UNIVERSIDADE FUMEC

ANDRÉ COURA COSTA LOPES DE SOUZA

RAFAEL FERREIRA MONTEIRO

GESTÃO DA QUALIDADE DE SOFTWARE:
Garantia da Qualidade Total

BELO HORIZONTE

2009

ANDRÉ COURA COSTA LOPES DE SOUZA

RAFAEL FERREIRA MONTEIRO

GESTÃO DA QUALIDADE DE SOFTWARE:
Garantia da Qualidade Total

Monografia apresentado à disciplina de Trabalho de Conclusão de Curso, da Graduação de Ciência da Computação, da UNIVERSIDADE FUMEC - FACE, sob a orientação dos professores Osvaldo Manoel Corrêa e Ricardo Terra.

BELO HORIZONTE

2009

LISTA DE FIGURAS

SUMÁRIO

1 Introdução

Obter qualidade no desenvolvimento de software e nos processos relacionados a ele não é uma tarefa trivial. São vários os fatores que dificultam atingir os objetivos de qualidade. No entanto, nada mais decepcionante do que entregar um software que não satisfaça as necessidades dos clientes. Muitos recursos são gastos, mas, em muitos casos, ocorre uma grande frustração por parte dos clientes pelo não atendimento dos requisitos ou até mesmo pelo erro nos levantamentos das necessidades. Diversos dos problemas são derivados da falta de atenção para a tarefa de definir e acompanhar a evolução dos requisitos durante o processo de construção de software.

Hoje em dia, sistemas de software não são mais um simples programa com poucas funcionalidades, mas sim, sistemas complexos com varias funcionalidades interligadas e que estão em constante evolução. A manutenção do software, isto é, modificações em telas usuais, usabilidade e artefatos já existentes, chegam a consumir três quartos do custo total do seu ciclo de vida. Aproximadamente, um quinto de todo o esforço de manutenção é usado para consertar erros de implementação e o restante é utilizado na adaptação do software em função de modificações em requisitos funcionais, regras de negócios e na reengenharia da aplicação.

A princípio se precisa entender que ao se referir à qualidade e Engenharia de Software[1] está se referindo também aos produtos e processos. De nada adianta centrarmos a atenção só no produto ou só no processo, pois o que realmente se visa é a satisfação do cliente. É necessário que todas as visões caminhem juntas. Portanto, para lidar com qualidade, é necessário conhecer claramente que o processo de produção deve ter qualidade e que o produto deve incorporá-la, o objetivo na Engenharia de Software é a qualidade do produto de software.

Em síntese, para se entender qualidade falaremos sobre o que é qualidade e o que é a qualidade do software, abordando sua importância desde a fase de concepção até à finalização do projeto exemplificando alguns dos principais processos estruturados para a melhoria da qualidade e seus estados de complexidade. Para melhor abordar esse assunto exemplificar-se-à com o modelos de qualidade de software atualmente mais difundido no

[1] Uma primeira definição de Engenharia de Software foi proposta por Fritz Bauer em 1969 na primeira grande conferência dedicada ao assunto: “O estabelecimento e uso de sólidos princípios de engenharia para que se possa obter economicamente um software que seja confiável e que funcione eficientemente em máquinas reais” (PRESSMAN, 1995, p. 31).

Brasil, o modelo de Melhoria de Processo de Software Brasileiro (MPS.BR), que teve como base aprimorada o C*apability Maturity Model Integration* – Modelo Integrado de Capacitação e Maturidade (CMMI) onde também abordamos seus níveis e processos.

Será apresentada a Organização Internacional de Padronização (ISO) em seus modelos propostos para informática: ISO/IEC 9000, que é mais genérico em relação à qualidade, tratando o software como um produto; ISO/IEC 12207 relacionada ao ciclo de vida de software; ISO/IEC 15504 aos processos de desenvolvimento.

Serão abordado os processos de vida de software e sua evolução a partir de metodologias implantadas desde o desenvolvimento de software até as fases de homologação.

Será apresentado o planejamento da qualidade em si, pois o gerenciamento é diretamente relacionado com este planejamento podendo com isto prever e evitar defeitos futuros em softwares.

Serão abordados modelos de automação de testes e sua importância desde os testes de unidade padrão automatizados garantindo a robustez do código, testes de integração e a garantia da compatibilidade do software, funcionais e de regressão onde garante que não houve impactos. Será abordado a importância da melhoria contínua e sua não estagnação, pois a qualidade é sempre relativa e a busca por ela deve ser sempre aprimorada e atualizada.

No capítulo 4, é abordado o processo de implantação do Gerenciamento de Qualidade onde serão abordados os problemas anteriores à implantação, todas as dificuldades para implantar o gerenciamento de qualidade, tanto nos aspectos funcionais como nos não funcionais, rejeição por parte dos envolvidos às novidades, e posteriormente as vantagens encontradas no processo.

Com isso, será proposto um modelo utopicamente ideal para atendimento da qualidade em função do tempo.

2 Qualidade

O conceito de qualidade é relativo, para cada pessoa pode significar algo diferente, o que atende a uma pessoa, não necessariamente atende à outra.

A qualidade depende muito dos requisitos e da complexidade envolvida, por exemplo, a arquitetura de uma casa é completamente diferente da arquitetura de um prédio e isso não está apenas relacionada à quantidade de tijolos que envolvem a construção, á muito mais complexidade nisso, por exemplo, os alicerces, a preparação do terreno, quantidade de vagas, volume da caixa d'água, ou seja, há inúmeros requisitos diferenciados envolvendo essas construções.

Na informática não é diferente, a computação vem ganhando nome e espaço desde 1950 até os dias de hoje em que o uso de computador se tornou indispensável. Desde aquela época, em que os computadores eram do tamanho de salas inteiras era indispensável que os programas atendessem aos requisitos propostos pois, caso contrário, seria muito difícil automatizar as tarefas propostas. E, até hoje, é assim, pois a qualidade agregada é o atendimento aos requisitos.

Entretanto, com o passar dos anos, foi ficando cada vez mais complexo manter o padrão de qualidade. Em 1971, a *Intel Corporate*, criou o primeiro microprocessador em silício: o Intel 4004, que tinha o mesmo poder de processamento que o ENIAC[2] e ocupava menos espaço com custo inferior. Segundo Dijkstra (1972), essa mudança de cenário agregou um grande efeito na produção de software em um curto período de tempo.

A crise de software que culminou no inicio dos anos 70 teve como fator precursor a dificuldade do desenvolvimento de software frente a sua crescente demanda, além da imaturidade da engenharia de software e a complexidade do processo de software, como consequência os projetos estouravam prazos e orçamentos além do produto possuir baixa qualidade.

Naquela época ainda tinha outro fator que agregava aos problemas, pois programar não é uma tarefa trivial nos dias de hoje e naquela época era uma tarefa ainda mais complexa. Atualmente, possuímos ferramentas que, por mais comuns que sejam, ainda nos

---

[2] Do inglês *Electrical Numerical Integrator and Computer*, foi o primeiro computador digital eletrônico de grande escala. Criado em fevereiro de 1946 pelos cientistas norte-americanos John Eckert e John Mauchly, da *Electronic Control Company*. Fonte: Disponível em: < http://pt.wikipedia.org/wiki/Eniac>. Acesso em: 03 nov. 2009.

auxiliam, tanto no desenvolvimento como nas correções do software. Contudo, naquela época. tanto o desenvolvimento como as correções eram demasiadamente trabalhosas e demandavam muito tempo, sem contar as linguagens utilizadas anteriormente, considerada até mesmo com linguagens de baixo nível ou linguagens de máquina. Nos dias de hoje, a programação é muito mais trabalhada e possui padrões que levados em consideração visam à qualidade.

Todos esses problemas vieram à tona na primeira conferência de Engenharia de Software realizada em 1968 na Alemanha, cujo tema foi a padronização dos ciclos de desenvolvimento de software, que vem evoluindo até hoje, e teve como principal objetivo a Qualidade de Software.

2.1. Qualidade de Software

A Engenharia de Software refere à garantia da qualidade do software como um processo de normalização dos processos com o intuito de atendimento dos requisitos funcionais e não funcionais.

Para Roger Pressman, "Qualidade de software é a conformidade com requisitos funcionais e de desempenho explicitamente declarados, padrões de desenvolvimento explicitamente documentados e características implícitas, que são esperadas em todo software desenvolvido profissionalmente" (PRESSMAN, 2002).

Alexandre Bartié em seu livro enfatiza que "Qualidade de software é um processo sistemático que focaliza todas as etapas e artefatos produzidos com o objetivo de garantir a conformidade de processos e produtos, prevenindo e eliminando defeitos" (BARTIÉ, 2002).

Segundo a norma NBR ISO 8402, qualidade é a totalidade das características de uma entidade que lhe confere a capacidade de lhe satisfazer às necessidades explícitas e implícitas. Entidade é o produto resultante de um determinado processo; necessidades explícitas são as metas que o produto deve cumprir e necessidades implícitas são as visões subjetivas do produto, como diferenças entre usuários e implicações éticas.

Em outras palavras, a qualidade de software pode ser entendida como o conjunto de requisitos que o software deverá abranger para que o produto atenda as necessidades e

satisfaça os *stakeholders*[3], pois a minimização dos *bugs*[4] diminui o custo do projeto e garante a qualidade.

O processo de garantia de software é importante não apenas no resultado final, a Engenharia de Software vêm como principal relevância em todos os principais modelos de maturidade propostos atualmente. Um dos principais autores sobre Engenharia de Software, Roger Pressman, sugere métricas para garantia da qualidade de software em análise, desenho[5], codificação, teste e manutenção (PRESSMAN, 2002).

Para um software ter qualidade, não basta estar livre de defeitos, pois é uma tarefa praticamente impossível, a tendência de sistemas de software é melhorar gradativamente, o que chamamos de melhoria contínua, e a integração com as demais funcionalidades acaba acarretando novos problemas, e a facilidade de resolução de problemas, assim como a sua respectiva solução e previsão cabe à Engenharia de Software, assim como a qualidade do mesmo.

2.2. Confiabilidade de Software

Existem determinados sistemas de software que são de extrema importância nas mais variadas áreas como saúde, engenharia, contábil, entre outras, que precisam de certeza e confiabilidade no resultado e no funcionamento desses sistemas.

Segundo a norma ISO/IEC 9126 (2003), confiabilidade de software é a "capacidade do produto de software de manter um nível de desempenho especificado, quando usado em condições especificadas". Ou seja, a confiabilidade de software é, geralmente, definida como a probabilidade do software operar sem ocorrência de falhas durante um período especificado de tempo em um determinado ambiente.

Já a disponibilidade, é também um fator comprometedor a ser agregado à confiabilidade, pois o software somente poderá ser confiável durante o tempo que ele puder ser operado.

Esses atributos, confiabilidade e disponibilidade, são a definição básica de Engenharia de Confiabilidade de Software que é definida como o estudo quantitativo do

---

[3] Em português, parte interessada. No contexto da referencia a organização ou cliente.

[4] Erro no funcionamento do software originado por problemas na codificação.

[5] Alguns autores traduzem como projeto.

comportamento de sistemas de software relacionadas ao seu comportamento e aos requisitos de usuários relativo a confiabilidade. A análise da confiabilidade de software inclui:

a. Métricas de confiabilidade, na qual inclui estimação e previsão com o uso de modelos de confiabilidade de software;

b. Atributos e métricas de projeto, processo de desenvolvimento, arquitetura de sistema, ambiente operacional do software e suas implicações sobre a confiabilidade;
c. Aplicação desse conhecimento na especificação e projeto da arquitetura de software do sistema, desenvolvimento, testes, uso e manutenção.

2.3. Garantia de Qualidade

Segundo Sommerville:

> Garantia de qualidade é o processo de definição de como a qualidade de software pode ser atingida e como a organização de desenvolvimento sabe que o software possui o nível de qualidade necessário. [...] O processo da garantia da qualidade esta, principalmente, relacionado à definição e seleção de padrões que devem ser aplicados ao processo de desenvolvimento de software ou ao produto de software (SOMMERVILLE, 2007).

A garantia da qualidade visa fornecer à gerencia uma eficácia no processo de desenvolvimento de software, padronizando a qualidade dos artefatos que estão sendo criados. A ideia para garantir a qualidade é desenvolver padrões. Esses padrões são embasados em diretrizes nacionais e internacionais como ponto de partida, a equipe responsável por essa padronização deve elaborar um manual de padrões que devera ser seguidos por toda a equipe.

Todos os artefatos devem seguir as especificações do padrão adotado, modelos de qualidade como a ISO/IEC 9001 possuem padrões genéricos ilustrados atraves da FIG 1, que podem ser utilizados em diversas áreas como indústrias, comercio e fabrica de software.

FIGURA 1 – Àreas cobertas pelo modelo ISO 9001 para garantia de qualidade.
Adaptado da Fonte: Disponível em: (SOMMERVILLE, 2007)

2.4. Processos e a Qualidade

2.4.1. CMMI

Chrissis diz, de uma forma sucinta, que o CMMI (*Capability Maturity Model Integration* – Modelo Integrado de Capacitação e Maturidade) criado pela SEI (*Software Engineering Institute*), consiste nas melhores práticas direcionadas ao desenvolvimento e à manutenção de produtos e dos serviços, abrangendo todo o ciclo de vida do produto, desde sua concepção até a sua entrega e manutenção (CHRISSIS; KONRAD; SHRUM, 2003). É um dos modelos de maturidade mais aceitos no mundo no que diz respeito a medir a maturidade dos processos de uma organização. Quanto mais maduro forem seus processos, maior será a qualidade obtida no produto final, pois os níveis de maturidade ajudam a aprimorar seus processos. Isso ocorre, pois prevendo o comportamento dos processos, torna-se a organização mais madura e competitiva no mercado.

São cinco os níveis de maturidades propostos pelo CMMI. E todos os níveis correspondem à capacidade da empresa em realizar grandes projetos.

Eles são classificados nos seguintes níveis:

a. Nível 1. Inicial: Nesse nível que estão todas as empresas que não possuem nenhuma certificação, ou seja, conforme FIG. 2 os processos são como caixas pretas, o que se sabe é que os requisitos entram e o produto sai, André Koscianki diz que “geralmente isto se deve ao trabalho dos heróis que fazem muitas horas extras para compensar

planejamentos mal feitos. Pode ocorrer também de os produtos serem entregues, mas a um custo alto ou com prazo excessivo" (KOSCIANSKI, 2006). Também conhecido como processo *ad-hoc* ou caótico.

FIGURA 2 – Nível inicial do modelo CMMI.
Adaptado da Fonte: Disponível em: (PAULK, 1995)

b. Nível 2. Gerenciado: Nesse nível, gestão é a palavra chave, pois os processos começam a serem desenvolvidos de maneira organizada. Como pode ser observado na FIG. 3, inicia-se análise e medição do desempenho do projeto tornando mais fácil alguma mudança ou eventual tomada de decisão.

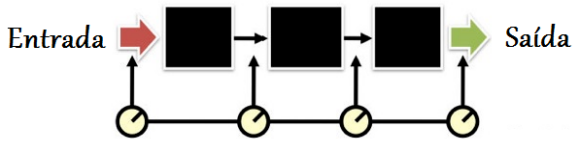

FIGURA 3 – Nível gerenciado do modelo CMMI.
Adaptado da Fonte: Disponível em: (PAULK, 1995)

c. Nível 3. Definido: "No nível 3, os processos são bem caracterizados e entendidos. A padronização de processos possibilita maior consistência nos produtos gerados pela organização" (KOSCIANSKI, 2006). Dessa forma as caixas pretas são abertas. Como pode ser observado na FIG. 4, as empresas já conseguem gerenciar e administrar seus projetos, o foco desse nível é a engenharia e o desenvolvimento.

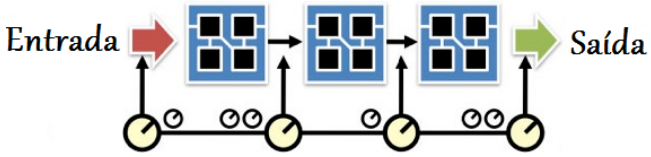

FIGURA 4 – Nível definido do modelo CMMI.
Adaptado da Fonte: Disponível em: (PAULK, 1995)

d. Nível 4. Gerenciado Quantitativamente: Nesse nível, a organização controla e gerencia seus processos e conhecem os pontos que realmente interferem no desempenho de seus processos, conforme pode ser observado na FIG 5.

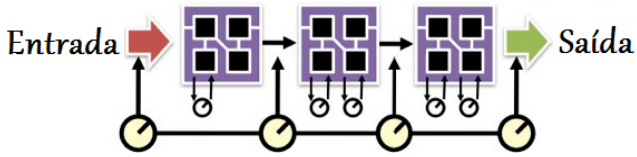

FIGURA 5 – Nível gerenciado quantitativamente do modelo CMMI.
Adaptado da Fonte: Disponível em: (PAULK, 1995)

e. Nível 5. Em Otimização: As empresas que chegaram a tal nível conseguem descobrir as causas de seus problemas, pois conhece o que pode influenciar no desempenho de seus processos. Como pode ser observado na FIG. 6, os processos sobressalentes são substituídos pelos processos otimizados, a melhoria contínua tornou o padrão da organização.

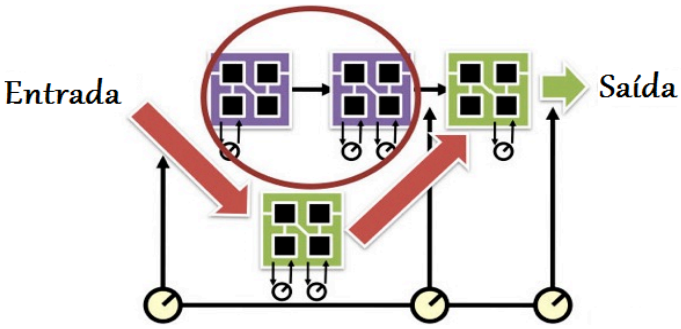

FIGURA 6 – Nível Otimizado do modelo CMMI.
Adaptado da Fonte: Disponível em: (PAULK, 1995)

Segundo André Koscianki (2006), o CMMI avalia o nível de maturidade da organização e dizer que uma empresa está em determinado nível quer dizer que ela implementa as áreas do processo de seu respectivo nível.

Áreas de Processo:

O modelo CMMI v1.2 (CMMI-DEV) contém 22 áreas de processo que são:

1. Nível de maturidade 2 - Gerenciado:

a. Gestão de Requisitos (REQM): O propósito é gerenciar os requisitos dos produtos e componentes de produto do projeto e identificar inconsistências entre esses requisitos e os planos e produtos de trabalho do projeto.

b. Planejamento de Projeto (PP): definir e documentar os objetivos do projeto, as entradas e o produto final a ser entregue. Definir os responsáveis por cada etapa do processo de criação. Definir um cronograma para o projeto.

c. Monitoramento e Controle de Projeto (PMC): O propósito é proporcionar um entendimento do progresso do projeto, de forma que ações corretivas apropriadas possam ser tomadas quando o desempenho do projeto desviar significativamente do plano.

d. Gestão de Acordo com Fornecedores (SAM): O propósito da Gestão de Acordo com Fornecedores é gerenciar a aquisição de produtos de fornecedores.

e. Medição e Análise (MA): O objetivo da medição e análise é desenvolver e sustentar a capacidade de medições utilizada para dar suporte às necessidades de gerenciamento de informações.

f. Garantia da Qualidade de Processo e Produto (PPQA): O propósito da Garantia da Qualidade de Processo e Produto é munir a equipe e a gerência com uma visão clara sobre os processos e seus produtos de trabalho associados.

g. Gestão de Configuração (CM): O propósito da Gestão de Configuração é estabelecer e manter a integridade dos produtos de trabalho, utilizando identificação de configuração, controle de configuração, balanço de configuração e auditorias de

configuração.

2. Nível de Maturidade 3 - Definido:

a. Desenvolvimento de Requisitos (RD): O propósito do Desenvolvimento de Requisitos é produzir e analisar e os requisitos de cliente, de produto e de componente de produto.

b. Solução Técnica (TS): O propósito da Solução Técnica (TS) é projetar, desenvolver e implementar soluções para requisitos. Soluções, designs e implementações englobam produtos, componentes de produto e processos de ciclo de vida relacionados ao produto isoladamente ou a combinações de produtos quando apropriado.

c. Integração de Produto (IP): O propósito da Integração de Produto é montar o produto a partir de componentes de produto, garantir que o produto integrado execute as funções de forma apropriada e entregar o produto.

d. Verificação (VER): O propósito da Verificação é assegurar que os produtos de trabalho selecionados atendem aos seus requisitos especificados.

e. Validação (VAL): O propósito da Validação é demonstrar que um produto ou componente de produto atende ao seu uso pretendido quando colocado em seu ambiente alvo.

f. Foco no Processo Organizacional (OPF): O propósito do Foco no Processo Organizacional é planejar, implementar e implantar melhorias do processo organizacional com base na compreensão dos pontos fortes e pontos fracos atuais dos processos e dos ativos de processo da organização.

g. Definição do Processo Organizacional + IPPD (OPD): O propósito da Definição do Processo Organizacional é estabelecer e manter um conjunto de ativos de processo da organização e padrões de ambiente de trabalho disponíveis para uso.
   Para IPPD, Definição do Processo Organizacional + IPPD cobre também o estabelecimento de regras e guias organizacionais que possibilitam a condução de trabalhos realizados por equipes integradas.

h. Treinamento Organizacional (OT): O propósito do Treinamento Organizacional é desenvolver as habilidades e o conhecimento das pessoas para que elas possam desempenhar seus papéis de forma eficiente e eficaz.

i. Gestão Integrada de Projeto + IPPD (OPD): O propósito da Gestão Integrada de Projeto é estabelecer e gerenciar o projeto e o ambiente dos *stackeholders* relevantes de acordo com um processo integrado e definido que é adaptado a partir do conjunto de processos padrão da organização.
   Para IPPD, a Gestão Integrada de Projeto + IPPD cobre também o estabelecimento de uma visão compartilhada para o projeto e o estabelecimento de equipes integradas que irão cumprir os objetivos do projeto.

j. Gestão de Risco (RSKM): O propósito da Gestão de Risco é identificar potenciais problemas antes que ocorram. Para isso, as atividades de tratamento de risco podem ser planejadas e colocadas em prática quando necessário, durante a vida do produto ou do projeto, para mitigar impactos indesejáveis na obtenção dos objetivos.

k. Análise de Decisão (DAR): O propósito da Análise de Decisão é analisar decisões possíveis usando um processo de avaliação

formal que avalia alternativas identificadas com relação a critérios estabelecidos.

3. Nível de Maturidade 4 – Gerenciado Quantativamente:

    a. Desempenho do Processo Organizacional (OPP): O propósito do Desempenho do Processo Organizacional é estabelecer e manter um entendimento quantitativo do desempenho do conjunto de processos padrão da organização no suporte dos objetivos de qualidade e de desempenho de processo, e prover dados de desempenho de processo, *baselines* [6]e modelos para gerenciar quantitativamente os projetos de uma organização.

    b. Gestão Quantitativa de Projeto (QPM): O propósito da Gestão Quantitativa de Projeto é gerenciar quantitativamente o processo definido do projeto para alcançar os objetivos de qualidade e de desempenho de processo estabelecidos do projeto.

4. Nível de Maturidade 5 – Em Otimização:

    a. Inovação Organizacional (OID): O propósito da Inovação Organizacional é selecionar e implementar melhorias incrementais e inovadoras que melhorem os processos e as tecnologias de uma organização de forma mensurável. As melhorias dão suporte aos objetivos de qualidade e de desempenho de processo da organização derivados dos objetivos do negócio na organização.

    b. Análise de Causa e Solução de Problemas (CAR): O propósito da Análise de Causa e Solução de Problemas é identificar causas de

[6] *Baselines* – (em português: linhas de base) Uma *baseline* é uma 'imagem' de uma versão de cada artefato no repositório do projeto.

defeitos e de outros problemas e tomar ações para evitar que ocorram no futuro.

2.4.2. MPS.BR

O modelo Melhoria de Processo de Software Brasileiro (MPS.BR) assim como o CMMI, é baseado em níveis de maturidades, voltado para pequenas e médias empresas e adaptado para perfil cultural das empresas brasileiras tendo como objetivo definir e aprimorar um modelo de melhoria e avaliação de processo de software de modo compatível com os padrões internacionais de qualidade de software. O MPS.BR é coordenado pela Associação para Promoção da Excelência da Software Brasileiro (SOFTEX) e conta com o apoio do Ministério da Ciência e Tecnologia (MCT), Financiadora de Estudos e Projetos (FINEP) e do Banco Interamericano de Desenvolvimento (BID) (SOFTEX, 2009).

A escala de níveis pode ser expressa da seguinte forma:

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

Na FIG. 7, pode ser ilustrado o nível de correlação entre o CMMI e o MPS.BR, apesar de ambas serem compostas por Áreas de Processos não existe uma equivalência direta entre eles.

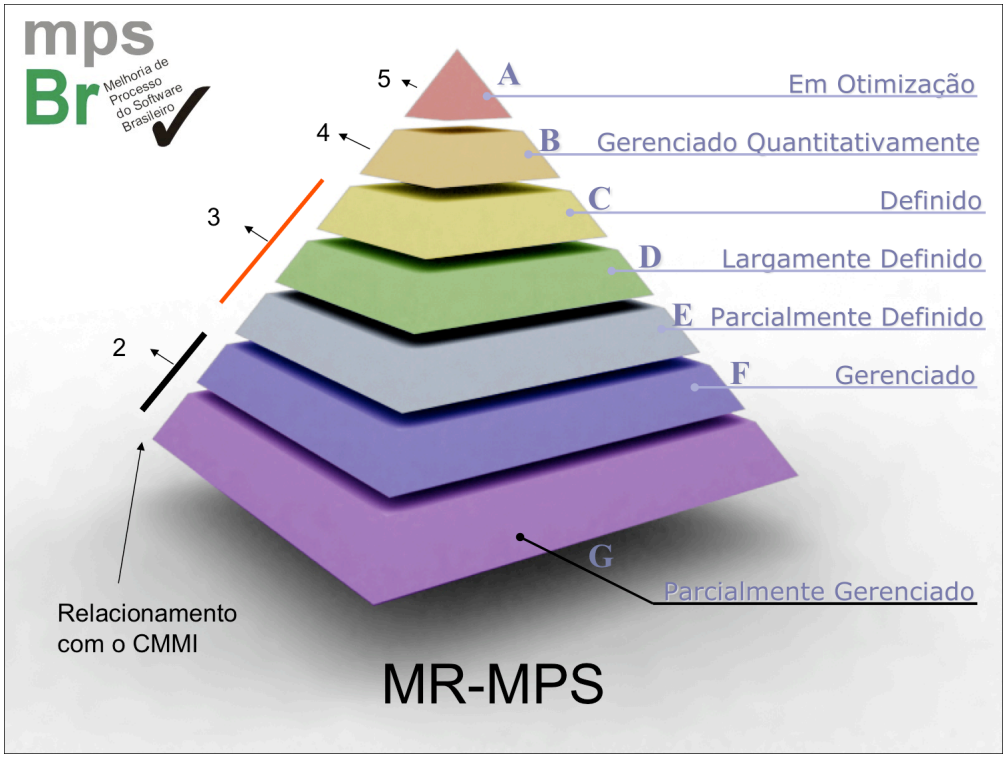

FIGURA 7 – Comparativo entre modelo CMMI e modelo MPS.BR

Fonte: Disponível em: <http://www.softex.br/mpsbr/_faq/faqDiversos.asp>. Acesso em: 31 out. 2009.

Analisando a FIG. 6, e baseando no que diz André Koscianki, podemos verificar que “os níveis do MPS.BR permitem que a empresa implante processos de uma forma mais gradual, ou seja, empresas de pequeno porte que desejam se certificar podem tomar a iniciativa de definir suas metodologias de processos” (KOSCIANSKI, 2006).

Na FIG. 8, podemos observar o crescente número de empresas certificadas e seus respectivos níveis de certificação MPS.BR, observamos que apesar do numero de empresas certificadas recuar em 2008, o nível de certificação dessas empresas vem melhorando, em 2009 são 24 empresas Nível F contra 9 em 2008, alem de outra duas empresas conseguirem o nível C e outra obter o mais alto nível de certificação MPS.BR o nível A – Em otimização. A Softex espera um aumento no numero de empresas certificadas para os próximos anos.

Totais por Níveis								
Ano	A	B	C	D	E	F	G	Totais por Ano
2005	0	0	0	0	1	3	1	5
2006	2	0	0	1	1	1	7	12
2007	1	0	0	0	1	12	41	55
Total 2005 a 2007	**3**	**0**	**0**	**1**	**3**	**16**	**49**	**72**
2008	1	0	0	0	1	9	40	51
2009	2	0	2	0	1	24	24	53
2010	0	0	0	0	0	0	0	0
Total 2008 a 2010	**3**	**0**	**2**	**0**	**2**	**33**	**64**	**104**
TOTAIS	**6**	**0**	**2**	**1**	**5**	**49**	**113**	**176**

FIGURA 8 - Nível de certificação MPS.BR segundo a Softex

Fonte: Disponível em: <http://www.softex.br/mpsbr/_avaliacoes/avaliacoes_mpsbr_total.pdf>. Acesso em: 25 nov. 2009.

2.5. Organização Internacional de Padronização (ISO)

2.5.1. ISO/IEC 9000

As normas ISO série 9000 possui um conjunto de normas internacionais para a certificação de que um produto, serviço ou processo está de acordo com as exigências especificadas em um referencial. Ela tem como objetivo desenvolver padrões de atividades que possam ser aplicadas em qualquer país.

Para André Koscianski, "a ISO 9000 é uma referência genérica em sistemas de qualidade que tem como função de promover a normalização de produtos e serviços, objetivando a satisfação dos clientes. Ela se aplica a todos os ramos de atividade, ou seja, qualquer empresa pode utilizá-la e funciona muitas vezes como um eficiente apelo de marketing" (KOSCIANSKI, 2006).

Sobre a ISO, José Carlos Maldonado diz que, "o ponto forte do modelo ISO 9000 é a flexibilidade, permitindo ser adaptado (p. ex.: QS9000 e TC9000) ou complementado (p. ex.: CMM e SPICE) para atender aos requisitos setoriais específicos" (ROCHA; MALDONADO; WEBER, 2001).

Essa norma garante que o produto que passar por esse processo de controle de qualidade apresentará padrão de qualidade semelhante ao de outro, que seguiu o mesmo processo. Convém salientar que isso não significa dizer que esses produtos terão o mesmo padrão de qualidade.

2.5.2. ISO/IEC 12207

A ISO/IEC 12207 Tecnologia da Informação – Processos de ciclo de vida de Software (*Standard for Information Technology - Software life cycle process)* visa o gerenciamento e a estruturação dos processos de ciclo de vida do software podendo ser adaptada quanto ao tipo do projeto no conjunto de processo, tarefas e atividades. Essas três cobrem o ciclo desde a concepção até o final do ciclo. Segundo a NBR ISO/IEC 12207, "ciclo de vida é a estrutura que contém processos, atividades e tarefas envolvidas no desenvolvimento, operação e manutenção de um produto de software, abrangendo a vida do sistema desde a definição de seus requisitos até o término de seu uso". José Maldonado (2001) diz que o objetivo da ISO/IEC 12207 é que a mesma estrutura dos processos e a arquitetura descrita na norma seja utilizada pelos clientes, desenvolvedores, fornecedores, operadores e gerentes.

José Maldonado diz que essa certificação vem sendo buscada por empresas prestadoras de serviços de software para que essas obtenham um patamar de qualidade para torná-las mais competitiva na economia globalizada. Diz, ainda, que para alcançar esse diferencial competitivo:

> Ela tem por objetivo auxiliar os envolvidos na produção de software a definir seus papeis, por meio de processos bem definidos, e assim proporcionar às organizações que utilizam um melhor entendimento das atividades a serem executadas nas operações que envolvem, de alguma forma o software (ROCHA; MALDONADO; WEBER, 2001).

Para isso, os processos que envolvem o ciclo de vida do software foram agrupados em três classes: Processo Fundamental, Processo de Apoio e Processo Organizacional, ilustradas através da FIG. 9.

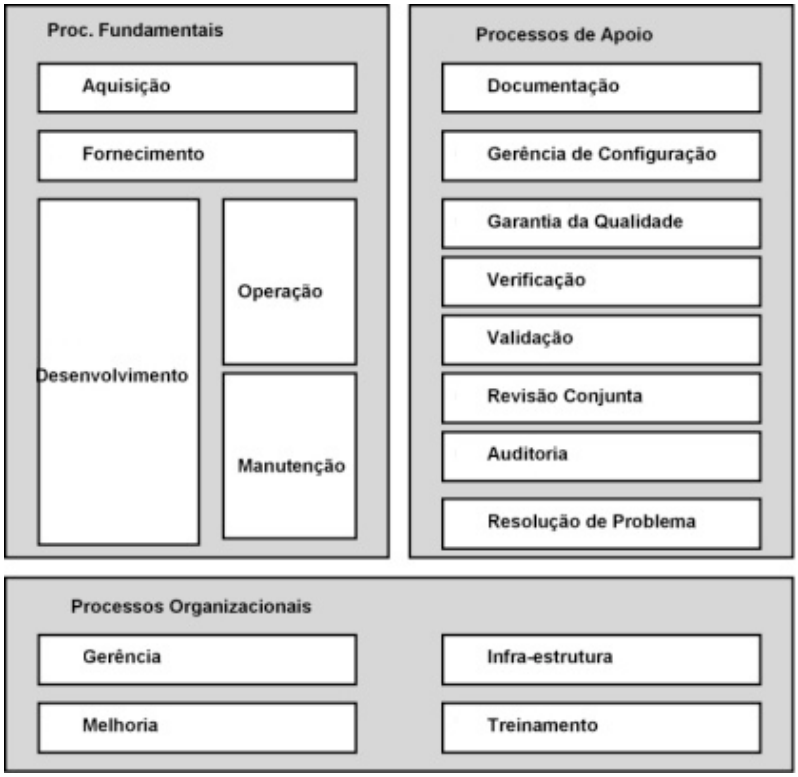

FIGURA 9 – Processos do ciclo de vida do software da NBR ISO/IEC 12207.
Adaptado da Fonte: Disponível em: (ROCHA; MALDONADO; WEBER, 2001)

A. Processo Fundamental

Como pode ser visto na FIG. 8, essa classe do ciclo de vida do software atende desde o processo de contratação passando pelo desenvolvimento operação até a manutenção do produto, segundo José Maldonado o processo fundamental está subdividido nos cinco sub-processos citado abaixo:

1. Aquisição: O adquirente[7] inicia o processo de aquisição através da necessidade de adquirir um produto ou serviço de software. Assim, inicia-se a preparação e a emissão do pedido de proposta, o propósito do processo de aquisição é obter um produto ou serviço que satisfaça as necessidades expressas pelo cliente, mediante a aceitação do sistema pelo adquirente;

[7] Organização que obteve um sistema ou produto de software (ROCHA; MALDONADO; WEBER, 2001).

2. Fornecimento: Tem como objetivo principal atender os requisitos acordados entre fornecedor[8] e adquirente. Esse sub-processo continua com o levantamento dos recursos necessários para garantir o projeto, desde o desenvolvimento até a entrega do sistema;

3. Desenvolvimento: Define as atividades do desenvolvedor[9] e contém todas as tarefas para construção do software, dentre elas: análise de requisitos, desenho, codificação, testes e a implantação;

4. Operação: Define as atividades do operador[10]. As atividades do sub-processo de operação envolvem a operação do produto de software e o suporte operacional aos usuários;

5. Manutenção: O sub-processo de manutenção é utilizado quando existe a necessidade de melhoria, adaptação ou correção no código e na documentação. Tem como objetivo modificar garantindo a integridade do produto.

Referente ao processo Fundamental José Maldonado diz que na prática, o sub-processo de Aquisição inicia o ciclo de vida de software. O sub-processo de Fornecimento responde sobre a execução dos sub-processos de Desenvolvimento, Operação e/ou Manutenção.

B. Processo Organizacional

Geralmente empregado fora do domínio de projetos e contratos específicos, o processos organizacionais estabelecem uma estrutura constituída pelos processos de ciclo de vida e pela equipe de desenvolvimento do software auxiliando na melhoria gerencial e organizacional dos processos. Ele é subdivididos no seguintes sub-processos:

1. Gerência: Nesse sub-processo, o gerente é responsável pela aquisição, fornecimento, desenvolvimento, operação, manutenção e apoio, assim como o gerenciamento do produto, projeto e das tarefas. Essa definição é genérica, ou seja, pode ser aplicada a qualquer parte do ciclo que necessita gerenciar seus processos;

---

[8] Organização que fornece um sistema ou produto ao adquirente (ROCHA; MALDONADO; WEBER, 2001).

[9] Organização que define e desenvolve o produto de software (ROCHA; MALDONADO; WEBER, 2001).

[10] Organização que provê os serviços de operação de um sistema computacional no seu ambiente de funcionamento para seus usuários (ROCHA; MALDONADO; WEBER, 2001).

2. Infra-Estrutura: Define as atividades para manter a infra-estrutura dos processos incluindo software, hardware e recursos para desenvolvimento técnicas operação e manutenção do software;

3. Melhoria: Define as atividades a serem executadas no intuito de avaliar, medir, controlar e melhorar o processo do ciclo de vida do software;

4. Treinamento: Define as atividades necessárias para manter o pessoal treinado, os processos fundamentais[11] são dependentes do treinamento e qualificação do pessoal.

C. Processo de Apoio

Segundo Maldonado (2001), esse processo tem a finalidade de auxiliar e contribuir para o sucesso e qualidade dos outros processos envolvidos no ciclo de vida do software. Um processo de apoio pode ser empregado e executado empregando um dos seguintes sub-processos:

1. Documentação: Como o próprio nome sugere, o sub-processo de documentação consiste em planejar, produzir, editar e distribuir a documentação produzida por um processo ou atividade do ciclo de vida aos *stakeholders;*

2. Gerência de Configuração: A responsabilidade desse sub-processo é gerenciar todas as mudanças, alterações, melhorias e versões presentes no ciclo de vida. André Koscianki diz que "a primeira atividade do processo consiste em planejar a gerência. Em seguida, é preciso identificar e definir os artefatos, como, por exemplo: diagramas estruturais, componentes de bibliotecas e repositórios de código" (KOSCIANSKI, 2006). Existem ferramentas que possibilitam o controle de versão e repositório de código facilitando a execução das tarefas pelos desenvolvedores e gerentes de configuração, por exemplo, o SVN (*Subversion*) ou CVS (*Concurrent Versions System*), algumas plataformas de desenvolvimento possuem esta ferramenta integrada, por exemplo, NetBeans, Visual Studio .NET, Eclipse;

3. Garantia de Qualidade: No sub-processo de qualidade, é verificado se os produtos satisfazem os requisitos e se a execução e gestão dos processos estão em conformidade com o que foi planejado;

---

[11] Sub-Processos do Processo Fundamental: Aquisição; Fornecimento; Desenvolvimento; Operação e Manutenção.

4. Verificação: Sub-processo utilizado para determinar se o software atende completamente aos requisitos ou às condições que foram impostas em atividades anteriores. O processo de verificação envolve diversas etapas do ciclo de vida, como contratos, padrões, requisitos, diagramas, codificação, integração e documentação;

5. Validação: O sub-processo de validação tem como objetivo determinar se o produto final cumpre com os objetivos de uso que foram estabelecidos. O software nesse processo deve ser submetido a testes de robustez e estresse, análise de falhas e testes de qualidade;

6. Revisão Conjunta: Define as atividades para avaliar as atividades de um processo do ciclo de vida, tais revisões são feitas tanto nos níveis de gerenciamento quanto nos níveis técnicos e são executadas ao longo da vigência do contrato;

7. Auditoria: Define atividades para determinas a adequação do produto aos requisitos, atividades como verificar se a documentação do usuário está nos padrões aplicáveis, se o produto de software foi corretamente testado e se corresponde as especificações, além de verificar se os custos e cronogramas estão sendo cumpridos.

8. Resolução de problemas: O objetivo do sub-processo de resolução de problemas é garantir, em tempo hábil, a análise e resolução de problemas encontrados nos processos incluindo não-conformidade.

2.5.3. ISO/IEC 15504

A Norma internacional ISO/IEC 15504 ou *Process Assessment* (Avaliação de Processo), também chamada de SPICE (*Software Process Improvement and Capability dEtermination*), baseia-se nos processos da ISO/IEC 12207, foca na avaliação de processos organizacionais e tem, como um de seus principais objetivos, a melhoria do processo e a determinação da capacidade dos processos. Assim como o CMMI, define níveis de capacidade cumulativos e sequenciais que avaliam como a organização está realizando um certo processo (KOSCIANSKI, 2006).

Segundo Koscianski (2006), a versão mais atual da norma 15504, diferentemente das anteriores não define processos, define modelos de referencia de processo (RPM – *Process Reference Model*). O RPM contém as descrições de escopo e uma definição de requisitos, que são os resultados desejados da execução de cada processo. As medições são definidas pelo modelo PAM (*Process Assessment Model*) que identifica os elementos que serão analisados na organização.

Assim, essa norma se divide em duas dimensões: dimensão de processo, que verifica se os processos são ou não executados (*assessment of process performance*) e dimensão de capacidade, que avalia o estado dos processos e as melhores práticas visando obter a avaliação de capacidade para empreender um determinado projeto (*assessment of process capability*).

O PAM se divide em níveis de capacidades para os processos:

- Nível 0: Incompleto – O processo não atinge seus objetivos, ou não é implementado;
- Nível 1: Executado - Os objetivos são atingidos mesmo que de forma pouco planejada;
- Nível 2: Gerenciado – Os produtos são desenvolvidos de forma apropriada e controlada, além disso, os processos desse nível são medidos e planejados;
- Nível 3: Estabelecido – Conta um processo consistente;
- Nível 4: Previsível – Tem-se o controle do processo, ou seja, pode verificar se está dentro dos limites ou atingindo os resultados;
- Nível 5: Otimizado – O processo está em constante adaptação para torná-lo mais eficiente, possibilitando atingir os objetivos do projeto.

É importante ressaltar que a dimensão de processo mencionada está relacionado apenas ao nível 1[12] do PAM enquanto a dimensão de capacidade engloba todos os níveis.

Segundo Koscianski:

> A dimensão de capacidade permite uma avaliação mais detalhada dos processos executados por uma organização. Enquanto a dimensão de processos se limita à verificação de execução ou não dos processos, a dimensão de capacidade leva a uma avaliação de nível semelhantes aos do CMMI (KOSCIANSKI, 2006).

Ou seja, a dimensão de melhoria de processo pode ser utilizada por uma empresa que busca melhorias internas, realizando avaliações e gerando um perfil do processo.

Referente à dimensão de capacidade, José Maldonado (2001) diz que a capacidade de processo de uma organização pode ser utilizada quando uma empresa tem o objetivo de contratar a prestação de serviço ou fornecimento de produtos de terceiros e quando se deseja

[12] Nível de capacidade Executado da norma ISO/IEC 15504.

obter seu perfil de capacidade a fim de estimar o risco associado à contratação daquele fornecedor em potencial para auxiliar na tomada de decisão de contratá-lo ou não.

3 Processos do Ciclo de Vida do Software

Processos do Ciclo de Vida do Software ou Modelos Prescritivos de Processo são os grupos de atividades relacionadas com o processo de criação de software, muitas vezes divididas em etapas distintas co-relacionadas podendo ser, ou não, concomitantes.

Segundo Pressman, "modelos prescritivos de processos definem um conjunto distinto de atividades, ações, tarefas, marcos e produtos de trabalho que são necessários para fazer engenharia de software com alta qualidade. Esses modelos de processos não são perfeitos, mas efetivamente oferecem um roteiro útil para o trabalho de engenharia de software" (PRESSMAN, 2002).

A ideia principal da definição de um ciclo de vida do software é utilizar um modelo que define um grupo de atividade e a forma com que elas se relacionam, podendo ser alternada quanto à necessidade do produto ou do processo. Não é uma regra a ser seguida, mas tem como finalidade sugerir uma referência para assegurar a continuidade do software.

Exemplificando, existem diversos tipos diferentes de ciclos de vida do software. Primeiramente, o ciclo que foi o mais bem aceito até meados da década de 1980 foi o modelo em cascata, conhecido também como modelo clássico. Sommerville diz que, “o ciclo de vida clássico é um dos modelos mais conhecidos na Engenharia de Software” (SOMMERVILLE, 2003).

O modelo clássico sugere uma abordagem sistemática e sequencial para o desenvolvimento de software. Nessa abordagem, se inicia com a especificação dos requisitos pelo cliente e progride ao longo do planejamento, modelagem, construção e implantação, culminando na manutenção progressiva do software finalizado. Segundo Brooks, “a principal falha do modelo cascata é que ele assume a construção de todo o sistema de uma só vez” (BROOKS, 1994). As etapas do modelo clássico podem ser observadas através da FIG. 10.

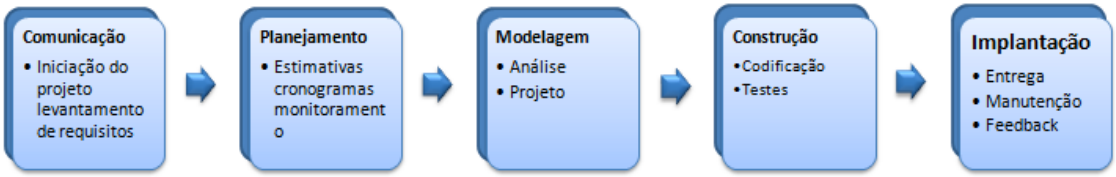

FIGURA 10 – Modelo Cascata.

Fonte: Dos Autores.

A evolução desse modelo, devido às necessidades e constantes alterações dos requisitos com o desenvolver do projeto, tornou o modelo mais interativo, originando o modelo incremental, que nada mais é que o modelo cascata sequencial com “incrementos” de módulos para sua otimização, com o intuito de satisfazer o núcleo do produto, ou raiz do produto, assim era chamado.

Esse modelo é particularmente útil quando a mão-de-obra para o desenvolvimento é escassa para que seja completado todo o desenvolvimento no prazo pré-estabelecido, pois os primeiros implementos podem ser desenvolvidos com um número menor de colaboradores. Como pode ser observado através da FIG. 11, em um determinado momento vários incrementos podem estar sendo desenvolvidos simultaneamente, cada incremento segue o modelo seguencial visto. Desse modo as funcionalidades e características do produto aparecem com o decorrer do tempo.

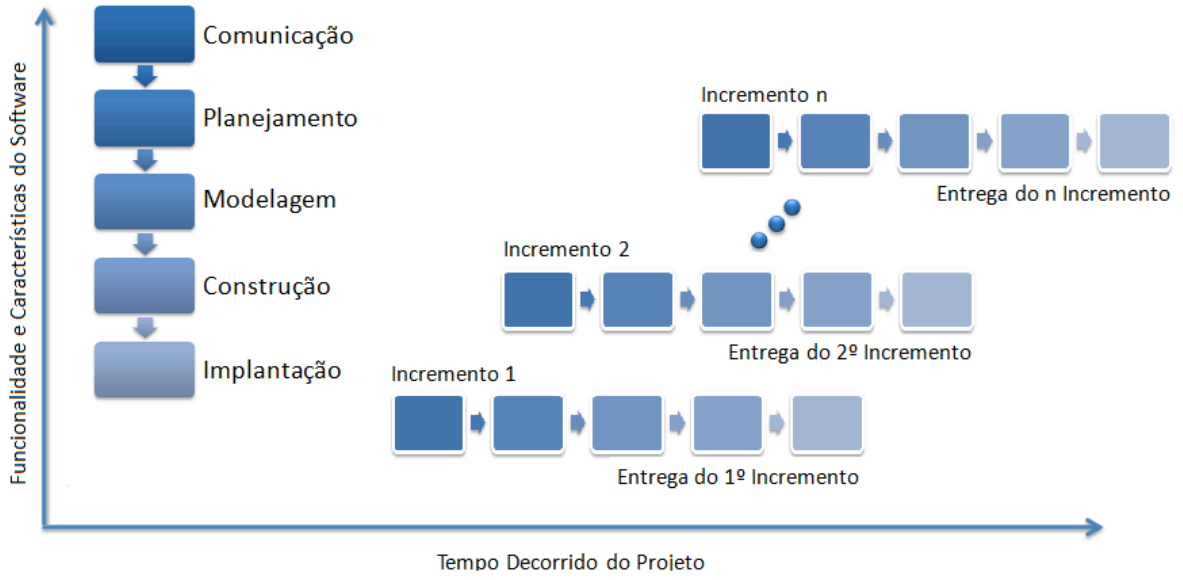

FIGURA 11 – Modelo Cascata Incremental.

Fonte: Dos Autores.

Mesmo o modelo incremental sendo uma otimização do modelo clássico, com o passar dos anos e a constante demanda e necessidade de sistemas de software foi necessária a criação de um modelo que aperfeiçoasse não somente o processo, mas o tempo de desenvolvimento do software. A partir disso, levantou-se a questão da divisão do projeto em

módulos e, com base no modelo incremental, propuseram o modelo RAD (*Rapid Application Development*, desenvolvimento rápido de aplicação) que é uma implementação rápida do modelo incremental em que o desempenho é otimizado através do desenvolvimento através de uma abordagem de construção através de componentes.

O modelo RAD se enquadra nos modelos genéricos anteriormente citados, mas tem como objetivo a construção de um sistema completamente funcional, desde que todos os requisitos sejam completamente compreendidos, em um período entre 60 a 90 dias. Isso ocorre devido a um planejamento que, se bem feito e o sistema for passível de modularização, divide as tarefas em paralelo, não demandando espera de conclusão de módulos.

Segundo Pressman, "A modelagem do Modelo RAD abrange três principais fases – modelagem do negócio, modelagem dos dados e modelagem dos processos – e estabelece representações de projeto que servem com base para a atividade de construção do RAD" (PRESSMAN, 2002).

Através da FIG. 12 podemos observar a divisão das tarefas propostas pelo modelo RAD em um gráfico em função do tempo. Após o planejamento várias equipes distintas trabalham com a modelagem de negócio, modelagem de dados e modelagem de processos e constroem os módulos utilizando o reuso de componentes, geração automática de códigos e testes. Finalizada a construção de cada módulo, esses são integrados e implantados no cliente.

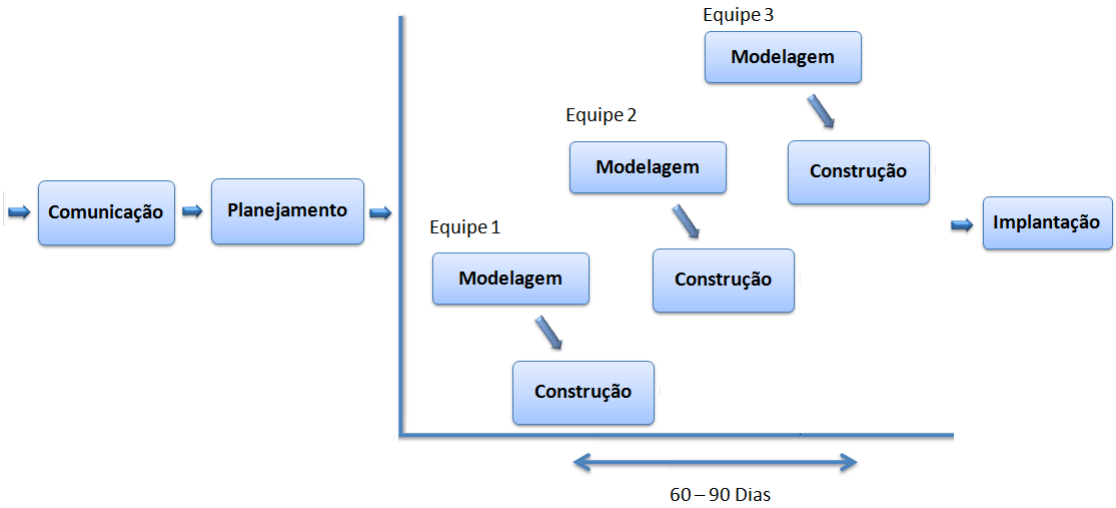

FIGURA 12 – Modelo RAD.

Fonte: Dos Autores.

Esses modelos citados quando bem estruturados e definidos garantem a engenharia de software com qualidade. Entretanto, não garantem a qualidade do produto

como um todo. Alguns dos modelos apresentados têm como objetivo definir uma visão do projeto dividido em partes. Em alguns casos, o levantamento dos requisitos pode não ter sido feito de maneira coerente ou esses requisitos não são claros, não por falhas no levantamento dos requisitos ou competência do analista, mas por regras de negócio complexas ou até mesmo pelo cliente não especificar seus requisitos de maneira objetiva. Não só os requisitos podem falhar como os desenvolvedores estão suscetíveis a falhas na codificação das regras e requisitos.

Com o intuito de atender essa necessidade, criou-se um modelo de desenvolvimento através de prototipagem, em que o desenho das telas são criados como um exemplo,exemplificando tanto para o cliente quanto para o desenvolvedor, tornando claros os requisitos.

Esse modelo, para Pressman (2002), mesmo sendo um modelo de processo de software, muitas vezes é utilizado dentro de outros modelos, por motivo de melhor entendimento entre as partes interessadas. Pressman sugere que esse modelo seja utilizado no desenvolvimento de sistemas de grande porte nos quais representem um certo grau de dificuldade na descrição rigorosa dos requisitos .

Segundo Brooks, o protótipo cumpriria a finalidade descrita da seguinte forma:

> Na maioria dos projetos, o primeiro sistema construído consegue ser apenas utilizável. Pode ser muito lento, muito grande, muito complicado de usar, ou tudo isso ao mesmo tempo. Não há outra alternativa senão começar de novo a construir uma versão reprojetada, na qual esses problemas são resolvidos. [...] Quando um novo conceito de sistema ou uma nova tecnologia é usada deve-se construir um sistema para ser descartado, porque nem mesmo o melhor planejamento é tão onisciente para fazer certo na primeira vez. A questão de gerencia, entretanto, não é se o sistema piloto elaborado deve ser descartado. Ele será. A única questão é planejar antecipadamente a construção de um descartável, ou prometer entregar o software descartável aos clientes (BROOKS, 1975).

Ao contrário dos modelos visto até aqui, a ideia básica desse modelo é que ao invés de manter inalterado os requisitos durante o projeto e codificação, um protótipo é desenvolvido para ajudar no entendimento do cliente sobre os requisitos do sistema. Esse desenvolvimento passa por um projeto, codificação e teste, sendo que cada uma dessas fases não é executada formalmente. Brooks explica um protótipo como:

> Um protótipo de um sistema de software é algo que simula as importantes interfaces e funções do sistema que se pretende construir, tipicamente representa a linha principal de atividades da aplicação. A proposta do protótipo é fazer real o conceito da estrutura especificada, só então o cliente pode testá-lo para consistência e usabilidade (BROOKS, 1975).

Dessa forma, utilizando os protótipos dos menus, telas e funcionalidades do sistema, o cliente então pode entender melhor os requisitos. Baseado na avaliação do cliente, a próxima versão do protótipo incorpora as mudanças e sugestões então relacionadas. Pressman ressalta que “um perigo está associado ao ciclo de vida da prototipação: o protótipo pode ser considerado como uma primeira versão do produto de software (PRESSMAN, 1995). Na FIG. 13, observamos o ciclo dos protótipos que seguem a sequência; comunicação; plano rápido; modelagem e projeto rápido; construção do protótipo implantação e entrega de *feedback* voltando ao início da próxima versão do protótipo.

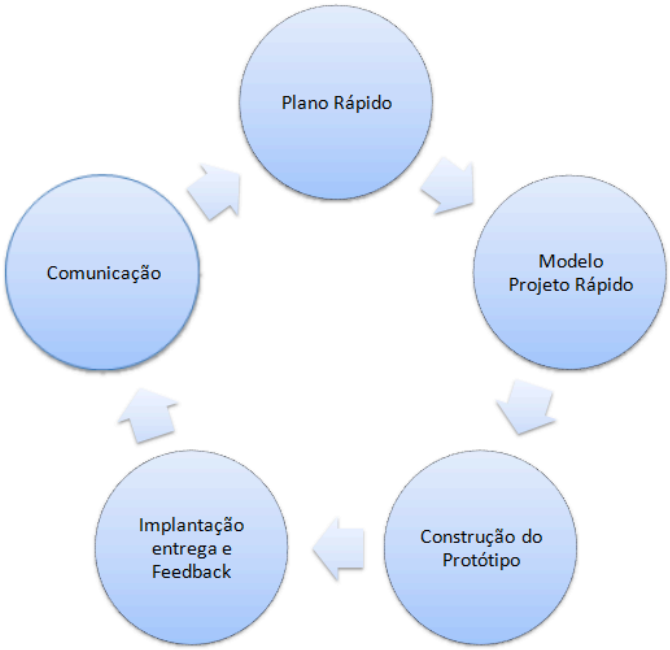

FIGURA 13 – Modelo com base em Prototipagem.

Fonte: Dos Autores.

Com base nos modelos descritos, Boehm (1988), propôs um modelo interativo, que combina a iteração do modelo de prototipagem com o modelo mais estruturado e controla em cascata, conhecido como o Modelo Espiral.

Como na FIG. 14, o ciclo no espiral começa no centro e vem contornando para fora e, a cada interação ao redor do anel, versões progressivas do software são construídas. Assim como no modelo de prototipagem, no modelo espiral também é possível refinar os requisitos à medida que o software está sendo desenvolvido.

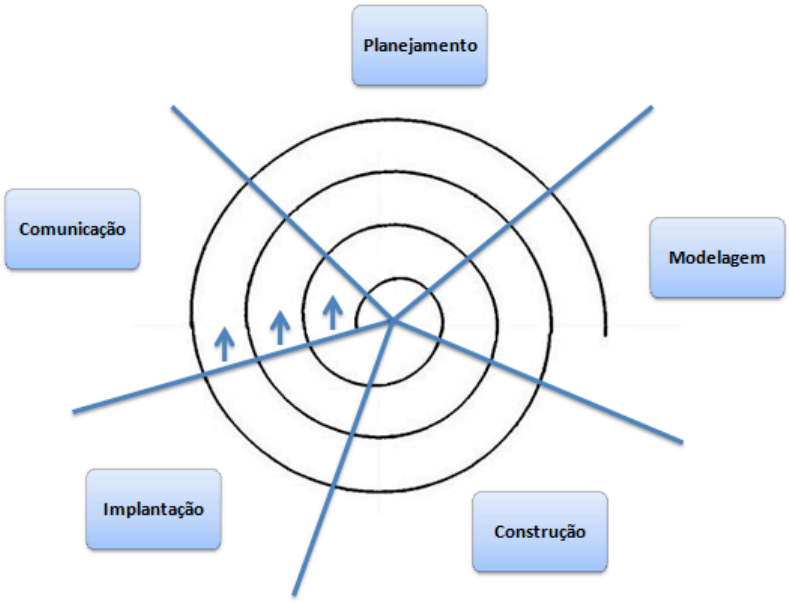

FIGURA 14 – Modelo Espiral.

Fonte: Dos Autores.

Boehm descreve seu modelo da seguinte maneira:

> O modelo espiral de desenvolvimento é um gerador de modelo de processo guiado por risco usado para guiar a engenharia de sistemas em software com vários interessados concorrentes. Eles têm duas principais características distintas. A primeira é uma abordagem cíclica, para aumentar incremetalmente o grau de definição e implementação de um sistema enquanto diminui seu grau de risco. A outra é um conjunto de marcos de ancoragem, para garantir o comprometimento dos interessados com soluções exequíveis e mutuamente satisfatórias para o sistema (Boehm, 1988).

3.1 Planejamento da Qualidade do Software

O Planejamento da Qualidade de um Software se baseia primeiramente no levantamento de toda a importância relevante com o intuito de satisfazer as necessidades alvo.

De acordo com o PMBOK (2004),

> Os processos de gerenciamento da qualidade do projeto incluem todas as atividades da organização executora que determinam as responsabilidades, os objetivos e as políticas de qualidade, de modo que o projeto atenda às necessidades que motivaram sua realização. Eles implementam o sistema de gerenciamento da qualidade através da política, dos procedimentos e dos processos de planejamento da qualidade, garantia da qualidade e controle da qualidade, com atividades de melhoria contínua dos processos conduzidas do início ao fim, conforme adequado (PMBOK, 2004).

Usualmente, ao se planejar a qualidade, utiliza-se o método de criação de um *checklist*[13], em que são listados todas as importâncias levantadas para a satisfação do objetivo do software, e regularmente, utilizam-no para ter certeza que os itens listados estejam sendo seguidos e em conformidade.

3.2 Prevenção de Defeitos

A prevenção de defeitos de software é uma maneira de tentar antecipar a localização de defeitos em um determinado software através de técnicas especificas de análise que, em muitas vezes, agilizam e localizam cerca de 80% de defeitos antes de chegarem ao cliente final.

Essas técnicas de localização de erros, tais como análise estática, teste de regressão, automatização de testes, testes funcionais visam beneficiar o processo, pois ajudam no planejamento e execução de testes, identificam pontos de melhoria e aperfeiçoam o código.

3.3 Auditorias no Processo de Desenvolvimento

As Auditorias no Processo de Desenvolvimento de Software visam à verificação da maneira como a empresa está executando os seus principais processos no desenvolvimento e se esse está de acordo com os objetivos, metas e estratégias definidas pelo cliente. Verifica também se o processo de desenvolvimento e implantação de sistemas da empresa está sendo realizado de acordo com as normas e padrões da empresa, dentro dos requisitos técnicos de qualidade, confiabilidade e segurança. Segundo a ISO/IEC 12207, a auditoria possui o propósito de determinar independentemente a conformidade dos produtos e processos contra os requisitos definidos.

As auditorias envolvem: (i) Determinação da missão, objetivos, metas e estratégias da empresa; (ii) Levantamento de como os processos de desenvolvimento que devem dar suporte a essas definições estão sendo executados; (iii) Avaliação de todos os sistemas que dão suporte ao desenvolvimento e a esses processos, para verificar como isso está sendo realizado; (iv) Identificação de todos os desvios e riscos associados à

[13] Checklist - Lista de tarefas a serem executadas.

implementação de cada processo e seus requisitos não funcionais; (v) Apresentação alternativas de como alinhar os processos e seus suportes de T.I. com as necessidades da empresa e como diminuir os riscos identificados.

O trabalho requer a análise de todos os sistemas em desenvolvimento (tanto interna como externamente) na empresa, para avaliar o seu grau de adequação aos critérios estabelecidos. A princípio devem ser identificados todos os padrões da empresa e para cada padrão serão acrescidos critérios técnicos de qualidade para compor um gabarito completo para se efetuar as avaliações. Cada sistema em desenvolvimento será avaliado e identificado o seu grau de aderência aos critérios e os riscos associados a cada projeto. O relatório final deve apontar os riscos e as alternativas para a sua eliminação

3.4 Automatização de Testes

Os testes de software em um processo de desenvolvimento é em sua maioria uma tarefa árdua e complicada, pois assim como o desenvolvedor está sujeito a falhas durante o desenvolvimento, o testados também está susceptível, e, muitas vezes, por grandes volumes de informações essas falhas pode acabar passando despercebidas.

Dentre muitos dos testes realizados como o teste de unidade, em que se testa uma unidade funcional do sistema. O teste de integração, em que o objetivo é encontrar falhas provenientes da integração interna dos componentes de um sistema. Contudo isso não é trivial. Na prática, os volumes de código e de iterações acabam deixando os testes complexos. Alem disso, acabam surgindo o problema de que a correção de determinado defeito, faz surgir novos defeitos.

Com base nisso surgiu à necessidade da criação dos testes automatizados, que variam desde o teste de unidade, onde é testado um código específico e sua funcionalidade individual até o teste de sistema em que testa-se o sistema inteiro exercendo todas as suas funcionalidades.

Abaixo relacionaremos os testes automatizados mais utilizados e que são de grande valia na detecção de falhas no ciclo de vida do software:

a. Testes de unidade, geralmente, são escritos pelo próprio desenvolvedor e mesmo testando uma parte pequena do sistema, é muito importância.

Roger S. Pressman descreve testes de unidade da seguinte maneira:

> O teste de unidade focaliza o esforço de verificação na menor unidade de projeto do software – componente ou módulo de software. Usando a descrição de projeto no nível de componente como guia, caminhos de controle importantes são testados para descobrir erros dentro dos limites do módulo e as estruturas de dados dentro dos limites de componente. Esse tipo de teste pode ser conduzido em paralelo para diversos componentes (PRESSMAN, 2002, p. 295)

A maioria dos defeitos encontrados pelo cliente poderia ser resolvido com os testes de unidade. Assim sua automatização é de grande importância, pois como é uma parte pequena muitas vezes é cansativo de ser re-testado à medida que o sistema seja corrigido e implementado novas funcionalidades. De maneira similar, também ocorre com os testes de integração.

b. Teste de integração são os testes realizados para a verificação dos módulos, ou seja, após todos os testes de unidade ter sido realizado, façam-se os testes da integração destes módulos em um contexto único, ou seja, no sistema que eles irão interagir.

Segundo Pressman:

> Teste de integração é uma técnica sistemática para construir a arquitetura do software enquanto, ao mesmo tempo, conduz testes para descobrir erros associados às interfaces. O objetivo é, a partir de componentes testeados no nível de unidade, construir uma estrutura de programa determinada pelo projeto (PRESSMAN, 2002, p. 297)

c. Testes de regressão é um teste, onde a cada versão ou ciclo de vida do sistema um novo teste é realizado, principalmente nos sistemas modulados e componentizados. Segundo Peters e Pedrycz, “o objetivo do teste de regressão é reexecutar automaticamente alguns testes em um software sempre que uma pequena mudança ocorrer no produto” (PETERS; PEDRYCZ, 2001, p. 412). À medida que é finalizada uma etapa é feito então o teste de regressão, esse teste é feito de forma manual, ou seja, além de demandar muito tempo, é complicado. A automatização vem como solução para isto, pois de forma automatizada todos os testes seriam feitos no novo módulo e seria refeito os testes na adaptação com o restante do sistema, verificando se há possíveis impactos nisso, ou não.

d. Testes fumaça são testes de regressão desenvolvimentos paralelamente ao software, ou seja, eles são efetuados enquanto o software ainda é desenvolvido, serve como

uma forma de teste de regressão conjuntamente com de integração comumente ao desenvolvimento de software.

McConnell (1996) citado por Pressman (2002, p.300) descreve o teste fumaça do seguinte modo:

> O teste fumaça deve exercitar o sistema inteiro de ponta a ponta. Ele não precisa ser exaustivo, mas deve ser capaz de expor os problemas principais. O teste fumaça deve ser suficientemente rigoroso para que, se a construção passar, você possa assumir que ela é suficientemente estável para ser testada mais rigorosamente.

e. A automatização dos testes funcionais é uma tarefa complicada, hoje no mercado existem algumas ferramentas que propõem uma solução para isso, como por exemplo, a ferramenta da IBM, IBM *Rational Functional Tester* e da Borland o *SilkTets, TestCase* e o *Selenium* uma das principais ferramentas para teste na plataforma Web. Um testador ao realizar todos os testes comuns e detalhados "grava" esses testes, que posteriormente as ferramentas de auxílio geram um *script* automaticamente com os testes o que facilita muito em futuros testes tanto de regressão para impactos e quanto para testes funcionais.

3.5 Organização e Documentação

A organização de um processo de desenvolvimento de software é um fator criterioso para o sucesso do modelo adotado e tem como base para os principais modelos de qualidade atuais que citamos como o CMMI e o MPS.BR.

Na Engenharia de Software e no Gerenciamento de Projetos, uma metodologia é um conjunto estruturado de práticas (por exemplo: Material de Treinamento, Programas de educação formais, Planilhas, e Diagramas) que pode ser repetível durante o processo de produção de software.

Este material, desde que documentado, passa a ser aprimorado a medida que esta sendo utilizado, passando a ter um papel fundamental no desenvolvimento do processo de uma empresa. O histórico de projetos passados serve com a experiência para projetos futuros, podendo com isso evoluir sempre buscando como objetivo a qualidade.

Um novo projeto a ser desenvolvido, seguindo a Engenharia de Software, inicializa-se com o levantamento de requisitos, seguindo a ideia que qualidade é o atendimento dos requisitos, esses bem documentados, fazem com que o produto desenvolvido possa atender ao levantado desde que esteja bem claro e evidente.

Foi criado padrões de documentação de processos e requisitos, o mais utilizado e aprovado mundialmente é a *Unified Modeling Language* (UML) que é uma linguagem de modelagem não proprietária.

Alguns processos de desenvolvimento de software utilizam a UML como base para todas as documentações em todos os processos envolvidos. O RUP, abreviação de *Rational Unified Process* (ou Processo Unificado da Rational), é um processo proprietário de Engenharia de software criado pela *Rational Software Corporation*, adquirida pela IBM, foi a pioneira na criação de um processo de desenvolvimento de software baseada em orientação a objetos, e até hoje é utilizada como referência na maioria dos processos.

Basicamente os objetivos da UML são: especificação; documentação; estruturação para sub-visualização e maior visualização lógica do desenvolvimento completo de um sistema de informação. A UML é um modo de padronizar as formas de modelagem.

A documentação normalmente é dividida em duas partes, são elas a Documentação Técnica e a Documentação de uso. A primeira, é direcionada aos desenvolvedores, é onde se usa atualmente a UML descrita anteriormente entre outras. A outra documentação é a gerada para que o cliente, usuário final, saiba utilizar o software conhecido como manual de uso.

Através da gestão de defeitos é possível definir práticas para prevenir os defeitos e minimizar os riscos de um projeto. A utilização de ferramentas automatizadas para cadastros de ocorrências próprias ou ferramentas já disponíveis no mercado como o *Bugzilla* da *Mozilla* ou *Mantis Bug Treckking* do grupo *Mantis*[14], ajudam a gerenciar e registrar ocorrências durante o projeto ou até mesmo na fase de manutenção do sistema. Essas ferramentas além de oferecer uma base comum para a entrada de informações, também oferece um meio para comunicação integrada entre a equipe de desenvolvimento e o time de teste. Dessa forma, uma vez que o defeito for encontrado, esse deverá ser reportado por meio do mecanismo

[14] Mantis Bug Tracker é uma ferramenta baseada na web que tem como principal função gerenciar bugs de outros softwares.(Fonte: Dos Autores)

estabelecido no processo de gestão de defeitos, esse *bug* pode ser direcionado a um desenvolvedor ou uma equipe de desenvolvimento, que fica encarregado de solucioná-lo.

Por meio de ferramentas de cadastro de ocorrências e possível gerar relatórios de gestão e estabelecer métricas utilizando essas ferramentas, assim, os gestores do projeto poderão promover a melhoria contínua do processo estabelecido.

3.6 Melhoria Contínua

A melhoria contínua é o grande objetivo dos programas de qualidade e produtividade. Melhoria é a transição para um melhor estado ou condição, normalmente, gerando vantagens. Para Harrington (1993), a melhoria contínua é a busca da perfeição. Para tal, a melhoria contínua vai além da definição de qualidade que, segundo ele, "é sempre fazer corretamente o trabalho", assumindo que, perfeição, "é sempre fazer corretamente o trabalho certo", com o objetivo de satisfazer os clientes internos e externos. Os clientes são assim definidos pelo autor: Clientes externos são aqueles de fora da empresa, que recebem o produto ou serviço final. Clientes internos são aqueles localizados dentro da cadeia de atividades da organização, que não recebem diretamente a saída do processo, mas são afetados se o processo gerar saídas erradas ou atrasadas.

Um fator diretamente relacionado para a melhoria contínua de um processo de qualidade é a gestão do conhecimento, partindo do pressuposto que o conhecimento em um projeto relacionado com pessoas, à gestão destas pessoas é uma forma de gerir o conhecimento. Nada tão importante como o conhecimento de quem faz parte do processo. Gerenciar corretamente um projeto, não é somente seguir o processo com base no que foi feito, é também incentivar e ajudar os demais a fazerem o mesmo, pois um projeto envolve a todos.

Segundo Martin[15] (1998), é comum pensar somente nas grandes melhorias, porém pequenas mudanças podem ser grandes nos quesitos qualidade e produtividade. A melhoria não é um fim em si próprio, portanto precisa ser contínua. O autor coloca, também, que a melhoria contínua de processos é baseada no método japonês chamado KAIZEN, em que "todos melhoram tudo, o tempo todo". Ou seja, cada participante da organização identifica problemas, fazem análises e propõem soluções. Ainda segundo o ele, no ocidente o KAIZEN

[15] MARTIN, James. **A grande transição**. Ed. Futura, São Paulo, 1998.

pode ser traduzido com TQM (*Total Quality Management*), TQC (*Total Quality Control*), TPM (*Total Produtivity Management*), entre outros.

4 Estudo de Caso

A escolha desse caso para estudo foi em função de ele trabalhar com aspectos referentes ao aperfeiçoamento da Gerencia da Qualidade de Software de um sistema de distribuição de serviços de uma grande empresa. Esses aspectos são abordados sob o ponto de vista conceitual e de implementação de forma análoga aos aspectos apresentados ao longo do estudo de caso.

Nessa empresa, todo o processo da distribuição é informatizado, desde as informações técnicas, comercias, reclamações, solicitações de serviços dos consumidores através da central de atendimento da empresa e também os serviços executados pelas equipes de campo.

Todas essas reclamações registradas, serviços programados de manutenção e demais serviços relacionados são registrados na base de dados desse sistema, que por meio do software *on-line* integrados com diversos outros sistemas cabendo a ele o gerenciamento por parte dos centros de distribuição. Contudo, esse sistema é de extrema importância a sua qualidade devendo ser priorizado seu funcionamento pleno 24 horas por sete dias da semana, sem interrupção ou falha. Toda e qualquer falha no fornecimento de energia, é registrado no órgão regulador e cabendo a designação da solução em campo ao sistema.

O sistema foi implantado, em 2002, desde então sua importância vem aumentando gradativamente e consequentemente seu porte em virtude destes anos. Quanto maior um sistema, maior sua complexidade, portanto a gerencia da qualidade de software é de total relevância, uma vez que sua manutenção e melhoria deverão ser constantes.

4.1 Problemas

Através da FIG. 15 exemplificamos como era o controle da manutenção do sistema anteriormente:

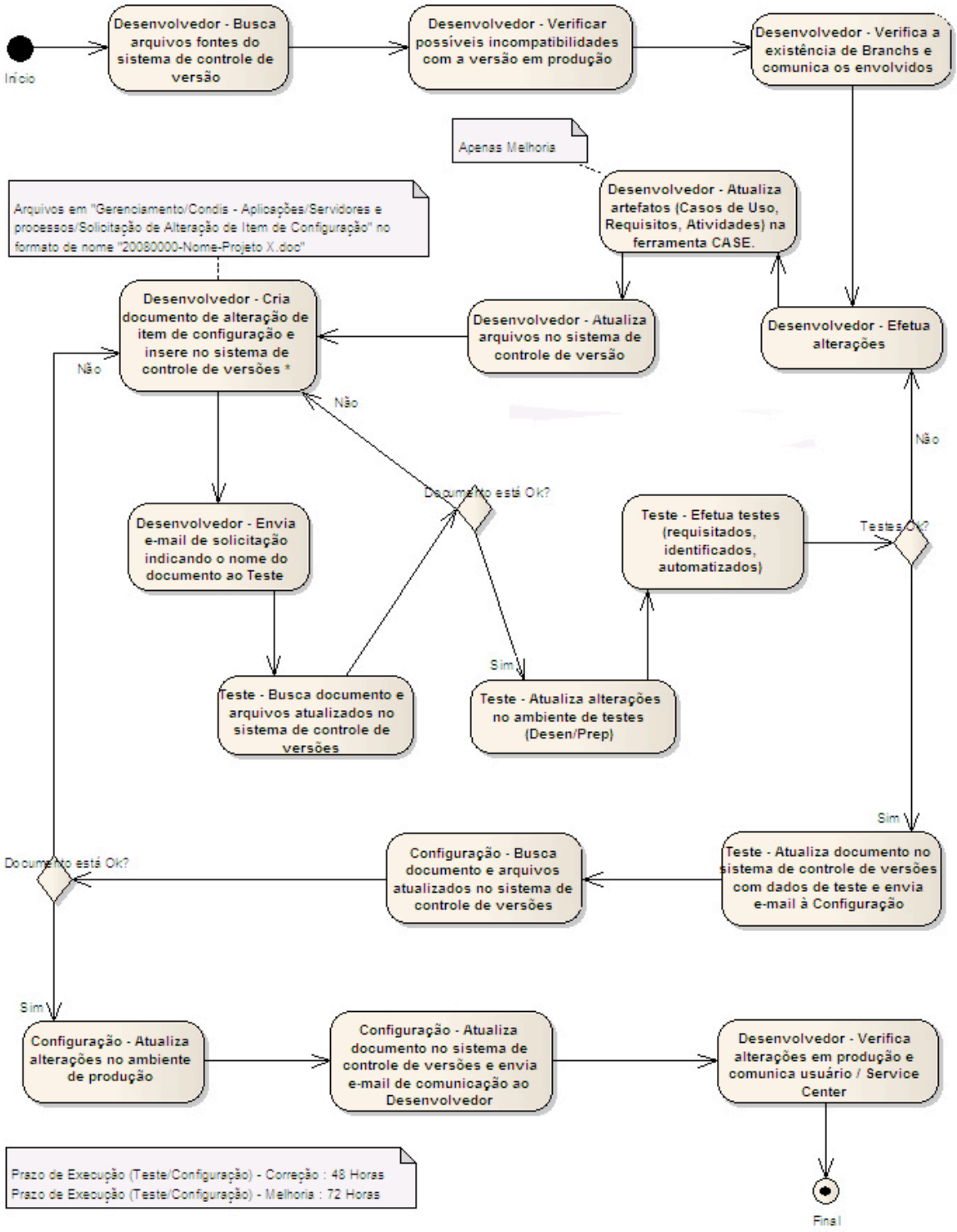

FIGURA 15 – Diagrama de Atividades referente à manutenção de problemas do sistema

Fonte: Dos autores.

Com a evolução desse sistema os problemas começaram a vir a tona, uma vez que, não conseguia garantir a qualidade, pois na maioria das vezes os testes falhavam por diversos outros fatores, como não integridade do banco de desenvolvimento, ou má compatibilidade com os demais sistemas integrados.

O fluxo inicializava quando o desenvolvedor recebia do gerente de projetos uma determinada demanda levantada pelos usuários do sistema. O desenvolvedor então verifica junto ao suporte do sistema o problema, podendo adotar como medida de correção a alteração de uma tela especifica, ou de um módulo inteiro do sistema. Havia também os problemas reportados pelos próprios analistas de suporte e pela equipe de testes.

O desenvolvedor envolvido era encarregado de buscar nos arquivos do sistema de controle de versões, o sistema utilizado atualmente é o CVS, ou *Concurrent Version System* (Sistema de Versões Concorrentes). Esse é um sistema de controle que permite trabalhar com diversas versões de arquivos simultaneamente, essas versões ficam organizados no repositório que pode ser local ou remoto, mantendo assim as versões antigas e os *logs* de atualização e manipulou dos arquivos.

Buscadas os arquivos, o desenvolvedor analisa as inconformidades relatadas, verificando se existe outro membro do desenvolvimento fazendo alterações no código fonte, e ao diagnosticar a falha, cabe a ele verificar se a documentação, caso exista, esta de acordo com o problema relatado ou se foi documentado erroneamente.

Após a alteração do problema cabe o usuário criar um documento de configuração, este por sua vez, tinha um modelo a ser seguido e era descrito todas as alterações feitas nele para que seja aplicado em produção igualmente. Este documento é um fator chave no problema, o que deveria ser a solução, pois não é sempre, mas o desenvolvedor acabava esquecendo de relatar neste documento alterações que poderiam vir a impactar no restante do processo, uma vez que, no ambiente de desenvolvimento onde era realizado os testes o funcionamento estava correto, não descrito as alterações feitas no documento, ao entrar em produção, ocorria problemas de compatibilidade, dependendo da alteração e de seu tamanho, este problema poderia ser, em muitas vezes, até sério impedindo alguma funcionalidade importante do sistema.

Finalizado os ajustes, essa versão é passada para a equipe de testes fazer a validação, solucionada a falha essa versão é aplicada em produção, e as alterações são repassadas aos usuários para homologação.

Muitas vezes, acontecem problemas urgentes, situações que inesperadamente ocorrem, e que devem ter prioridades na manutenção, uma vez que é pré-estipulado um tempo

para correções de problemas recorrentes, esse prazo é de 48 á 72 horas, normalmente falhas ocasionadas pela incoerência do ambiente de desenvolvimento em relação ao ambiente de produção, ou seja, ao aplicar as alterações essa acabam impactando em outros pontos do sistema, para isso, foi criado um fluxo alternativo para problemas urgentes como demonstrado na FIG. 16 a seguir:

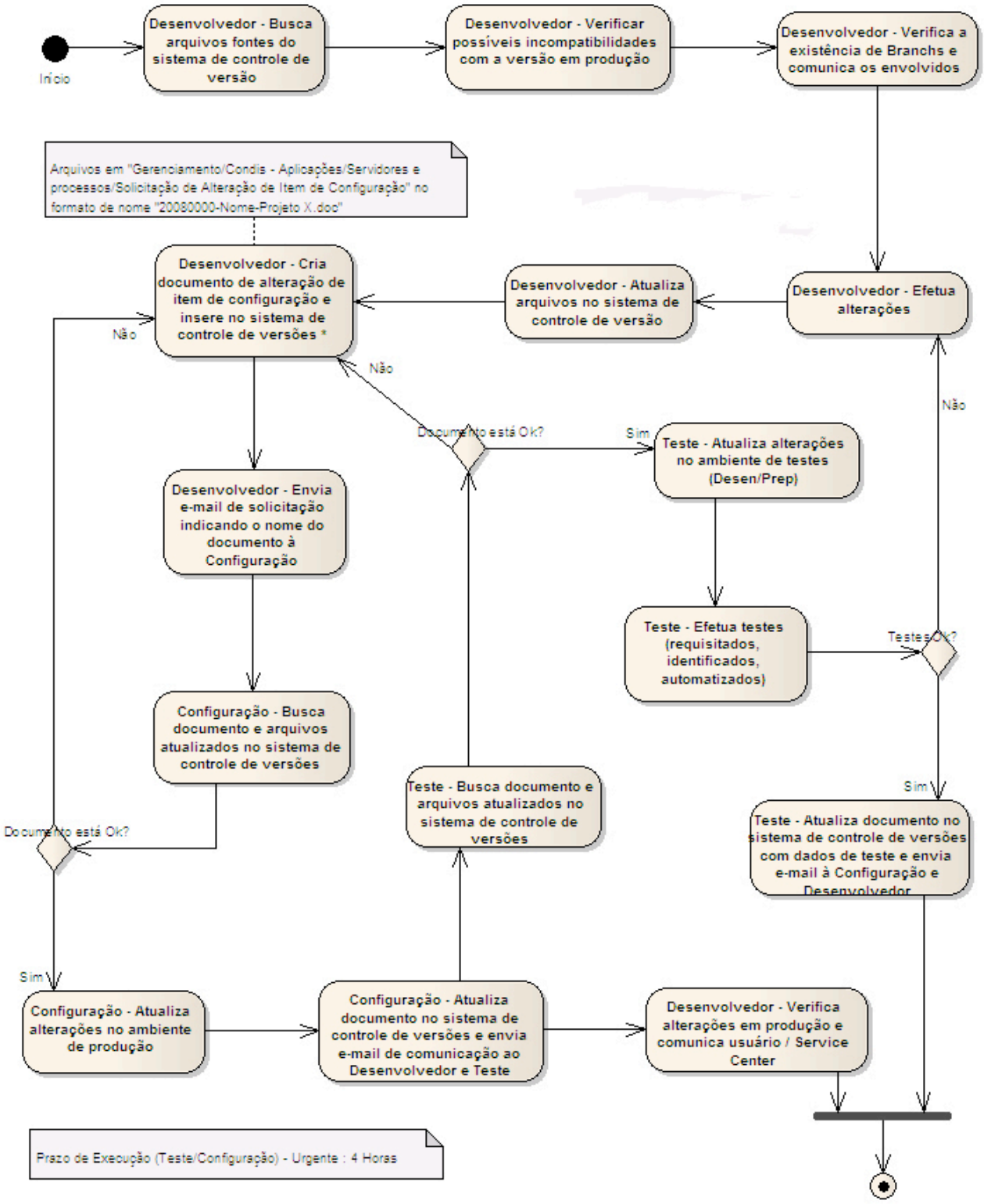

FIGURA 16 – Diagrama de Atividades referente à manutenção de urgência do sistema

Fonte: Dos Autores

Com o crescimento do sistema, praticamente 60% das manutenções se tornaram urgentes, praticamente todas as manutenções começaram a impactar de alguma forma em outros módulos do sistema, o acoplamento alto das aplicações e a alta dependência de outros sistemas fizeram com que a qualidade das aplicações começasse a ser afetada devido ao alto custo das manutenções.

A implantação de novos módulos no projeto, também era um fator agravante, pois tendia a impactar outros módulos, mas como já era esperado, tinha-se um fluxo próprio e um trabalho a mais para isso, pois passou a ser necessário um teste de regressão em outras partes que não eram diretamente direcionadas ao projeto em si, a partir disso foi levantada a necessidade de alteração do processo propriamente dito para um processo de qualidade de maior abrangência.

Com a criação de um fluxo próprio para os projetos, foi constatado que este fluxo era mais completo e apresentava menos problemas que a manutenção em si, pois muitos dos problemas ocasionados pela implantação de módulos novos eram checados pela equipe de testes e não chegava a ser homologado e posteriormente apresentado em produção.

A figura a seguir, FIG. 17 demonstra através do diagrama de atividades, como era a implantação de novos módulos no sistema:

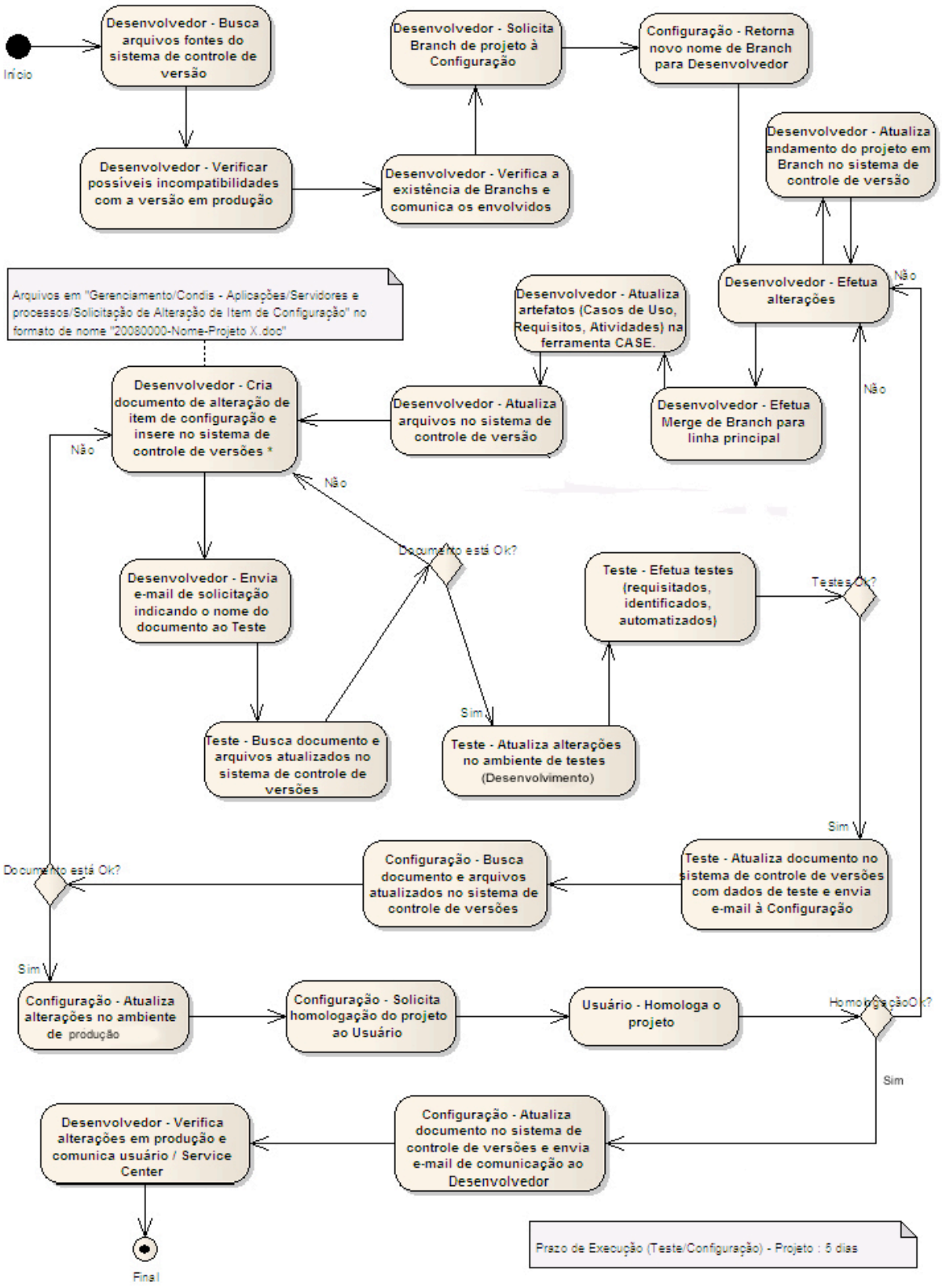

FIGURA 17 – Diagrama de Atividades referente ao desenvolvimento de projetos

Fonte: Dos Autores

4.2 Objetivo

Apresentado os problemas, e a gravidade da dificuldade de manutenção do sistema, foi diagnosticado que era necessário alterar o fluxo de desenvolvimento e sua implantação.

Com isso, a ideia era o versionamento do sistema, uma vez que, para a implantação de módulos e manutenções cotidianas, o sistema não era versionado com isso desde sua implantação ele crescia sem métricas e controle.

Para o versionamento seria necessário a princípio um ambiente próprio de testes, separado do de desenvolvimento, o que chamamos de pré-produção. Este por sua vez, encarregaria mais o gerente de configuração de atualizá-lo, sendo necessário um encarregado para isto, e já serviria como um ambiente de homologação, pois todos os artefatos necessários deveriam esta em perfeito funcionamento para que este venha a ser implantado, sendo um espelho do ambiente de produção.

Com isso, o objetivo passou a ser a criação de um ambiente de homologação, a formalização de um gerente de configurações e a criação da gestão de qualidade, seguindo os modelos de maturidade também expostos nesta monografia, onde objetiva a princípio o gerenciamento do processo, onde anteriormente era caótico.

O número médio de erros, levando em conta os dados mensais, diminuiu 80%, diminuindo com isto o custo de manutenção e aumentando a confiabilidade no software.

4.3 Implantação

A proposta do versionamento do processo de desenvolvimento e sua implantação foi uma das soluções mais viáveis. Uma vez que, o ciclo de manutenção é constante, passa-se a ser um processo mensal, se tornando padronizados todos os estágios do desenvolvimento, testes e configuração.

Foram adotadas algumas novas ferramentas para a otimização dos estágios de desenvolvimento e manutenção, entre elas está o Mantis, um gerenciador de defeitos de interface simples desenvolvido em código aberto, que permite com isso gerenciar os erros, *bugs* e defeitos encontrados no sistema.

Foi implementado um sistema de gerenciamento dos arquivos de configuração, seguindo a facilidade que foi de adaptação dos envolvidos com o projeto de se adaptarem ao Mantis, foi criada uma ferramenta semelhante, onde gerencia os documentos criados pelos desenvolvedores, fazendo com que sejam mais bem gerenciados ao serem implantados em produção, podendo ser gerado relatórios de modificações por períodos de tempo e por desenvolvedores.

Seguindo a ideia de plataformas abertas e de documentações colaborativas, devido ao problema referente ao tamanho do sistema e a dificuldade de documentá-lo, foi proposto a implantação de uma ferramenta *Wiki*,[16] com o intuito de documentar gradativamente todo o sistema, onde o desenvolvedor ao estudar a funcionalidade de uma determinada parte do sistema, ele relatá-lo-ia na ferramenta disponibilizando esta documentação para correções futuras.

O fluxo do processo passando a ser mensal, ou seja, seria disponibilizada uma versão com as correções do sistema até o quinto dia útil de cada mês, com isto, possibilitou a criação de métricas dando maior controle dos serviços dos desenvolvedores ao Gerente de Projetos (Ger. Projetos) cabendo a ele a definição de métricas envolvidas e a gestão de riscos.

Ao ser levantado um problema pelos centros de distribuição, suporte ou pela equipe de qualidade o gerente de projeto passou a avaliar o tipo de problema reportado, cabendo a ele verificar a gravidade do problema classificando-o como urgente ou normal.

---

[16] Software colaborativo que permite a edição coletiva dos documentos usando um sistema que não necessita que o conteúdo tenha que ser revisto antes da sua publicação. (Fonte: Dos Autores)

O processo representado na FIG. 18 exemplifica o processo implantado:

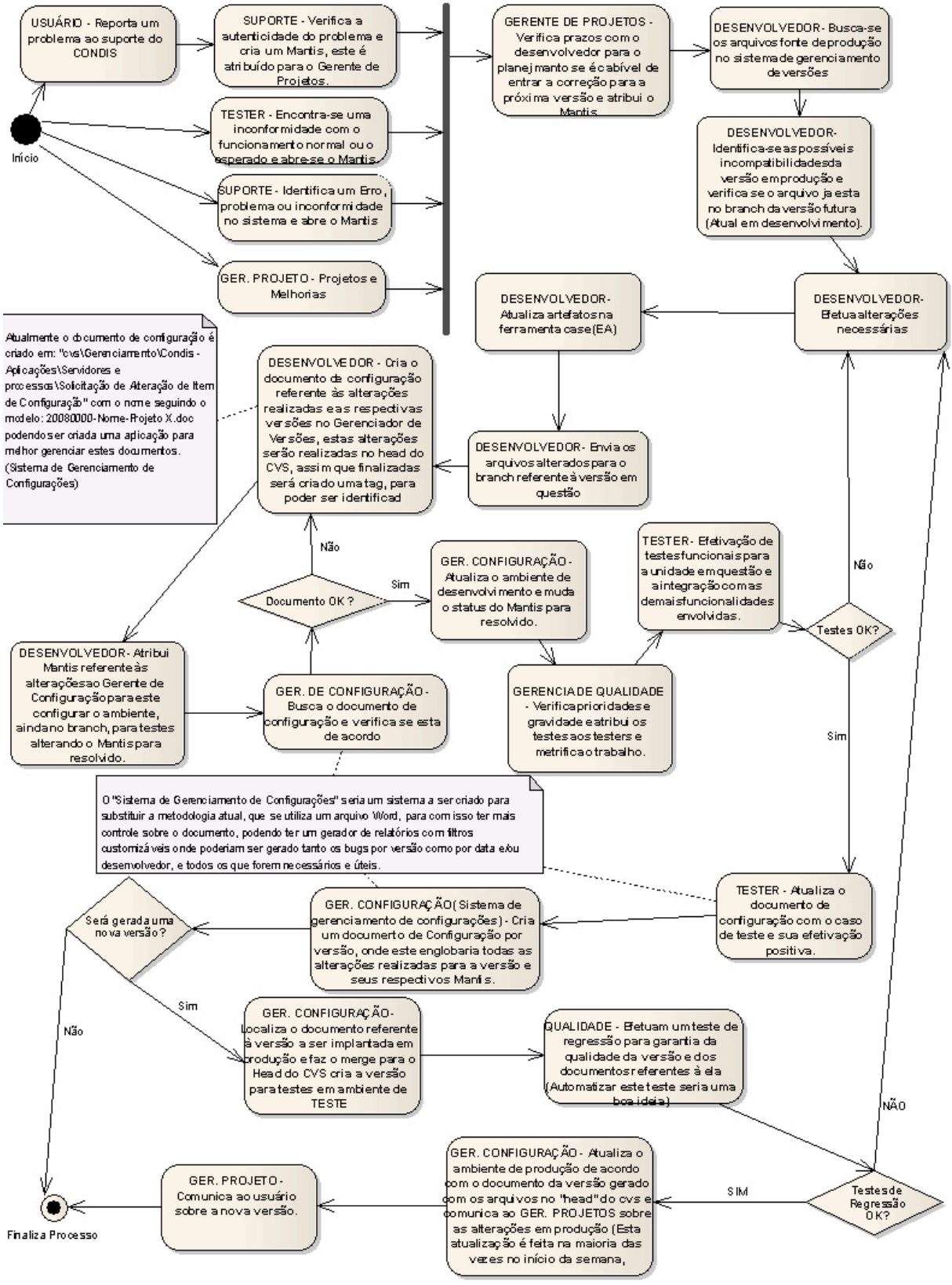

FIGURA 18 – Diagrama de Atividades representando a proposta de versionamento.

Fonte: Dos Autores

O Ger. Projeto, em seu plano normal, recebe uma notificação levantada pelos usuários do sistema e verificada pelo suporte através da ferramenta Mantis reportando o problema a um Desenvolvedor, onde este buscará pelos arquivos fontes problemático e analisará o problema relatado.

O Desenvolvedor verifica na ferramenta *Wiki* se possui documentação a respeito da funcionalidade com problemas, caso não a tenha, cabe a ele documentar o sistema e descrever todas as funções abordadas para correção do problema encontrado e a funcionalidade em questão em seu perfeito funcionamento. Feito isto, ele necessita de criar o documento de configuração com as alterações feitas por ele para que o Gerente de Configurações implante os arquivos alterados ainda em desenvolvimento para que seja testado, ele utilizando o CVS, envia os arquivos testados para o *branch* [17]da versão em questão.

Anteriormente, todos os testes eram feitos em ambiente de desenvolvimento, e posteriormente implantado em produção. Mas foi implantado um servidor dedicado à qualidade, para que seja feito um ambiente de pré-produção com o intuito da diminuição de problemas recorrentes à configuração em produção.

O sistema passou a ser testado duas vezes, aumentando com isso a sua integridade e conseqüentemente a qualidade.

Após passar pela primeira bateria de testes, e esses serem aprovados, normalmente na ultima semana do mês, é implantado a versão no ambiente de pré-produção (homologação) e é feito outro teste, mas um teste de regressão, onde verifica o sistema inteiro, ou grande parte dele, com o intuito de achar defeitos não encontrados anteriormente e possíveis impactos.

Foi proposto também, para esses testes, automatizá-los utilizando a ferramenta descrita nesta monografia, o *Selenium*, mas este ainda não pode ser implantado devido a não capacitação técnica dos envolvidos no processo de teste com relação à ferramenta, mas está sendo estudado um treinamento para esta funcionalidade devido ao tamanho ganho na produtividade.

---

[17] Branch é uma ramificação no desenvolvimento, usada para descrever o processo de divisão dos arquivos de um projeto em linhas de desenvolvimento independentes. Podendo servir para teste de uma nova funcionalidade ou para projetos destinados a um cliente específico.(Fonte: Dos Autores)

Feito os testes de regressão, e aprovado, passa-se para o gerente de configuração implantar a versão em produção e comunicar ao Ger. Projetos a homologação juntamente ao usuário.

Para problemas emergenciais, foi criado processo ilustrado através da FIG. 19.

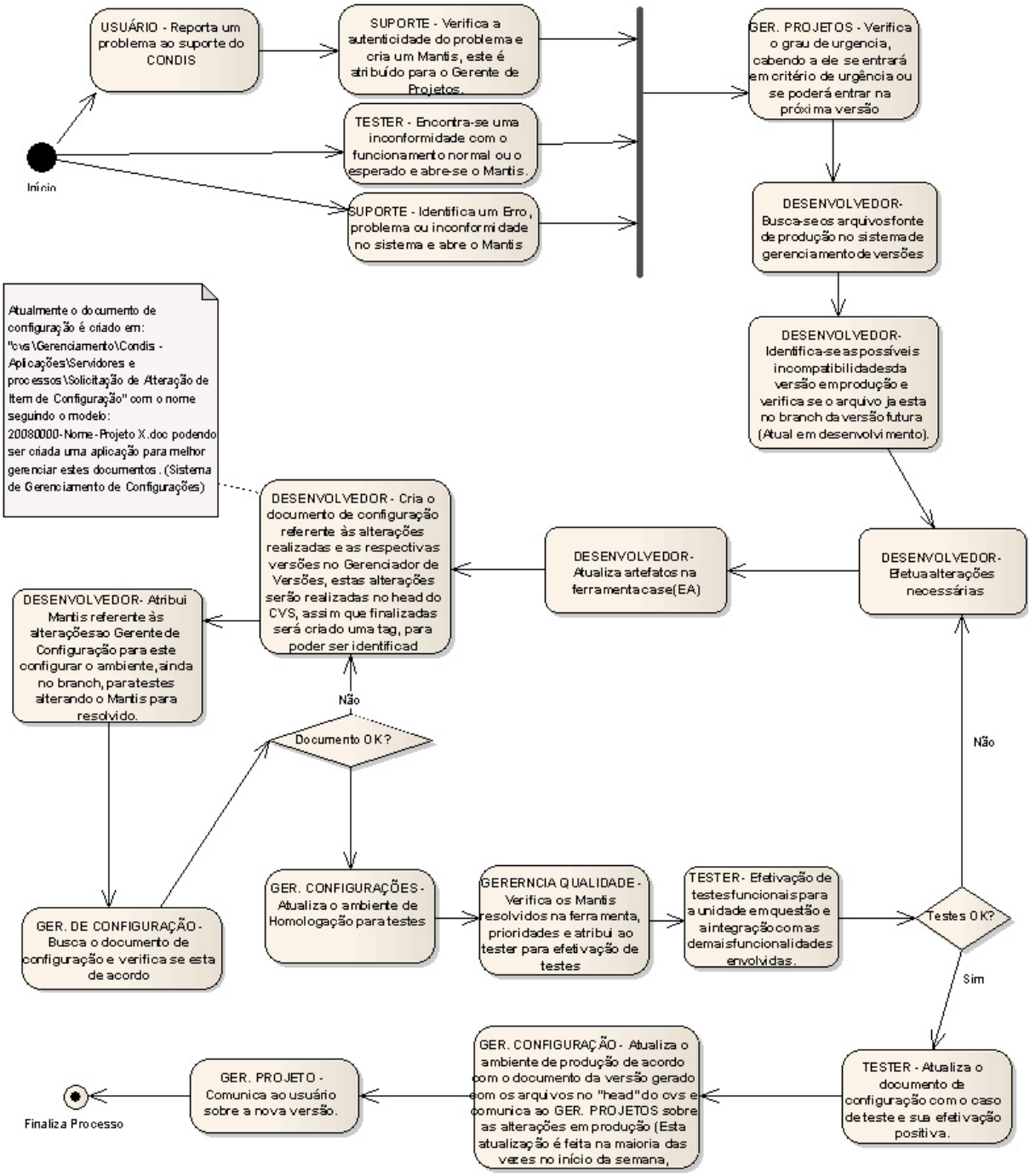

FIGURA 19 – Diagrama de Atividades referente à manutenção de urgência proposta

Fonte: Dos Autores

Foi necessária a criação de um processo para manutenções urgentes, uma vez que problemas encontrados pelos usuários não poderiam esperar pela versão mensal proposto que

entraria em produção a correção em até quatro horas considerando essas urgências como *patchs* [18] da versão atual em produção.

O fluxo não é muito diferente, mas como é de urgência, não é criado um *branch* próprio para esta correção, onde é feito com os próprios códigos em produção, e sua configuração no ambiente de homologação é mais rápida, tornando o ambiente único para testes, mantendo ainda assim a qualidade, uma vez que já é testado no ambiente de homologação antes de entrar em produção, podendo ser reprovado pela qualidade ou não.

Este *patch* ao entrar em produção, era imediatamente comunicado aos *stakeholders* e assim que esses homologassem era considerado o problema corrigido.

4.4 Impacto

O impacto na implantação do processo, como já era esperado, foi grande. Uma vez que os desenvolvedores já estavam acostumados com o antigo processo, passou a ser mais criterioso em algumas fazes do desenvolvimento.

O principal impacto observado, foi com relação a ferramenta de documentação corporativa, pois no início do processo não havia documentação registrada alguma, passando a ser um trabalho do próprio desenvolvedor documentar tudo que ele esta fazendo, o que anteriormente não era feito.

Outra questão encontrada a início foi a dificuldade de adaptação dos *testers* pois passaram a ter que testar o sistema duas vezes, o que no início foi complicado, pois passaram a achar muitos defeitos no software, principalmente nos testes de regressão, que era muito pouco aplicado anteriormente.

Depois do terceiro mês, com a terceira versão implantada, o processo passou a ser mais bem visto pelos envolvidos, o Ger. Projeto conseguiu planejar todo o processo perfeitamente cumprindo todas as fases do processo no seu devido escopo de prazo e qualidade.

4.5 Resultados

[18] *Patch* (literalmente, "remendo") é um programa criado para atualizar ou corrigir um software.

Os resultados encontrados com o processo implantado foram espantosos. A princípio, com a implantação rígida dos processos, muitos dos envolvidos tendo dúvidas chegaram a questionar o problema, mas com as versões posteriores lançadas em produção, passaram a ver o processo como um todo.

Os erros encontrados no início do processo, principalmente em ambiente de homologação com testes de regressão foram diminuindo, o usuário passou a se preocupar com questões de melhoria com o software, e não com qualidade que chegou a ser questionada com o processo anterior.

Os *stakeholders* com a satisfação dos usuários do sistema sentiram satisfeitos com o processo implantado no sistema e pretende expandir o mesmo processo para outros projetos, e falam em certificar o processo implantado a um outro projeto.

5 Conclusão

A qualidade passou a ser um marco no desenvolvimento de software, não somente o resultado final do produto em si é interessante, mas também todo o processo para a sua criação. A qualidade dos padrões adotados, os métodos e motivos pela adoção são de igual relevância.

O desenvolvimento de software com qualidade passa necessariamente pela execução de artefatos que auxiliem na obtenção dessa característica, sendo que a atividade de testes possui grande relevância neste contexto. O processo da padronização do processo demonstrou de extrema importância na criação e manutenção de softwares.

A qualidade depende diretamente de todos os requisitos e de sua clareza para o seu entendimento, o processo em que é envolvido notoriamente trata esses requisitos como um artefato para a sua referencia, e assim que compreendido passa a ser um objetivo o seu atendimento e se concluído com sucesso torna-se um mérito seu desfecho.

Estudamos neste trabalho os principais modelos referentes ao desenvolvimento, com o intuito de entendermos o ciclo de vida de software para compreender e planejar a qualidade de software. Abrangendo o que é o defeito sua prevenção, identificação e correção.

Vimos no estudo de caso, que para manter a qualidade do software, não necessariamente esta relacionada simplesmente ao desenvolvimento e sim com todo a gerencia do projeto, a gestão das configurações e a administração dos testes. Comprovando com ele que o estudo da Engenharia de Software é de grande relevância quando o objetivo é a qualidade.

6 REFERÊNCIAS

ABNT. NBR ISO/IEC 8402 - **Gestão da qualidade e garantia da qualidade – terminologia.** Rio de Janeiro, 1994.

ABNT. NBR ISO 9000 NORMA BRASILEIRA. 2 ed. São Paulo: ABNT, 2005.

ABNT. NBR ISO/IEC 9126 (2003), **Engenharia de software - Qualidade de produto.** Modelo de qualidade.

ABNT. NBR ISO/IEC 12207 – ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. ISO/IEC 12207 – **Tecnologia da Informação – Processos de ciclo de vida de software**. Rio de Janeiro: ABNT, 1996.

BARTIÉ, Alexandre. **Garantia da qualidade de software**: adquirindo maturidade organizacional. Rio de Janeiro: Elsevier, 2002

BOEHM, Barry W. A spiral model of software development and enhancement. ***IEEE Computer,*** p. 61-72, maio. 1988.

BROOKS, Frederick P. Brooks, Jr.: *The Mythical Man-Month —* ***Essays on Software Engineering***, Reading, Massachusetts: Addison-Wesley Publishing Company 1975.

CHRISSIS, M. B.; KONRAD, M.; SHRUM, S. *CMMI:* ***Guidelines for Process Integration and Product Improvement***. Pensilvânia, EUA: SEI Software Engineering Institute Addison-Wesley, 2003

CMMI. **Capability Maturity Model® Integration para Desenvolvimento** (CMMI-DEV), versão 1.2. Tradução: André Villas Boas e José Marcos Gonçalves.

DAHL, O.-J.; DIJKSTRA, E. W. ; HOARE, C. A. R.. Hierarchical program structures. In: **STRUCTURED PROGRAMMING**. Academic Press, London, England, 1972.

DUVALL, Paul M. **Continuous Integration:** Improving Software and Reducing Risk. 6 ed. São Paulo: Addison Wesley, 2003.

ENIAC. 2009. Wikipédia, A Enciclopédia Livre. Último acesso em 03 de novembro de 2009. Disponível em: <http://pt.wikipedia.org/wiki/Eniac>.

GUERRA, Ana Cervigni.; COLOMBO, Regina M. T. **Tecnologia da Informação:** Qualidade de Produto de Software. PBQP Software, 2009.

GUIA GERAL MPS.BR - Melhoria do Processo de Software Brasileiro (versão 1.2) . SOFTEX Disponível em: <http://www.softex.br/mpsbr/_guias/default.asp>. Acesso em: out. 2009.

HARRINGTON, James**. Aperfeiçoando processos empresariais**. Makron Books Editora, São Paulo, 1993.

KOSCIANSKI, André; SOARES Santos, Michel dos. ***Qualidade de Software***. Novatec, 2006.

NAIK, Kshirasagar.; TRIPATHY, Priyadarshi. **Software Testing and Quality Assurance**: Theory and Pratice. New Jersey: Jonh Wiley & Sons Inc., 2008.

PÁDUA FILHO, Wilson de. **Engenharia de Software**: fundamentos, métodos e padrões. 2 ed. Rio de Janeiro: LTC, 2003.

PAULK, M., Curtis, B., CHRISS, M. e WEBER, C., 1995, **The Capability Maturity Model*:* Guidelines for Improving the Software Process**, Addison-Wesley.

PETERS, James F.; Pedrycz, Wiltold. **Engenharia de Software**: Teoria e Prática. Rio de Janeiro: Campus, 2001.

PMBOK, *Um guia do conjunto de conhecimentos em gerenciamento de projetos*. Project Management Institute (PMI). 3 ed. 2004.

PRESSMAN, Roger S. **Engenharia de Software**. 6 ed. Rio de Janeiro: McGraw-Hill, 2002.

ROCHA, A. R. C.; MALDONADO, J. C.; WEBER, K. C. **Qualidade dos produtos de software:** teoria e prática. São Paulo: Prentice Hall, 2001.

SOMMERVILLE, Ian. **Engenharia de Software**. 6 ed. São Paulo: Addison Wesley, 2003.

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