. *et al.* **Melhoria de Processo do Software Brasileiro (MPS.BR): um Programa Mobilizador**. In: Latin-American Conference On Informatics, XXXII, 2006. Anais... Santiago, Chile: 2006. Disponível em: <<http://golden.softex.br/portal/softexweb/uploadDocuments/26.pdf>>. Acesso em 20 abr. 2011.

APÊNDICE





APÊNDICE A – *TEMPLATE*: MODELO DE CRONOGRAMA DO PROJETO

ATIVIDADES MACRO





	Nome *	Início *	Fim *	Prioridade *	Tempo Estimado *	Tempo Gasto	Progresso	Usuário	Situação *
	Modelagem Banco de Dados	21/06/2011	21/06/2011	5	04:00	03:00	100,00 %	anderson	Concluído
	Criação da Cadastro	21/06/2011	22/06/2011	4	13:00	11:00	100,00 %	anderson	Concluído
	Criação da Consulta	27/06/2011	27/06/2011	4	08:30	14:00	100,00 %	anderson	Concluído
	Criação dos Menus	28/06/2011	28/06/2011	4	08:30	08:30	100,00 %	anderson	Concluído
	Alteração formata avisos	29/06/2011	29/06/2011	3	04:30	04:00	100,00 %	anderson	Concluído
	Criação rotina de impressão	29/06/2011	29/06/2011	3	04:00	02:00	100,00 %	anderson	Concluído
	Testes	30/06/2011	01/07/2011	2	17:00	03:00	100,00 %	anderson	Concluído
	Adição filtro aniversariantes	13/07/2011	13/07/2011	1	08:30	08:30	100,00 %	anderson	Concluído
	Alteração - Central de Avisos	19/07/2011	19/07/2011	1	02:00	03:00	100,00 %	anderson	Concluído
	Alteração - Edição Expressa	25/07/2011	26/07/2011	1	12:00	12:00	50,00 %	anderson	Concluído

DETALHAMENTO DE ATIVIDADES

Cadastro de Atividades

Data	Tempo Gasto	Descrição *
 25/07/2011 11:58:46	01:30	-Criados novos campos e alteradas rotinas visuais
 25/07/2011 17:34:21	04:00	-Rotina criada -Testes básicos concluídos
 25/07/2011 17:34:30	00:30	-Atualização provedores
 27/07/2011 08:10:09	06:00	-Continuação desenvolvimento google maps

APÊNDICE B – TEMPLATE: PLANO DE DESENVOLVIMENTO

PLANO DE DESENVOLVIMENTO		
Legenda		Tarefas a serem realizadas na virada de versão
		Tarefas que dependem do Analista de Requisitos
		Tarefas constantes durante todo o desenvolvimento
		Tarefas em segundo plano
		Demais tarefas
Descrição da Tarefa		Atual
Abrir ticket de atualização		
Opção de consultar notas no cadastro de clientes		
Inclusão de baixa automática via código de barras		
Alteração do campo vlan da tabela do banco de dados		
Atualização do changelog		X
Conferência identificador de débito automático – Banco do Brasil		
Trocar comando “echo” por macro específica na página de login		
Incluir rotinas de alterações no cadastro dos clientes		X
Incluir hints de ajuda		

APÊNDICE C – *TEMPLATE*: CASO DE USO ESTENDIDO

Nome do Caso de Uso	
Autor/Equipe	
Data	
Atores	
Descrição	
Pré-condições	
Pós-condições	
Fluxos Alternativos	

APÊNDICE D – TEMPLATE: TABELA DO BANCO DE DADOS NO EXCEL


Nome da Tabela		
Campo	Tipo	Descrição
Codigo	BIGINT(15)	Código único do cliente
Nome	VARCHAR(100)	Nome do cliente
Endereco	VARCHAR(200)	Endereço do cliente com o número
Bairro	VARCHAR(30)	Bairro do cliente
Cidade	VARCHAR(30)	Cidade do cliente
UF	CHAR(2)	UF do cliente
CEP	VARCHAR(8)	CEP do cliente (apenas números)
CPF	VARCHAR(11)	CPF do cliente (apenas números)
CNPJ	VARCHAR(14)	CNPJ do cliente (apenas números)
Sexo	CHAR(1)	Sexo do Cliente: M=Masculino/F=Feminino
Nascimento	DATE	Data de Nascimento do cliente (aaaa-mm-dd)
Inclusão	DATE	Data de Inclusão do cadastro (aaaa-mm-dd)
Tipo	INT(5)	Tabela ClienteTipo
DiaCobranca	VARCHAR(2)	Dia de Vencimento das duplicatas
BancoCobranca	INT(4)	Banco de Cobrança do cliente
TipoCobranca	CHAR(1)	Tipo Cobrança: S=Simple/R=Registrada
Assinante	CHAR(1)	Assinante de planos: S=Sim/N=Não

Índices

Primário
Secundário

Código (AUTOINCREMENT)
CPF

APÊNDICE E – TEMPLATE: RASCUNHO DE INTERFACE



**FÓRUM
FUTURO 10
PARANÁ**

RADIO BUTTON

EDIÇÃO EXPRESSA

TIPO DE ALTERAÇÃO: CLIENTES FINANCEIRO FISCAL


OPÇÕES CLIENTES

ALTERAÇÕES	VENCIMENTO
_____	_____
_____	_____
_____	_____
_____	_____

BANCO

OK

Realização:






- FOCSEP
- FOCORRÊNDO
- EPB
- GASEPAN
- ADP

Promoção:


GAZETA DO POVO

Respeito por você.

Apóio:

Apóio Gráfico:



APÊNDICE F – TEMPLATE: RASCUNHO DE APLICAÇÃOAUDITORIA DO SISTEMA

-> LAÇO PRINCIPAL

-> RECUPERAR TORRES

• LAÇO SECUNDÁRIO {TORRES}

{

REGISTRA TORRE;

REGISTRA PORTAS;

REGISTRA PAINÉIS;

}

• REGISTRA VARIÁVEIS LOCAIS;

• REGISTRA VARIÁVEIS GLOBAIS;

• ATUALIZAR_CADASTRO();

-> FINALIZA LAÇO

APÊNDICE G – TEMPLATE: CARTA DE ALTERAÇÃO

Aplicação: app_faturamento [Faturamento mensal]		Versão do <i>release atual</i> : 2.0.006 Versão da <i>alteração</i> : 2.0.007
Tipo de alteração	<input checked="" type="checkbox"/> Interface	<input type="checkbox"/> BD <input checked="" type="checkbox"/> Regras de Negócio
Detalhamento: Interface		
<ul style="list-style-type: none"> • Criação do campo <i>Data de Emissão</i> de Nota Fiscal no bloco <i>Faturamento</i> e abaixo do campo <i>Dia Vencimento</i>. • O campo <i>Data de Emissão</i> não deve aceitar datas anteriores <u>ao dia de hoje</u> ou com <u>mais de 30 dias à frente</u>. • O campo <i>Data de Emissão</i> deverá aparecer sempre que o Mês/Ano for menor do que o atual. Para mostrar/esconder o campo, poderá ser utilizada a macro <code>sc_field_display</code>. 		
Detalhamento: BD		
<ul style="list-style-type: none"> • Nenhuma alteração 		
Detalhamento: Regras de Negócio		
<ul style="list-style-type: none"> • Quando o faturamento for referente a um mês anterior ao atual, o sistema utilizará o campo <i>Data de Emissão</i> para gerar as notas fiscais referentes ao faturamento. 		
Observações		

APÊNDICE H – *TEMPLATE*: SCRIPT DE BANCO DE DADOS

```

CREATE TABLE IF NOT EXISTS `NFCab` (
  `Emitente` varchar(14) NOT NULL,
  `Modelo` varchar(2) NOT NULL,
  `Tipo` int(1) NOT NULL,
  `Serie` int(3) NOT NULL,
  `Numero` int(9) NOT NULL,
  `Codigo` int(8) NOT NULL,
  `ChaveAcesso` varchar(44) NOT NULL,
  `Emissao` date NOT NULL,
  `Saida` datetime NOT NULL,
  `NatOper` varchar(60) NOT NULL,
  `TpUtilizacao` int(1) NOT NULL,
  `Municipio` int(7) NOT NULL,
  `Pagto` int(1) NOT NULL,
  `EmiTipo` int(1) NOT NULL,
  `EmiFinalidade` int(1) NOT NULL,
  `ImpProcesso` int(1) NOT NULL,
  `ImpDANFE` int(1) NOT NULL,
  `VersaoApl` varchar(20) NOT NULL,
  `ContData` datetime NOT NULL,
  `ContMotivo` varchar(256) NOT NULL,
  `CliFor` int(12) NOT NULL,
  `RetPIS` double(15,2) NOT NULL,
  `RetCOFINS` double(15,2) NOT NULL,
  `RetCSLL` double(15,2) NOT NULL,
  `RetIRRFBBaseC` double(15,2) NOT NULL,
  `RetIRRF` double(15,2) NOT NULL,
  `RetPrevBaseC` double(15,2) NOT NULL,
  `RetPrev` double(15,2) NOT NULL,
  `FreteMod` int(1) NOT NULL,
  `InfAdicFisco` text NOT NULL,
  `InfAdicContr` text NOT NULL,
  `Origem` VARCHAR( 1 ) NOT NULL DEFAULT 'F',
  `LoteNum` int(15) NOT NULL,
  `Recibo` bigint(15) NOT NULL,
  `Situacao` int(1) NOT NULL,
  `ProtAutorizacao` bigint(15) NULL,
  `CancUsuario` VARCHAR( 20 ) NULL,
  PRIMARY KEY (`ChaveAcesso`),
  KEY `Numero` (`Serie`,`Numero`),
  KEY `Codigo` (`Codigo`),
  KEY `DtEmissao` (`Emissao`),
  KEY `CliFor` (`CliFor`),
  KEY `LoteNum` (`LoteNum`),
  KEY `Recibo` (`Recibo`),
  KEY `Situacao` (`Situacao`)

```

```
) ENGINE=MyISAM DEFAULT CHARSET=latin1;
```


APÊNDICE I – TEMPLATE: CÓDIGO FONTE COM COMENTÁRIOS

```
/* Recupera informações da empresa */
$select = "SELECT * FROM Empresa LIMIT 1";
$doin = mysql_query ($select) or die (mysql_error());
if(mysql_num_rows($doin)>0){
    $vresults = mysql_fetch_array($doin);
    $vrazaosocial = $vresults['RazaoSocial'];
    $vcnpj = $vresults['CNPJ'];
}
```

APÊNDICE J – *TEMPLATE*: DOCUMENTO DE REVISÃO

Documento de Revisão		
Aplicação: app_maps_nas	Programador: Anderson	Data: 01/08/11
Obs. 1	Intervalo de linhas: 45 à 120	
<p>- O laço principal não está considerando as coordenadas dos NAS. Se um NAS estiver fora dos limites dos clientes, ele ficará fora da visualização do mapa.</p> <p>- Adicionar, após o laço principal, um outro laço checando os limites do NAS.</p>		
Obs. 2	Intervalo de linhas: à	
Obs. 3	Intervalo de linhas: à	

APÊNDICE K – *TEMPLATE*: CARTA DE ATUALIZAÇÃO

Router Box
 ===== Versão ISP

Página 1/10

Changelog 31/05/2011

Versão 2.0.006

RESUMO

- Emissão de NFe - Nota Fiscal Eletrônica (modelo 55);
- Emissão de Nota Fiscal Via Única modelo 21;
- Emissão de Nota Fiscal Via Única modelo 22;
- Pool de IPs para blocos maiores que /24;
- Agrupamento de contratos;
- Cobrança para Banco do Nordeste;
- Vencimento contra apresentação no faturamento;
- ⊕ 66 implementações em aplicações;
- 🔧 27 alterações em aplicações;
- 🔥 11 correções de erros em aplicações;

- Executáveis Sistema Operacional (daemons)
 - 🔧 Adequações de compatibilidade com Linux Kernel 2.6.32.
 - 🔥 Corrigido erro na aplicação de regras de clientes com controle de navegação por horário. Atenção: Para servidores que já utilizam Kernel 2.6.32 será necessário reiniciá-lo para o correto funcionamento desta função.
 - 🔥 Corrigido intervalo de atualização automática de horário (NTP);
 - ⊕ Adicionado aplicativo para geração automática de notas fiscais a partir de pré-notas;
- Atendimentos/Tópicos
 - ⊕ Adicionada rotina de dependência com a tabela *Empresa*, que impede a exclusão de um tópico que esteja sendo utilizado nos parâmetros da *Central do Assinante* na opção *Empresa/Central Assinante*.
- Empresa/Certificados Digitais
 - ⊕ Repositório dos certificados digitais da empresa. É possível cadastrar apenas certificados digitais do modelo A1.
- Empresa/Clientes/Cadastro/Contratos
 - ⊕ Foi adicionado o campo *Agrupar* nas opções de inclusão e edição de contratos. O objetivo deste campo é indicar se determinado contrato será agrupado com outros na geração de movimentação financeira. Durante a rotina de faturamento, para todos os contratos que estiverem com a opção *Agrupar=Sim* será gerado um único título, tal como sempre aconteceu. Para cada contrato que estiver com a opção *Agrupar=Não*

APÊNDICE L – *TEMPLATE*: CASOS DE TESTE

ESPECIFICAÇÃO DE CASO DE TESTE

Identificação do Projeto:			
Responsável:			
Requisitos do Ambiente:			
CT	Pacote	Descrição	Dependência

ESPECIFICAÇÃO DE PROCEDIMENTOS DE TESTE

Identificação do Projeto:		
Responsável:		
Requisitos do Ambiente:		
CT	Roteiro de Teste (passos)	Resultado Esperado

Universidade Estadual de Maringá
Departamento de Engenharia de Produção
Av. Colombo 5790, Maringá-PR CEP 87020-900
Tel: (044) 3011-4196/3011-5833 Fax: (044) 3011-4196