CASSIANO CONSTANTINO CASAGRANDE

MINIMIZANDO PROBLEMAS DE ANÁLISE NO PROCESSO DE SOFTWARE, APLICADO AO MÉTODO RUP.

Florianópolis (SC)
2008

CASSIANO CONSTANTINO CASAGRANDE

MINIMIZANDO PROBLEMAS DE ANÁLISE NO PROCESSO DE SOFTWARE, APLICADO AO MÉTODO RUP.

Monografia apresentada à Universidade Federal de Santa Catarina como requisito parcial à obtenção do diploma de Bacharel em Sistemas de Informação.

Orientador: Marcelo de Andrade Machado

Florianópolis (SC)

2008

CASSIANO CONSTANTINO CASAGRANDE

MINIMIZANDO PROBLEMAS DE ANÁLISE NO PROCESSO DE SOFTWARE, APLICADO AO MÉTODO RUP.

Essa Monografia foi julgada adequada para a obtenção do título de Bacharel em Sistemas de Informação e aprovada pelo curso de Sistemas de Informação da Universidade Federal de Santa Catarina.

Florianópolis, dia de junho de 2008.

Banca Examinadora:

_____
Orientador: Marcelo de Andrade Machado

_____
Professor Doutor Ricardo Pereira da Silva

_____
Professor Doutor José Leomar Todesco

RESUMO

Neste trabalho, foi realizado um estudo dos problemas encontrados durante a aplicação do método estruturado RUP em uma organização que trabalha no sistema de fábrica de software. O escopo do estudo foi restrito aos problemas encontrados na disciplina de Implementação com origem na disciplina de Análise e Design. Isso porque os maiores impactos nas atividades realizadas pela fábrica de software eram provocados por problemas com essas características. Também foi dada maior ênfase e foco para o estudo dos requisitos nesse contexto, sem abranger a parte de arquitetura, também integrante da disciplina de Análise e Design. Requisitos incompletos e/ou inconsistentes, dificuldade para conseguir um entendimento comum dos requisitos, diagramas com defeitos e/ou inconsistentes são exemplos de problemas que se encaixam no escopo do trabalho. O método estruturado RUP prega rotinas bem definidas de revisões para cada fase do processo de software, sendo que uma diretriz inteira e várias funções do método estruturado são dedicadas às revisões. Com a aplicação das revisões, espera-se sempre a diminuição dos problemas e a prevenção de novos problemas em um projeto. A escolha da aplicação da diretriz de revisão do RUP como solução ao problema do trabalho se fortaleceu logo no início dos estudos, frente a opções mais sofisticadas do próprio RUP, como a aplicação de um rigoroso controle de qualidade. Isso porque as revisões dos produtos de trabalho deveriam fazer parte das atividades básicas do RUP, mas não estavam sendo aplicadas. A diretriz começou a ser aplicada e dificuldades de comunicação e padronização precisaram ser vencidas para que os resultados fossem positivos. Com isso os impactos no projeto foram bem identificados e a importância de se revisar as atividades dentro de cada fase do processo de software ficou evidente.

PALAVRAS-CHAVE: RUP, análise, design e revisão.

ABSTRACT

In this work, a study was conducted on problems encountered during the application of a structured method RUP in an organization that works in a software factory system. The scope of the study was restricted to the problems encountered in the implementation discipline with origin in the Analysis and Design discipline. That's because the largest impacts on activities carried out by the software factory were caused by problems with these characteristics. It was also given greater emphasis and focus on the study of the requirements in that context, without covering the architecture, also part of the Analysis and Design discipline. Incomplete and / or inconsistent requirements, difficulty to achieve a common understanding of the requirements, diagrams with defects and / or inconsistencies are examples of problems that fit in the work scope. The structured method RUP suggests well-defined routines of revision for each phase of the software process, which an entire guideline and a lot of the structured method functions are dedicated to the revisions. With the revisions implementation, problems decrease and preventing of new problems are always expected in a project. The choice of the RUP guideline revision implementation as a solution to the work problem was strengthened in the beginning of the studies, as opposed to more sophisticated options of RUP, as the implementation of a more rigorous quality control. That's because the work products revision should be part of the RUP basic activities, but they were not being applied. The guideline began to be applied and communication difficulties and standardization needed to be overcome so the results were positive. With that, the impacts on the project were well identified and the activities review importance within each software process phase were evident.

KEYWORDS: RUP, analysis, design and review.

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

1.1 Apresentação

Nos últimos anos, temos vivenciado a crescente adoção de abordagens mais metódicas no desenvolvimento de softwares pelas organizações, principalmente em projetos de grande porte. Dessa forma, surgiram métodos estruturados para o desenvolvimento de software como o *Rational Unified Process* – ou apenas RUP – e o OPENUP.

Por uma questão de padronização, deste ponto em diante, será adotado o termo "método estruturado" para este projeto, termo este que também é adotado por Sommerville (2000). Também para este trabalho, será considerada organização como sendo qualquer grupo de indivíduos associados com um objetivo comum.

Para promover a simplificação, o presente trabalho tratará o termo "processo de desenvolvimento de software" como "processo de software". Para Sommerville (2000), "Os processos de software são complexos e, como todos os processos intelectuais, dependem de julgamento humano." Para lidar com a complexidade envolvida nos processo de software, as organizações são levadas ao uso de métodos estruturados.

Por meio da adoção de um método estruturado para o processo de software, as organizações buscam garantir um mínimo de controle e qualidade. Dessa forma, os custos são reduzidos e é garantido um mínimo de qualidade ao produto de software sem depender exclusivamente de esforço e talento dos envolvidos no processo de software. Isso se deve à relação entre o processo de software e a qualidade do produto de software colocada por Sommerville (2000):

> Para sistemas muito grandes, compostos de subsistemas separados, desenvolvidos por diferentes equipes, o determinante principal da qualidade do produto é o processo de software.

Cada método estruturado possui seus pontos fortes e fracos, cada qual voltado para determinadas necessidades e situações. Os métodos estruturados podem ser adotados na sua forma de uso padrão, indicada por seus criadores, ou personalizados conforme as características da organização que o adotar. Kroll e Kruchten (2003) relatam como o método estruturado *Rational Unified Process*, o RUP, pode ser configurado para se enquadrar à realidade de projetos de grande, médio e até pequeno porte por possuir as características de ser customizável, flexível e configurável.

Devido às experiências profissionais do autor deste trabalho com o RUP e ao fato desse método estruturado ter sido o mais comentado ao longo da elaboração de sua elaboração, o presente estudo o terá como foco. Segundo seus criadores, a empresa multinacional IBM, o RUP diz respeito a um processo de software bem sucedido. Esse método estruturado é baseado em iterações e organizado em partes especializadas chamadas de disciplinas (IBM, 2006). O RUP é apresentado mais a fundo no capítulo dois.

Mas apesar do uso de métodos estruturados, a documentação do RUP (2006) também fala que a influência de fatores como custo, tempo, cronograma, qualidade pessoal e tecnologias pode variar em um projeto de acordo com o seu tamanho e tipo. Em um processo de software, existe sempre a possibilidade de surgirem diversos fatores problemáticos que podem prejudicar o produto de software. Quando esses fatores não são devidamente previstos e tratados, podem comprometer o cumprimento das metas a que o projeto de software se propõe. Sendo assim, não é raro que os

envolvidos em um processo de software tenham dúvidas sobre como realizar as atividades que compõem um método estruturado e problemas que dificultem a realização dessas atividades. Esses fatores problemáticos podem ser relevantes a ponto de desgastar as pessoas e atingir, de forma prejudicial, os prazos de um projeto.

Este trabalho irá avaliar o uso dos dados obtidos com as dúvidas dos envolvidos na disciplina de Implementação do RUP para a melhoria da disciplina de Análise e Design, também do RUP. Somente dúvidas provenientes da disciplina de Análise e Design serão avaliadas neste estudo, cujos dados virão de um projeto de software de grande porte de uma empresa nacional de TV a cabo. Este projeto de software está aplicado ao método estruturado RUP e será usado como estudo de caso para este trabalho. Com o surgimento e a coleta dos referentes às dúvidas da disciplina de Análise e Design, serão sugeridas melhorias na customização e configuração dessa disciplina e na comunicação dos envolvidos, se for o caso, a fim de melhorar o processo de software e minimizar possíveis prejuízos na qualidade do produto de software.

1.2 MOTIVAÇÃO

Não é raro encontrar nas organizações projetos de software com cronogramas atrasados e equipes desgastadas. Problemas gerados nas disciplinas de Análise e Design podem contribuir para compor esse cenário. Muitos impactos na qualidade e no cronograma nas atividades da disciplina Implementação podem ter origem na disciplina de Análise e Design, gerando atrasos e servindo como justificativas para atitudes como fugas dos padrões, definições e documentos do método estruturado RUP.

1.3 OBJETIVOS GERAIS

O objetivo deste trabalho é conseguir reduzir problemas na disciplina de Implementação, provenientes da disciplina de Análise e Design em projetos que utilizarão o método estruturado RUP. Melhorando, assim, possíveis prejuízos na qualidade do processo de software.

1.4 OBJETIVOS ESPECÍFICOS

a) levantar os problemas da disciplina de Implementação provenientes da disciplina de Análise e Design no estudo de caso;

b) classificar os problemas relacionados;

c) buscar uma forma de solucionar os problemas levantados;

d) aplicar no estudo de caso a solução escolhida e

e) avaliar se o estudo de caso obteve a diminuição esperada dos problemas.

1.5 JUSTIFICATIVA

Em um levantamento feito antes do início deste trabalho, em 63 implementações, foi constatado que cerca de 87,6% das implementações de funcionalidades para o software não eram finalizadas no tempo estimado. Gerando impactos diretos no cumprimento do cronograma do projeto.

Problemas relacionados à Análise e Design podem tomar um tempo dispendioso dos envolvidos para sua solução. Durante o processo de software podem surgir problemas provenientes da disciplina de Análise e Design que impactem no bom andamento da disciplina de Implementação, ambas disciplinas do RUP. Leffingwell (2003) constatou em seus trabalhos que entre 40% e 60% de todos os problemas encontrados em um projeto de software são causados por falhas ocorridas na fase de levantamento de requisitos.

Conceber uma alternativa para solucionar ou remediar os problemas em qualquer projeto que esteja utilizando o método estruturado RUP é a justificativa deste trabalho.

1.6 ORGANIZAÇÃO DOS CAPÍTULOS

Este trabalho foi divido em quatro partes: primeiro, a revisão literária, demonstrando os principais conceitos necessários para o trabalho, em segundo, uma revisão literária mais específica para a solução proposta ao problema. Na seqüência, o estudo de caso em si e, por fim, a análise e os resultados obtidos com a pesquisa.

Nos capítulos 2, 3 e 4 serão vistos os conceitos base e será feita a revisão

bibliográfica do trabalho. O capítulo 5 compreende uma revisão literária mais aprofundada e voltada à solução do problema e o capítulo 6 abrange os relatos do estudo de caso. Finalmente, no capítulo 7, é apresentada a análise e os resultados obtidos com a pesquisa.

2 CONCEITOS BASE

2.1 METODOLOGIAS E MÉTODOS

Para o dicionário Michaelis (2007), metodologia é o estudo científico dos métodos. Alguns autores preferem tratar metodologia e método como sinônimos, já outros tratam metodologia como sendo um modelo de princípios que guia uma gama de métodos. Para este trabalho, metodologia será considerada como sendo um modelo de princípios abstratos. Por exemplo: o termo “metodologia orientada a objetos” é usado para classificar métodos que seguem os princípios da orientação a objetos. O RUP é exemplo desse método, pois segue os princípios da orientação a objetos.

Para Sommerville (2000), na engenharia de software um método é uma abordagem estruturada voltada para o processo de software, com o objetivo de melhorar a produção de software de qualidade com baixo custo. Para facilitar o entendimento dos conceitos de metodologia e método, seguem-se dois exemplos de métodos que seguem a metodologia orientada a objetos:

a) método RUP – fornece uma forma sistemática para se obter reais vantagens do uso desse modelo;

b) método Scrum – método ágil, que tem como objetivo, segundo Schwaber e Beedle (2002), definir um processo de software que seja focado nas pessoas e indicado a ambientes em que os requisitos surgem e mudam rapidamente.

2.2 A ENGENHARIA DE SOFTWARE

Ao final da década de 1960, com a quase inexistência da engenharia de software, o mundo sofreu diante de muitos problemas referentes ao desenvolvimento e à manutenção de software. Para tentar resolver esta crise – período que ficou conhecido como "crise do software" (NAUR; RANDELL, 1969) – foi criada a engenharia de software, uma tentativa de trazer das engenharias existentes toda a organização usada para resolver problemas de sistemas complexos em outras áreas do conhecimento.

A primeira definição para engenharia de software foi apresentada por Friedrich Ludwig Bauer, na *Nato Conference on Software Engineering*, em 1968*,* que a definiu como

> a criação e a utilização de sólidos princípios de engenharia a fim de obter software de maneira econômica, que seja confiável e que trabalhe eficientemente em máquinas reais (BAUER, 1968).

Com o tempo, outras definições foram sendo propostas. Pressman (2002) cita que a engenharia de software abrange um processo, um conjunto de métodos e ferramentas que devem apoiar e garantir a organização e a qualidade do software. A engenharia de software envolve o uso de metodologias abstratas para que se possa analisar, projetar (modelar, design, uso da UML), implementar (codificar), testar e manter os softwares. A engenharia de software também contém formas de se planejar e gerenciar um processo de software.

Na tentativa de reunir uma referência sobre quais assuntos são considerados próprios pela comunidade tecnológica a ponto de serem englobados na engenharia de software, a *IEEE Computer Society* criou o SWEBOK – *Guide to the Software Engineering Body of Knowledge* Segundo o documento, a engenharia de software está organizada em disciplinas fundamentais:

a) Análise de requisitos de Software;

b) Projeto (Design) de Software;

c) Construção de Software;

d) Teste de Software;

e) Manutenção de Software;

f) Gerência de Configuração de Software;

g) Gerência de Engenharia de Software;

h) Processos de Engenharia de Software;

i) Ferramentas e Métodos de Engenharia de Software e

j) Qualidade de Software.

2.3 PROCESSO DE SOFTWARE

> Um processo de desenvolvimento de software é um conjunto de ferramentas, métodos, documentos e práticas usadas para construir um produto de software (HUMPHREY, 1990)

Segundo Sommerville (2000), processo de software é o conjunto de atividades que tem como objetivo desenvolver ou melhorar o produto de software. Dentro de um

processo as mesmas atividades podem ter variação em seus prazos e resultados se pertencerem a processos de software diferentes. O autor explica que diferentes organizações podem utilizar diferentes processos de software para produzir o mesmo tipo de produto. Todavia, alguns processos de software são mais indicados do que outros para determinados tipos de aplicação. É importante escolher o processo certo, pois se um processo de software inadequado for utilizado, isso provavelmente reduzirá a qualidade do produto de software desenvolvido.

Algumas atividades fundamentais estão inclusas nos diferentes modelos de processo de software existentes hoje. Sommerville (2000) cita que todos os processos de softwares são formados por quatro atividades fundamentais:

a) especificação de software: em que se definem as funcionalidades e restrições do software;

b) desenvolvimento de software: em que o software é desenvolvido para atender às especificações;

c) validação do software: em que o software é testado para confirmar se atende às necessidades do cliente e

d) evolução de software: em que acontecem as modificações necessárias do software.

2.4 MODELOS DE PROCESSOS DE SOFTWARE

Sommerville (2000) define alguns modelos de processo de software de forma genérica (que também são chamados de paradigmas de processo).

> Esses modelos genéricos não são descrições definitivas de processos de software. Em vez disso, são abstrações úteis, que podem ser utilizadas para explicar diferentes abordagens do processo de software.

Os modelos de processo de software definidos são:

a) Modelo em cascata: esse modelo engloba as atividades fundamentais do processo de software e as divide como fases separadas e seqüenciais do processo, como a especificação de requisitos, o projeto de software, a implementação e os testes. Uma fase seguinte não pode iniciar até que sua antecessora tenha terminado;

b) Desenvolvimento evolucionário: essa abordagem intercala as atividades fundamentais de especificação, desenvolvimento e validação. Uma implementação inicial é rapidamente desenvolvida para que se tenha um *feedback* do cliente e, dessa forma, produzir um sistema que atenda às suas necessidades;

c) Desenvolvimento formal de sistemas: essa abordagem se baseia na produção de uma especificação formal matemática do sistema e na transformação dessa especificação, utilizando-se métodos matemáticos para construir um programa;

d) Desenvolvimento orientado ao reuso: essa abordagem faz uso de componentes reutilizáveis na maior parte do programa que está sendo

desenvolvido. O processo software se concentra na integração desses componentes em um sistema.

Nas seções anteriores foram apresentados alguns conceitos e definições iniciais. Como explanado, a engenharia de software estrutura as fases pelas quais o processo de software passa. Uma destas fases, a Análise de requisitos de Software, acabou gerando outra linha de estudo aprofundado chamada engenharia de requisitos. Como o problema estudado, diminuir os problemas provenientes da especificação do software, tem suas causas restringidas ao contexto da engenharia de requisitos, ela será estudada mais a fundo no capítulo a seguir.

3 ENGENHARIA DE REQUISITOS

> As descrições das funções e das restrições são os requisitos para o sistema; e o processo de descobrir, analisar, documentar, verificar essas funções e restrições é chamado de engenharia de requisitos (SOMMERVILLE, 2000).

3.1 O QUE SÃO REQUISITOS DE SOFTWARE?

Segundo Sommerville (2000), quase sempre o termo “requisito” é utilizado pelas organizações de duas formas, dependendo do tipo de leitor que o documento de requisitos terá. Na forma de uma descrição alto nível, abstrata das restrições e funções que um software terá, ou na forma de uma descrição detalhada das restrições ou funções do software.

Para Thayer e Dorfman (1993), um requisito é:

a) uma condição ou capacidade necessária para o usuário resolver um problema ou alcançar um objetivo;

b) uma condição ou capacidade que deve ser encontrada ou possuída por um sistema ou componente do sistema para satisfazer um contrato, padrão, especificação ou outro documento imposto formalmente ou

c) uma representação documentada de uma condição ou capacidade como em (a) ou (b).

Muitos dos problemas dentro da engenharia de software provêm da má definição e separação dos níveis de requisitos, explica Sommerville (2000). Em linhas gerais, para auxiliar na solução desse tipo de problema o autor propõe o uso dos

termos “requisitos de usuário” para descrições de alto nível e “requisitos de sistema”, também conhecido como “especificação funcional”, para descrições detalhadas das funções e restrições de um software. Também pode ser definida uma descrição ainda mais detalhada, como uma “especificação de projeto de software”. Este trabalho irá adotar, deste ponto em diante, os termos “requisitos de usuário” e “requisitos de sistema” como forma de padronização.

3.2 REQUISITOS FUNCIONAIS E NÃO-FUNCIONAIS

Westfall (2005, tradução livre) classifica os requisitos de um projeto como funcionais, não-funcionais e de domínio, definindo-os respectivamnte do seguinte modo:

a) O que o software deve fazer para adicionar valor aos *stakeholders:* definem a capacidade do produto de software;

b) O que o software deve ser para adicionar valor aos *stakeholders*: requisitos não-funcionais que definem as características, propriedades ou qualidades que o produto de software deve possuir, além do nível de qualidade que o produto de software irá desempenhar em suas funções;

c) Quais as restrições sobre as escolhas que os desenvolvedores fazem na implementação do produto de software: a definição da interface externa, e outras amarrações definem essas limitações.

Já Sommerville (2000) define requisitos funcionais e não-funcionais da seguinte forma:

a) Requisitos funcionais: são definições de funções que o software deve ter,

como deve reagir as suas entradas e seu comportamento em determinadas situações. Podem também definir o que o software não deve fazer. Exemplos: efetuar login e efetuar venda;

b) Requisitos não-funcionais: são restrições sobre os serviços ou as funções oferecidas pelo software. Exemplos: tempo de retorno após falha e número de acessos;

c) Requisitos de domínio: são derivados do domínio da aplicação do software, em vez de serem obtidos a partir de novas necessidades específicas dos usuários do sistema. Exemplo: a média de um aluno é calculada pela fórmula: (nota1 + nota2) / 2.

3.3 REQUISITOS DE USUÁRIOS

Especifica apenas o comportamento externo do software, podendo utilizar-se de linguagem natural para escrever os requisitos. Para Sommerville (2000), os problemas que podem aparecer com o uso de linguagem natural são:

a) falta de clareza: um documento de requisitos pode ficar confuso;

b) confusão de requisitos: dificuldade de definição dos requisitos em requisitos funcionais e não-funcionais e

c) fusão de requisitos: requisitos diferentes podem ser considerados iguais.

3.4 REQUISITOS DE SISTEMA

Descrevem detalhes das funções e restrições dos requisitos do usuário de um software. Pode utilizar alguns tipos de modelos para sua especificação, como, por exemplo, um modelo de objetos ou um modelo de fluxo de dados. Para este trabalho, o foco será dado aos modelos de objetos. O uso de linguagem natural para descrever esses requisitos também é comum para os requisitos de sistema e provoca outros problemas, segundo Sommerville (2000):

a) compreensão da linguagem: o entendimento da linguagem natural depende do uso dos mesmos conceitos e palavras;

b) requisitos em linguagem natural são muito flexíveis: o leitor pode ter que decidir quando os requisitos são os mesmos e quando são diferentes e

c) não existe meio fácil de padronização dos requisitos: para se saber o impacto de mudanças, pode ser necessário examinar cada requisito.

Dentre as alternativas para a linguagem natural existe a especificação em linguagem estruturada, em que podem ser limitadas as terminologias utilizadas e permitir o uso de *templates* para os documentos.

3.5 PROCESSO DE ENGENHARIA DE REQUISITOS

Segundo Booch (1994), a engenharia de requisitos é um processo de aplicação de um método estruturado, o qual é muito útil para o processo de engenharia de requisitos. Sommerville (2000) cita a importância dos métodos estruturados para a engenharia de software. Se dentro de uma organização todo processo de software não

for devidamente documentado, estruturado e divulgado entre os desenvolvedores é possível que a qualidade de todo processo dependa exclusivamente da experiência e formação das pessoas envolvidas. Embora sua importância seja reconhecida, o autor também cita que os métodos estruturados não fornecem apoio efetivo aos estágios iniciais do processo de engenharia de requisitos. Mesmo assim, a importância dos métodos estruturados continua sendo grande, pois eles fornecem o suporte para que haja qualidade no processo de engenharia de requisitos.

> Consegue-se uma economia de custos de até 200:1 encontrando erros nas fases de requisitos comparado com erros encontrados na fase de manutenção no de ciclo de vida do software (LEFFINGWELL, 2003, tradução livre).

Para Wiegers (1999) e BlasChek (2002), o processo de engenharia de requisitos é composto das seguintes atividades:

a) levantamento: ou elicitação de requisitos é o nome usualmente destinado às atividades voltadas para descobrir (identificar, extrair, deduzir, evocar, obter) os requisitos de um sistema por meio de entrevistas com os interessados pelo software, de documentos do software existente, da análise do domínio do problema ou de estudos de mercado;

b) análise: na análise os requisitos elicitados são compreendidos e revisados por todos os seus interessados. Nessa atividade, surgem muitos conflitos, sendo comum a necessidade de negociação para que os requisitos sejam aceitos por todos;

c) especificação: os requisitos devem ser documentados com um nível de detalhamento adequado, produzindo a especificação de requisitos de software.

Pode ser utilizada linguagem natural ou diagramas, como os propostos pela UML;

d) validação: após terem sido documentados, é necessário que os requisitos sejam cuidadosamente validados, principalmente quanto à consistência e à completude. Essa atividade visa à identificação de problemas nos requisitos, antes do início da construção, e é caracterizada pelo fato de que a correção de algum erro nessa fase possui um custo muito inferior do que a correção de um erro nas fases mais adiantadas do processo de software.

3.6 LEVANTAMENTO E ANÁLISE DE REQUISITOS

Nas atividades de levantamento e análise de requisitos, os membros da equipe de desenvolvimento interagem com o cliente e os usuários em busca de informações do domínio do software. Segundo Sommerville (2000), o termo *stakhoder* é utilizado para se referir a qualquer pessoa que se envolver com o trabalho dos requisitos de sistema e será utilizado a partir deste ponto neste trabalho. A análise de requisitos é considerada uma área do conhecimento da engenharia de software pela SWEBOK, sendo um processo que engloba todas as atividades para a produção e manutenção de um documento de requisitos. Sommerville (2000) cita as atividades envolvidas no processo de levantamento e análise como sendo:

a) compreensão do domínio: o analista deve compreender plenamente o domínio no qual a organização e o projeto estão. Dessa forma, a comunicação entre os colaboradores e os usuários será mais objetiva;

b) coleta de requisitos: é o processo de interagir com os *stakeholders* do sistema para descobrir seus requisitos;

c) classificação: é a atividade de organizar os grupos de requisitos de forma coerente;

d) resolução de conflitos: quando vários *stakeholders* forem envolvidos, os requisitos apresentarão conflitos. Essa atividade ocupa-se de encontrar e solucionar esses conflitos;

e) definição das prioridades: envolve interação com os *stakeholders* para descobrir os requisitos mais importantes e

e) verificação de requisitos: essa verificação é feita para se descobrir se os requisitos estão completos e consistentes.

A grande complexidade da engenharia de requisitos é uma das dificuldades que acaba atrapalhando o bom andamento de um desenvolviemnto de software. Para este trabalho a engenharia de requisitos é o alvo de estudos, por ser parte de sua causa do problema.

Alguns métodos estruturados possuem em seus processos boas práticas e diretrizes que podem auxiliar no controle da complexidade envolvida em todo o processo de desenvolvimento de software. No próximo capítulo será apresentado o processo estruturado RUP, que é o processo utilizado neste trabalho.

4 *Rational Unified Process* (RUP)

4.1 O que é RUP?

O *Rational Unified Process* (RUP) é um método estruturado e concebido pela *Rational Software Corporation*. O RUP (IBM, 2006) diz respeito a um processo de software bem-sucedido, sendo um conjunto de práticas coletadas de engenharia de software que são continuamente aprimoradas para refletirem alterações nas boas práticas do segmento de mercado. Para Sommerville (2000),

> Esses 'métodos', na verdade, são notações padronizadas e incorporações das boas práticas. Seguindo esses métodos e aplicando as diretrizes, um projeto razoável deve surgir como resultado.

Esse documento cita também três elementos centrais que definem o RUP:

a) um conjunto subjacente de filosofias e princípios para um processo de software bem-sucedido;

b) uma estrutura de conteúdo de método reutilizável e blocos de construção de processo e

c) um meta-modelo de arquitetura de método unificado.

O RUP enfatiza a adoção de boas práticas consagradas do desenvolvimento de software moderno como uma forma de reduzir o risco inerente ao desenvolvimento de um novo software. Para especificar e documentar seus produtos de trabalho, o RUP faz grande uso da Linguagem de Modelagem Unificada (UML). Larman (2004) define a UML como sendo uma linguagem para especificar, visualizar, construir e documentar os artefatos de sistemas de software, bem como para modelar negócios e outros sistemas

que não sejam de software.

A documentação do RUP (IBM, 2006) define de forma simplificada as boas práticas empregadas pelo método:

a) desenvolver iterativamente;

b) gerenciar requisitos;

c) utilizar arquiteturas com base no componente;

d) modelar visualmente;

e) verificar continuamente a qualidade e

f) controlar alterações.

Essas boas práticas são elaboradas nas definições de:

a) funções, conjuntos de tarefas executadas e artefatos elaborados;

b) disciplinas, áreas de enfoque de esforço de engenharia de software como Requisitos, Análise e Design, Implementação e Teste;

c) tarefas, definições da forma que os artefatos são produzidos e avaliados;

d) artefatos, os produtos de trabalho utilizados, produzidos ou modificados no desempenho das tarefas.

> Os métodos estruturados foram aplicados com sucesso em muitos projetos de grande porte. Eles podem proporcionar significativas reduções de custos, porque utilizam notações padronizadas e asseguram a produção de documentação-padrão para o projeto (SOMMERVILLE, 2000).

4.2 FASES DO RUP

O RUP tem sua estrutura base dividida em partes, as quais são chamadas de fases. Assim como quase todos os métodos estruturados de desenvolvimento, as fases do RUP são baseadas nas quatro atividades fundamentais do processo de software. São apresentadas aqui, de uma forma geral, as fases a serem percorridas no decorrer do ciclo de vida de um projeto que faz uso do RUP. A documentação do RUP (IBM, 2006) mostra a divisão das suas quatro fases:

a) Iniciação: a meta dominante da fase de iniciação é atingir o consenso entre todos os investidores sobre os objetivos do ciclo de vida do projeto. A fase de iniciação tem muita importância, principalmente para novos esforços de desenvolvimento, os quais compreendem muitos riscos de negócios e de requisitos que devem ser tratados para que o projeto possa prosseguir;

b) Elaboração: a meta da fase de elaboração é criar a base para a arquitetura do sistema a fim de fornecer uma base estável para o esforço da fase de construção. A arquitetura desenvolve-se a partir de um exame dos requisitos mais significativos (aqueles que têm grande impacto na arquitetura do sistema) e de uma avaliação de risco. A estabilidade da arquitetura é avaliada por meio de um ou mais protótipos de arquitetura;

c) Construção: a meta da fase de construção é esclarecer os requisitos restantes e concluir o desenvolvimento do sistema com base na arquitetura definida e

d) Transição: o foco da fase de transição é assegurar que o software esteja disponível para uso dos seus usuários. A fase de transição pode atravessar

várias iterações, inclui testes no produto e fazer ajustes pequenos, baseados no *feedback* do usuário, como preparação antes de uma nova versão.

4.3 DISCIPLINAS DO RUP

"Uma Disciplina é uma categorização de Tarefas baseadas na semelhança de preocupações e cooperação de esforço de trabalho" (IBM, 2006). Outro fator que motivou a organização do RUP em disciplinas foi a possível categorização de várias tarefas que representam o desenvolvimento de trabalhos relacionados pela mesma disciplina. Além do que, por meio das disciplinas encontrou-se uma forma de padronização do trabalho que ela categoriza. Em sua organização, o RUP baseia-se na estrutura de disciplinas fundamentais consideradas atualmente pela engenharia de software. As disciplinas fundamentais identificadas são (IBM, 2006):

a) Requisitos: mostra como captar os requisitos dos clientes e transformar em um conjunto de requisitos detalhados sobre o que faz o sistema;

b) Análise e Design: explica como transformar os requisitos em produtos de trabalho especificando o design do software do projeto;

c) Implementação: explica como desenvolver, organizar, testar de forma unitária e integrar os componentes implementados de acordo com as especificações do design;

d) Teste: fornece orientação sobre como avaliar a qualidade do produto;

e) Implantação: descreve as atividades necessárias para garantir que o produto de software esteja disponível a seus usuários;

f) Gerenciamento de Configuração e Mudança: explica como controlar e

sincronizar a evolução do conjunto de produtos de trabalho que compõem o sistema de software;

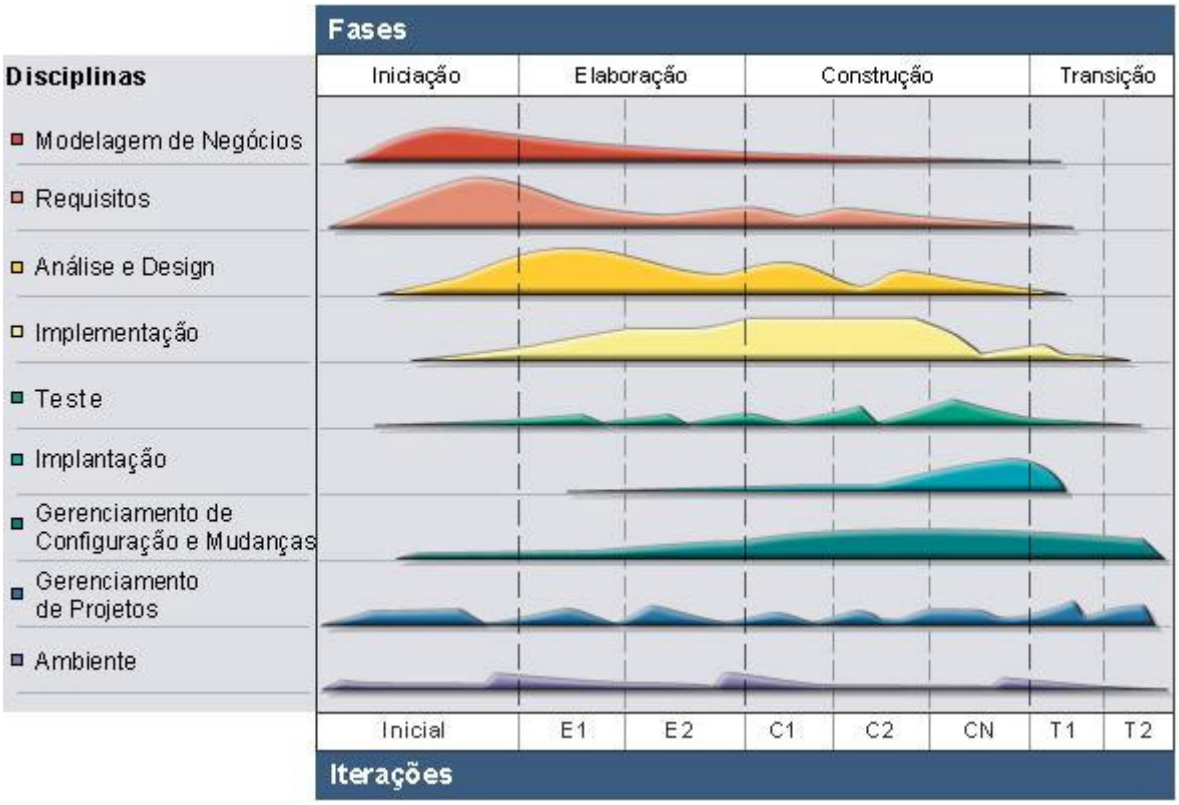
g) Gerenciamento de Projeto: detalha o planejamento do projeto, gerenciamento de riscos, monitoramento do progresso e métricos e

h) Ambiente: organiza o ambiente de trabalho para o processo de software, dando suporte à equipe, inclusive quanto aos processos e ferramentas.

4.4 ARQUITETURA GERAL DO RUP

A Figura 1 demonstra a arquitetura geral do RUP e é interpretada do seguinte modo (IBM, 2006): no eixo horizontal estão representados os aspectos dinâmicos do processo ao longo do tempo e como isso se reflete em termos de ciclos, fases e iterações. Já no eixo vertical estão representados os aspectos estáticos do processo, como os artefatos envolvidos e fluxos. Além disso, percebe-se no gráfico que cada fase do RUP pode ser subdividida em iterações.

Figura 1: Arquitetura geral do RUP



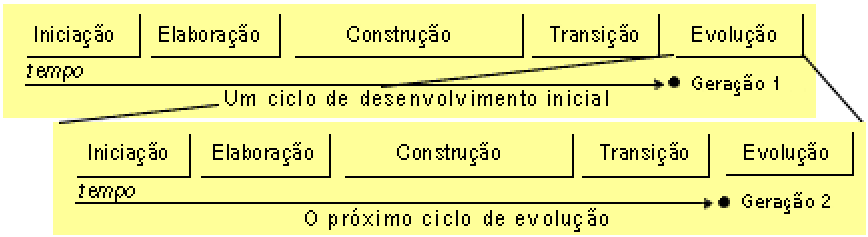
Fonte: IBM (2006).

Como visto na figura anterior, as fases do RUP são compostas por uma ou mais iterações. A divisão em fases é adotada para organização dos ciclos que compõem o **ciclo de vida** do RUP (IBM, 2006). O ciclo de vida do RUP é composto por um **ciclo de desenvolvimento** inicial e vários **ciclos de evolução** subseqüentes.

Como mostra a Figura 2, o ciclo de desenvolvimento e os ciclos de evolução nada mais são do que uma passagem breve pelas quatro fases do RUP e o início de uma nova fase de evolução. Cada ciclo pode ser chamado de uma nova **geração** do software, pois cada passagem pelas quatro fases produz uma geração do software, uma nova versão executável. À medida que o produto atravessa vários ciclos, são

produzidas novas versões, ou seja, o software evolui a cada nova versão até chegar à sua versão final.

Figura 2: Ciclo de desenvolvimento do RUP.



Fonte: IBM (2006)

4.5 AFINAL, QUANDO SE DEVE UTILIZAR O RUP?

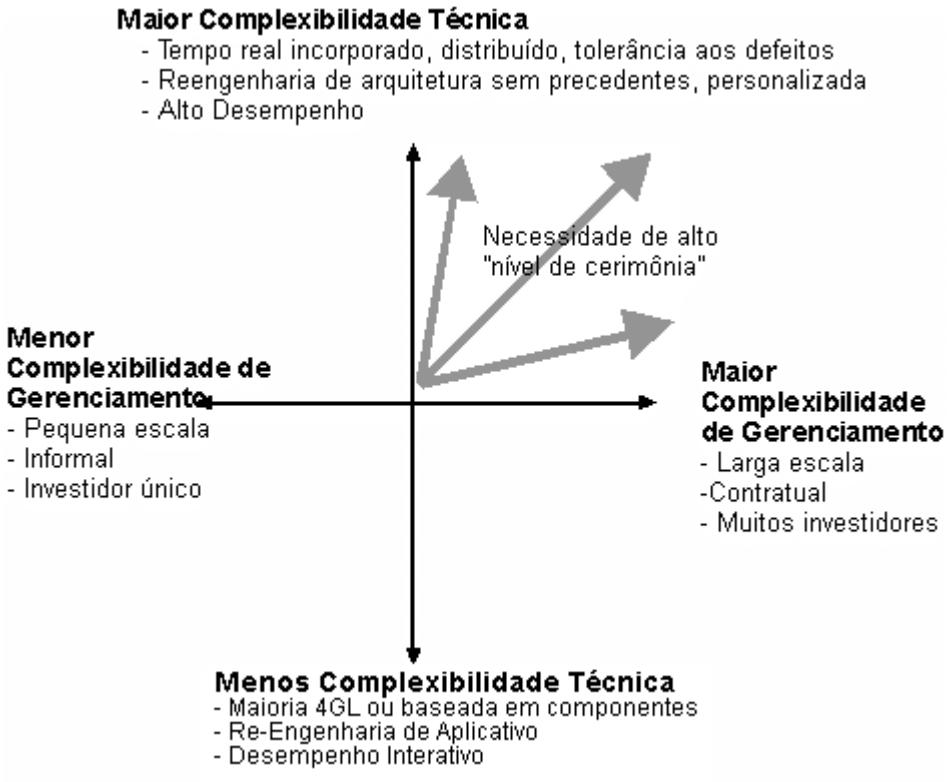
Para se decidir quando é preciso utilizar o RUP, IBM (2006) recomenda analisar as características do projeto em questão, para, assim, classificar o projeto segundo sua complexidade. Projetos com maior nível de complexidade exigem maior "nível de cerimônia" e podem se tornar elegíveis ao uso do RUP. Número de envolvidos, nível de exigência dos requisitos e localidade das equipes envolvidas são características que tendem a aumentar a complexidade de um projeto qualquer. A figura abaixo demonstra as principais características avaliadas no RUP para auxiliar na decisão de uso do RUP (IBM, 2006):

a) ciclo de vida do projeto (número de iterações, duração de cada fase, duração do projeto);

b) meta de negócio do projeto, visão, escopo e risco e

c) amanho do esforço no processo de software.

Figura 3: Complexidade no uso do RUP.



Fonte: IBM (2006).

A IBM disponibiliza duas configurações padrões do RUP chamadas "RUP para Grandes Projetos" e "RUP para pequenos projetos", juntamente com sua ferramenta de composição de métodos *IBM Rational Method Composer*. Segundo a documentação do RUP (2006), pequenos projetos são aqueles que possuem de três a dez pessoas e com duração menor que um ano. A definição da configuração "RUP para pequenos projetos" é para demonstrar que o RUP pode ser utilizado em projetos desse porte da mesma forma que em projetos grandes e muito complexos. Essas duas configurações podem atender à maioria dos projetos se adaptadas às pequenas peculiaridades de cada um. Só o que muda de uma configuração para outra é praticamente o nível de detalhamento, sendo este diretamente proporcional ao tamanho do projeto e a sua

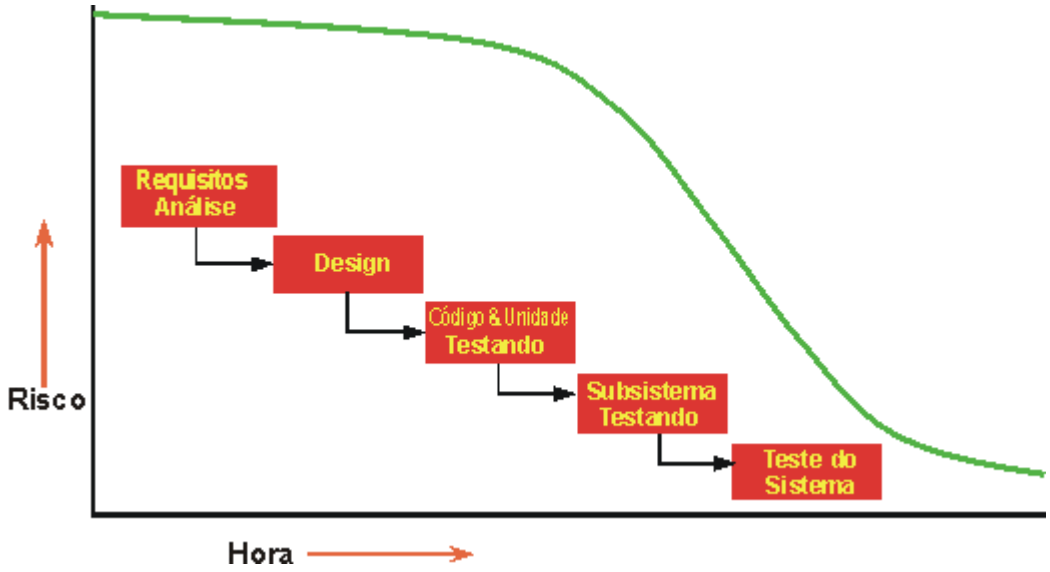
complexidade.

4.6 O MODELO DE PROCESSO DE SOFTWARE ITERATIVO

Dificilmente em um projeto é possível definir o problema a ser resolvido pelo software de uma maneira definitiva. Os requisitos têm grandes chances de sofrerem alterações durante o andamento do processo de software devido às restrições impostas pela arquitetura ou por necessidades do usuário. Um projeto que usa o processo iterativo tem um ciclo de vida que consiste em várias iterações, sendo que cada iteração incorpora um conjunto de tarefas relacionadas à fase atual do projeto. Uma iteração deve ser gerenciada por meio de um planejamento de trabalho para um determinado período de tempo com a definição de escopo e conteúdo ser trabalhado. Cada iteração precisa ser monitorada ativamente para avaliar se o planejamento está sendo cumprido.

É possível que nos primeiros ciclos de vida de um projeto, os artefatos gerados por algumas disciplinas contenham falhas diversas. A descoberta tardia dessas falhas pode resultar em inúmeros transtornos e, em alguns casos, até mesmo inviabilizar um projeto. Dificilmente encontra-se algum projeto qualquer que não contenha riscos envolvidos. Segundo a documentação do RUP, IBM (2006) com o uso de um modelo iterativo de desenvolvimento é possível obter uma diminuição significava dos riscos envolvidos em um projeto.

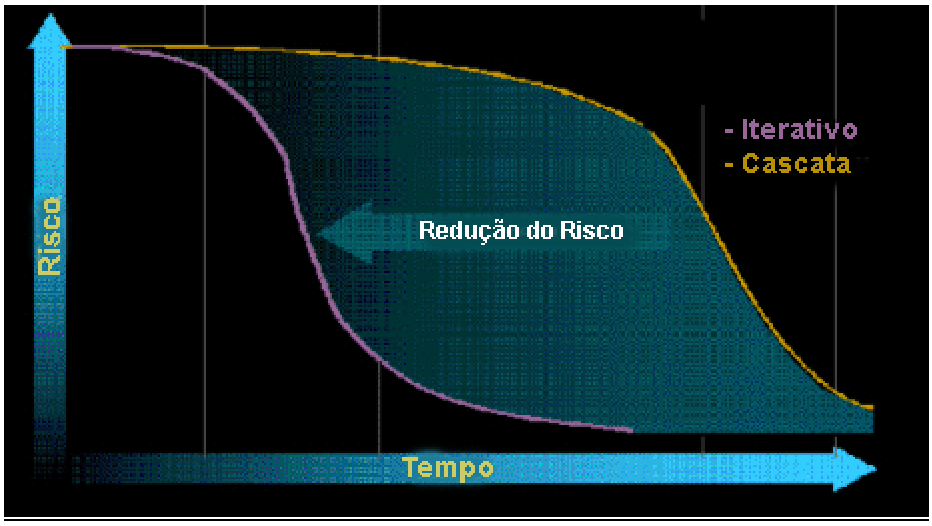
Figura 4: Riscos modelo Cascata



Fonte: IBM (2006).

Em um modelo de desenvolvimento em cascata, não é possível verificar como os riscos estão sendo tratados até o final de todo processo, o que torna mais difícil identificar e tratar possíveis falhas no andamento do projeto.

Figura 5: Redução do risco



Fonte: IBM (2006).

Já em um modelo iterativo, como cita a documentação do RUP, IBM (2006), a seleção das atividades que irão compor uma iteração é feita com base em uma lista de riscos. Dessa forma, é possível acompanhar como os riscos estão evoluindo ao longo de cada iteração. Segundo o mesmo documento, após cada ciclo de evolução é gerada uma nova versão do software, o que também permite avaliar o impacto dos riscos sobre o projeto a cada nova versão em desenvolvimento do software.

O processo iterativo no RUP busca (IBM, 2006):

a) redução inicial dos riscos;

b) maior previsibilidade no decorrer do projeto e

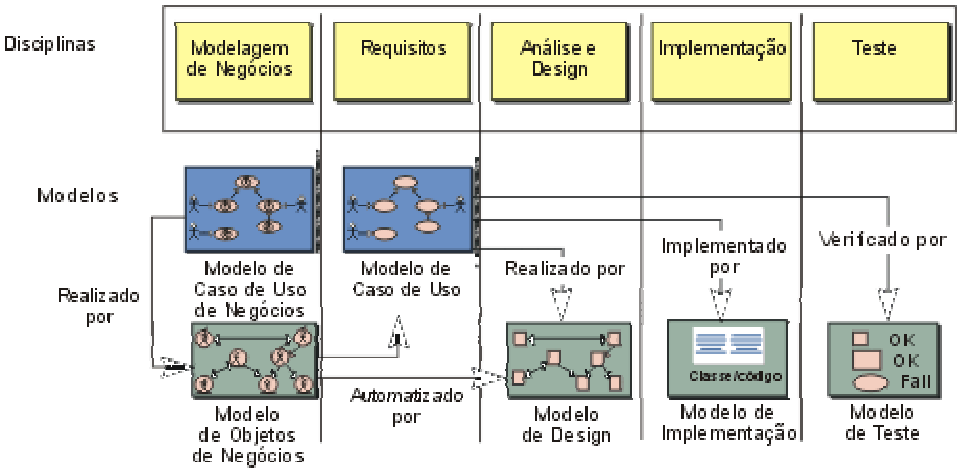
c) confiança entre os investidores.

Um processo repetitivo torna possível acomodar facilmente as alterações para se obter *feedback*, reduzir os riscos inicialmente e ajustar o processo dinamicamente.

4.7 DESENVOLVIMENTO ORIENTADO POR CASOS DE USO

Geralmente, para um modelo de desenvolvimento orientado a objetos, não é fácil descrever como um software irá fazer o que se espera que ele faça. Essa dificuldade resulta da falta de linha de encadeamento das funcionalidades no software quando ele executa determinadas tarefas. Segundo a documentação do RUP, IBM (2006), os casos de uso fazem o papel dessa linha, pois definem o comportamento de um software. Sua importância tornou-se ainda mais evidente pelo fato de que os casos de uso fazem parte da UML.

Figura 6: Disciplina, metodologias e artefatos do RUP.



Fonte: IBM (2006).

Como demonstra a figura acima, o RUP emprega uma "abordagem orientada por casos de uso", o que significa que os casos de uso definidos para um sistema são a base de todo o processo de software.

A documentação do RUP, IBM (2006) utiliza os casos de uso, que fazem parte de várias disciplinas, da seguinte forma:

a) o conceito de casos de uso pode ser utilizado para representar processos de negócios, variante denominada "caso de uso de negócios", abrangida pela disciplina de modelagem de negócios;

b) como requisitos de software, são descritos na disciplina de requisitos e constituem um conceito fundamental que deve ser aceitável para cliente, desenvolvedores e testadores do sistema;

c) na disciplina de gerenciamento de projeto, são utilizados como uma base para planejar o processo iterativo;

d) são realizados em um modelo de design como parte da disciplina de Análise e Design. As realizações de casos de uso descrevem como estes são suportados pelo design em termos de iteração de objetos no modelo de design;

e) como se tornam cenários implementados e testáveis, são um foco importante nas disciplinas Implementação e Teste. Eles são utilizados para originar casos de teste e scripts de teste; a funcionalidade do sistema é verificada executando-se cenários de teste que praticam cada caso de uso;

f) na disciplina de Implementação, os casos de uso formam uma base para o que está descrito nos manuais do usuário. Eles também podem ser usados para definir as unidades de pedido do produto. Um cliente pode, por exemplo, obter um sistema configurado com uma combinação específica de casos de uso.

Neste capítulo foi apresentado o método estruturado RUP e suas disciplinas. Este trabalho está restrito a suas disciplinas Análise e Design e a Implementação. No próximo capítulo serão apresentadas, de forma detalhada, as disciplinas envolvidas neste trabalho. Também será apresentada a diretriz de Revisão do RUP que será utilizada como forma de atingir os objetivos específicos do trabalho.

5. MINIMIZANDO PROBLEMAS DE ANÁLISE NO PROCESSO DE SOFTWARE BASEADO NO MÉTODO RUP

Neste capitulo, serão apresentadas as diretrizes seguidas na busca dos objetivos propostos para este trabalho e os relatos do estudo de caso realizado. Na medida do possível, e do permitido pelas circunstâncias, será seguida a implementação das diretrizes de revisões técnicas definidas pelo RUP e utilizada neste trabalho como meio para atingir os objetivos estabelecidos.

5.1 DISCIPLINAS NO RUP

São abordadas aqui apenas as disciplinas de Análise e Design e Implementação, necessárias para este estudo.

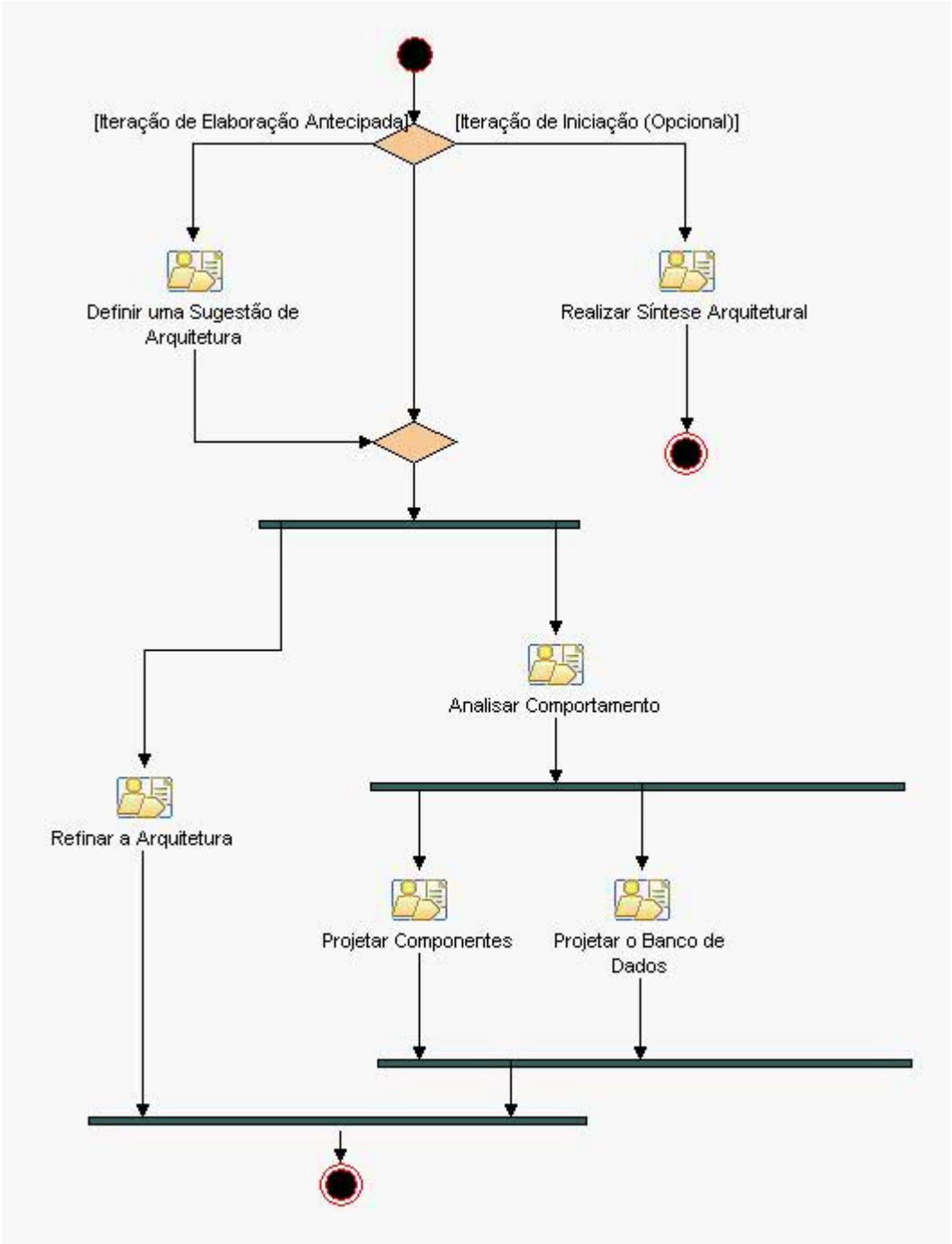
5.1.1 Análise e Design

Segundo a documentação do RUP, IBM (2006), essa disciplina fala de como transformar os requisitos do sistema em produtos de trabalho, especificando o design do software que o projeto desenvolverá. As finalidades desta disciplina são transformar os requisitos em um design do sistema a ser criado, desenvolver uma arquitetura sofisticada para o sistema e adaptar o design para que corresponda ao ambiente de implementação.

A disciplina Análise e Design é relacionada, neste trabalho, a outras disciplinas, como a de requisitos e a de Implementação. A disciplina de Requisitos por fornecer a entrada principal para Análise e Design, e a disciplina de Implementação, por

implementar a Análise e Design..

Figura 7: Fluxo base de trabalho proposto pelo RUP para a disciplina de Análise e Design



Fonte: IBM (2006)

Para o projeto em estudo, as tarefas de Análise e Design do RUP utilizadas ao longo do processo foram (IBM, 2006):

a) Análise Arquitetural: concentra-se em definir uma arquitetura sugerida e restringir as técnicas arquiteturais a serem utilizadas no sistema. Ela conta com a experiência obtida com sistemas ou domínios de problema semelhantes para restringir e enfocar a arquitetura, de modo que o esforço não seja desperdiçado com um retrabalho de montagem arquitetural;

b) Análise de Caso de Uso: descreve como desenvolver uma realização de casos de uso em nível de análise a partir de um caso de uso;

c) Criar um Protótipo da Interface do Usuário: essa tarefa explica como desenvolver um protótipo da interface visual com o usuário e obter *feedback* das suas funcionalidades;

d) Descrever a Arquitetura em Tempo de Execução: essa tarefa define uma arquitetura de processo para o sistema em termos de classes ativas e suas instâncias;

e) Design da Classe: define como projetar a estrutura de classes de um subsistema ou componente;

f) Design de Banco de Dados: explica como projetar um banco de dados para implementar persistência dentro de um software;

g) Design de Caso de Uso: define como refinar os produtos da análise de casos de uso, desenvolvendo suas realizações em um nível mais detalhado que contemple seu design;

h) Design do Subsistema: descreve como documentar elementos de subsistemas e seu comportamento, assim como suas dependências relacionadas;

i) Identificar Elementos de Design: explica como identificar subsistemas, classes, interfaces, eventos e sinais;

j) Projetar a Interface com o Usuário: explica como conduzir o design da interface visual com o usuário dando ênfase na utilidade;

l) Revisar a Arquitetura:

- define quando e como conduzir a revisão de uma arquitetura e como endereçar descobertas da revisão;
- revela riscos desconhecidos, observados na programação ou no orçamento;
- detecta falhas no design da arquitetura. As falhas de arquitetura são conhecidas como as mais difíceis para serem corrigidas, as que acarretam mais danos no decorrer do processo;
- detecta uma possível incompatibilidade entre os requisitos e a arquitetura: design excessivo, requisitos fora da realidade do projeto ou falta de requisitos. Em particular, a avaliação pode examinar alguns aspectos geralmente negligenciados nas áreas de operação, administração e manutenção. Como o sistema está instalado? Atualizado? Como fazemos a transição dos bancos de dados atuais?
- avalia uma ou mais qualidades arquiteturais específicas: desempenho, confiabilidade, capacidade de modificação, segurança;
- identifica as oportunidades de reutilização.

m) Revisar o Design: define como conduzir a revisão de um design e como encaminhar as descobertas da revisão;

n) Verificar se o modelo de design obedece aos requisitos do sistema e serve

como base para sua implementação:

- assegura que o modelo de design é consistente no que diz respeito às diretrizes gerais de design;
- assegura que as diretrizes de design atingirão seus objetivos.

5.1.2 Implementação

A disciplina de Implementação diz respeito a como desenvolver, organizar, testar a unidade e integrar os componentes implementados de acordo com as especificações do design (IBM, 2006).

A finalidade da implementação é:

a) definir a organização do código em termos de subsistemas de implementação organizados em camadas;

b) implementar os elementos de design em termos de elementos de implementação (executáveis e outros);

c) testar os componentes desenvolvidos como unidades e

d) integrar os resultados produzidos por implementadores individuais (ou equipes) ao sistema executável.

A disciplina de Implementação limita o seu escopo a como classes individuais devem ser testadas em unidade. O teste do sistema e o teste de integração são descritos na disciplina de teste. A Implementação está relacionada com outras disciplinas:

a) A disciplina de requisitos descreve como um modelo de caso de uso deve capturar requisitos aos quais a implementação deve atender;

b) A disciplina Análise e Design descreve como desenvolver um modelo de design, o qual representa o propósito da implementação, e é a entrada principal para a disciplina de Implementação;

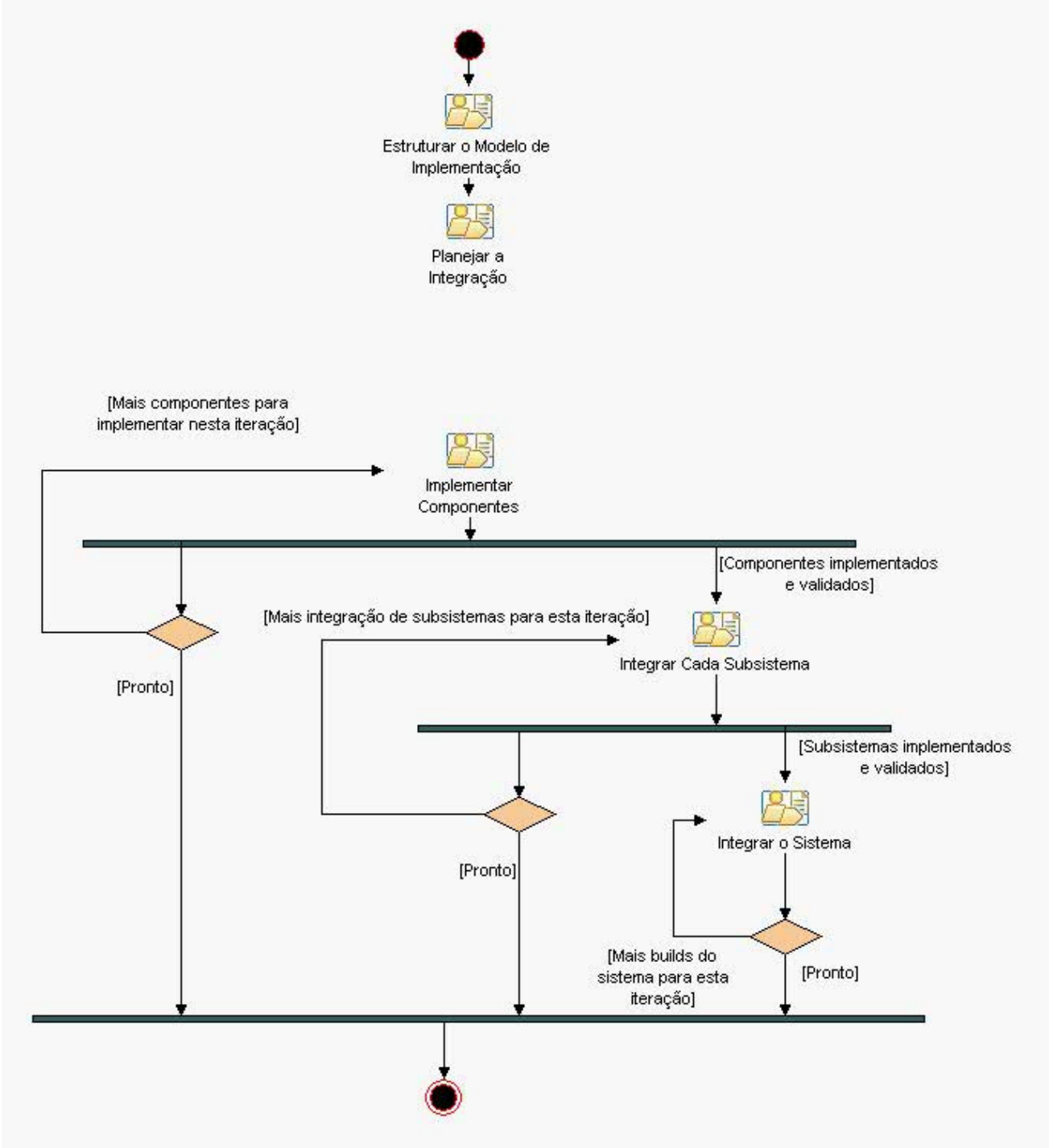
c) A disciplina de teste descreve como realizar o teste de integração de cada construção durante a integração do sistema e também como testar o sistema para verificar se todos os requisitos foram atendidos e como os defeitos são detectados e enviados;

d) A disciplina de ambiente descreve como desenvolver e manter os artefatos de suporte que são utilizados durante a implementação, como, por exemplo, a descrição do processo, as diretrizes de design e as diretrizes de programação;

e) A disciplina Implementação descreve como utilizar o modelo de implementação para produzir e liberar o código para o cliente final;

f) A disciplina gerenciamento de projeto descreve como planejar melhor o projeto. Aspectos importantes do processo de planejamento são o plano de iteração, o gerenciamento de mudanças e os sistemas de controle de defeitos.

Figura 8: Fluxo base de trabalho proposto pelo RUP para a disciplina de Implementação:



Fonte: IBM (2006)

Para o projeto em estudo, as tarefas de implementação do RUP que foram utilizadas ao longo do processo foram (IBM, 2006):

a) Analisar Comportamento do Tempo de Execução: descreve como analisar o comportamento de um componente durante sua execução, para identificar aperfeiçoamentos que podem ser feitos;

b) Estruturar o Modelo de Implementação: descreve como estabelecer a estrutura dos elementos de implementação baseados nas responsabilidades designadas para subsistemas de implementação e seu conteúdo;

c) Executar Testes de Desenvolvedor: delineia como executar e avaliar um conjunto de testes para validar o funcionamento apropriado do componente funcionando apropriadamente, antes que mais teste formal seja executado no componente;

d) Implementar Elementos de Design: descreve como produzir uma implementação para parte do design (como uma classe, realização de caso de uso ou entidade de banco de dados) ou para corrigir um ou mais defeitos. O resultado é normalmente código fonte novo ou modificado e arquivos de dados, referidos geralmente como Elementos de Implementação;

e) Implementar Teste do Desenvolvedor: expõe como criar um conjunto de testes para validar que o componente esteja funcionando apropriadamente, antes que mais teste formal seja executado no componente;

f) Integrar Sistema: descreve como integrar os subsistemas de implementação por partes em uma construção;

g) Planejar Integração de Sistema: revela como planejar a integração do sistema;

h) Revisar o Código:

- descreve como revisar o código para verificar a implementação;
- estabelece pontos de verificação para a implementação e
- prepara registro de revisão e documentar defeitos.

5.2 DIRETRIZ DE REVISÕES NO RUP

Uma revisão é um procedimento formal em que um produto de trabalho é apresentado para as partes interessadas para comentários e aprovação (IBM, 2006). Essa diretriz explica como preparar e conduzir uma revisão.

5.2.1 Passos Gerais

a) Revisões podem ser conduzidas em um formato de reunião e os participantes das reuniões podem preparar algumas revisões eles mesmos;

b) realizar o monitoramento contínuo da qualidade durante as tarefas do processo para impedir que defeitos passem despercebidos. Em cada tarefa no RUP, utilizar uma lista de verificação em reuniões informais de revisão ou durante o trabalho diário (IBM, 2006).

5.2.2 Tipos de Revisão

Em um glossário de padrões de IEEE (2004) são definidos três tipos de revisão:

a) Revisão: uma reunião formal na qual um produto de trabalho, ou um conjunto de produtos de trabalho, é apresentado ao usuário, ao cliente ou a outras partes

interessadas para comentários e aprovação;

b) Inspeção: uma técnica de avaliação formal na qual os produtos de trabalho são examinados em detalhes por uma pessoa ou um grupo diferente do autor para detectar erros, violações dos padrões de desenvolvimento e outros problemas;

c) Navegação: um processo de revisão na qual um desenvolvedor orienta um ou mais membros da equipe de desenvolvimento por meio do segmento de um produto de trabalho que ele/ela escreveu, enquanto outros membros fazem perguntas e comentários sobre técnicas, estilos, possíveis erros, violações dos padrões de desenvolvimento e outros problemas.

Quando implementadas em equipes, as revisões também fornecem oportunidades para que as pessoas acompanhem o design e o código de outros grupos, aumentando as chances de detectar códigos-fonte comuns e reutilizar esses códigos (IBM, 2006). As revisões também fornecem uma forma de coordenar o estilo da arquitetura entre vários grupos. No RUP, as revisões são uma parte secundária, mas muito importante para garantir a qualidade. Os principais contribuintes de qualidade no RUP são bem descritos em Royce (1998), na seção Inspeções de Período.

5.2.3 Planejamento

Planejar revisões é determinar o foco e o escopo da revisão para garantir que todos os participantes entendam suas funções e metas (IBM, 2006). A documentação do RUP recomenda, antes de uma revisão, definir seu escopo, determinar as perguntas

a serem feitas, definir o que será avaliado e por quê. As perguntas exatas dependerão da fase do projeto, por exemplo: revisões anteriores se concentrarão em problemas amplos de arquitetura, já revisões posteriores serão mais específicas.

Depois de definido o escopo da revisão, é preciso definir seus participantes e as informações para executá-la. Ao selecionar os participantes, é interessante estabelecer um equilíbrio entre a especialização de arquitetura de software e a especialização de domínio. A documentação do RUP (2006) também cita a necessidade de eleger-se um líder para a avaliação que coordenará a revisão.

Segundo essa documentação, é recomendado que o número de revisores seja menor que sete e maior que três integrantes. Muitos revisores reduzem, na verdade, a qualidade da revisão, tornando a reunião mais longa e a participação mais difícil ao injetar problemas e discussões à reunião. Com menos de quatro revisores é aumentado o risco de "miopia" na revisão, à medida que a diversidade de preocupações é reduzida.

Os revisores devem ser experientes na área a ser revisada; para casos de uso, os revisores devem compreender o domínio do problema; para a arquitetura de software é necessário ter conhecimento das técnicas de design de software.

Os revisores apropriados para o material são:

a) aqueles que têm conhecimento para entender o material apresentado e

b) aqueles que têm um papel ativo na qualidade do produto ou do produto de trabalho sendo revisado.

5.2.4 Conduzindo as Revisões

Existem vários passos para se conduzir uma revisão bem-sucedida (IBM, 2006):

a) Compreender o processo de revisão, que, de forma geral, segue um ciclo repetitivo:

- um problema é levantado por um revisor;
- o problema é discutido e potencialmente confirmado;
- um defeito é identificado e
- os passos se repetem até que nenhum outro problema seja identificado.

b) Certificar-se de que os revisores entendam suas funções:

- o moderador deve garantir que a revisão siga a pauta e mantenha-se focada nos tópicos apresentados;
- o anotador controla o que foi discutido e documenta as ações a serem executadas. Designar essa tarefa a um dos revisores, essencialmente, o mantém fora da discussão;
- o apresentador é o autor do produto de trabalho sob revisão. O apresentador explica o produto de trabalho e as informações adicionais necessárias para compreendê-lo (se o produto de trabalho não for auto-explicativo, ele provavelmente dará trabalho). O apresentador está lá para iniciar a discussão, responder às perguntas e oferecer esclarecimentos;

- os revisores levantam os problemas. É importante manter o foco e não se deter em discussões paralelas sobre como solucionar o problema. O foco deve estar no resultado e não nos meios.

c) Ter um moderador, um líder nas reuniões de revisão, para não deixar o foco dos revisores se perder. A função do moderador é iniciar a discussão e garantir uma participação igualitária de todos os envolvidos;

d) Manter a revisão estritamente de acordo com a pauta. As revisões são mais eficientes quando são breves e focadas em objetivos bem identificados. Como é difícil manter a concentração por períodos longos, e como os revisores têm outros trabalhos a serem feitos, uma reunião deve durar no máximo duas horas. Se uma revisão tem previsão de ser longa, o melhor é dividi-la em várias revisões mais focadas e pequenas;

e) Identificar os problemas, mas sem tentar corrigi-los. Uma das principais razões para a falta de êxito das reuniões de revisão, segundo a documentação do RUP (2006), está relacionada à fuga de foco dos revisores para discussões de como um problema deve ser corrigido. A revisão deve permanecer concentrada na identificação de problemas, para isso, o moderador precisa impor-se de forma considerável.

5.2.5 Executando Ações com Base nos Resultados da Revisão

A revisão não terá muito valor se nada surgir dela (IBM, 2006). Na conclusão da revisão:

a) priorize a lista de problemas;

b) crie defeitos para controlar problemas e suas resoluções;

c) se for necessária uma investigação adicional, designe uma pessoa ou equipe para pesquisar o problema (e não para solucioná-lo);

d) para os problemas que podem ser resolvidos na iteração atual, designe uma pessoa ou uma equipe para corrigir o problema e

e) alimente a lista de problemas não resolvidos em futuros esforços de planejamento de iteração.

5.3 A FUNÇÃO DO REVISOR TÉCNICO NO RUP

Essa função contribui com o *feedback* técnico sobre os produtos de trabalho do projeto. Ao se criar em um projeto a função de Revisor Técnico, é necessário considerar as habilidades exigidas para a função e as diferentes abordagens que podem ser feitas para designar a função para uma boa equipe.

Figura 9: Relacionamentos.

Fonte: IBM (2006)

Uma pessoa que desempenha a função de Revisor Técnico precisa ter habilidades e conhecimento apropriados, incluindo:

a) conhecimento do domínio ou conhecimento de assunto apropriado ao produto de trabalho que está sendo revisado;

b) habilidades requeridas para produzir o produto de trabalho que está sendo revisado;

c) responsabilidade por outros produtos de trabalho, conteúdo do qual esse produto de trabalho é uma transformação ou que, de alguma maneira, reflita o contrário, e

d) responsabilidade pelas tarefas subseqüentes nas quais esse produto de trabalho será consumido.

A função de Revisor Técnico é designada a uma ou mais pessoas dependendo de cada caso, de acordo com o(s) produto(s) de trabalho que estão sendo revisados, as equipes envolvidas e a disponibilidade dos membros da equipe de tomarem parte na revisão.

Neste capítulo foram apresentados pontos importantes para o desenvolvimento do estudo de caso, conceitos necessários para a solução dos problemas encontrados na disciplina de Implementação. A disciplina de Análise e Design é a responsável pelo surgimento dos problemas que impactam na Implementação de software. O próximo capítulo descreve o estudo de caso realizado. As dificuldades encontradas, as ações tomadas e os resultados obtidos.

6. ESTUDO DE CASO

Este estudo de caso tentou minimizar o grande número de problemas que o projeto TV, que será apresentado na próxima seção, estava tendo na em sua Implementação. Neste estudo de caso, serão levados em conta apenas os problemas provenientes da Análise e Design que, segundo o gerente de projeto, é onde está sendo gerado o maior número de problemas detectados na Implementação.

Logo ao iniciar os estudos para a realização deste trabalho percebeu-se que a diretriz de Revisão do RUP não vinha sendo aplicada no projeto estudado. Nehuma forma de avaliação dos problemas encontrados nos produtos do trabalho era realizada. Tomando como base que a documentação do RUP, que deixa claro que revisões dos produtos de trabalho devem fazer parte das atividades básicas do RUP, a aplicação da diretriz de revisão foi a forma definida para se buscar cada objetivo estabelecido neste trabalho.

"*Diretriz é uma linha segundo a qual se traça*

um plano de qualquer caminho."

(Michaelis, 2007)

O tipo de revisão aplicado foi a Revisão Técnica do tipo inspeção. A inspeção foi escolhida por ser a mais fácil de aplicar e manter. Ela se baseia na avaliação de produtos de trabalho por uma pessoa que não o seu autorer. Essa avaliação busca detectar possíveis problemas.

Dentre os principais motivos para que essa diretriz ainda não estivesse sendo usada até aquele momento podemos citar:

a) a falta de conhecimento especializado dos envolvidos no projeto sobre a

existência e a importância da aplicação das diretrizes do método estruturado RUP no projeto TV;

b) o cronograma rígido de projeto.

Além destes fatores, a aplicação da diretriz de Revisão implica alocação de recursos capacitados para a formação de uma equipe de revisão, como mostram as seções anteriores. A existência de uma equipe de revisão não havia sido prevista no planejamento do projeto para a Fábrica. A princípio isso contribuiu para que a diretriz de revisão não fosse aplicada mesmo após ser estudada pelos responsáveis do projeto.

6.1 DESCREVENDO A ORGANIZAÇÃO E O PROJETO DE PESQUISA

O estudo de caso foi realizado em uma organização multinacional norte-americana situada na Ilha de Santa Catarina - SC. Por solicitação dessa organização, não será revelada sua razão social e será adotado o codinome "Fábrica" para ela. No início deste estudo, mais de 120 funcionários atuavam na Fábrica em sua representação de Florianópolis. Grandes organizações brasileiras das áreas de manufatura, finanças, comunicação, transporte e energia fazem parte da gama de clientes da Fábrica em todas as suas representações no território nacional.

O projeto de software a ser estudado foi desenvolvido para uma organização nacional que atua na área de comunicação, mais especificamente no mercado de TV a cabo. Também não foi autorizada a revelação da razão social dessa organização, então, por este motivo, será adotado o codinome "Cliente TV" para a mesma. Esse projeto de software engloba o desenvolvimento de módulos web para Gestão de

Relacionamento com o Cliente, ou apenas CRM e Emissão de Boletos. Daqui em diante, esse projeto de software específico será tratado pelo codinome "TV".

O envolvimento entre a Fábrica e o Cliente TV deu-se por meio do sistema de trabalho fábrica de software. O sistema de trabalho adotado leva esse nome por ser comparado com as fábricas de linhas de montagem comuns nas últimas décadas. Para o caso específico, a contratada Fábrica ficou responsável pela fase de implementação, enquanto o contratante Cliente TV ficou responsável pelas fases de iniciação, elaboração, testes e implantação do processo de software. Como o Cliente TV foi o responsável pelas fases citadas acima e sua sede está situada no Estado de São Paulo, o projeto TV terá necessariamente equipes espalhadas geograficamente em diferentes unidades da Federação.

O projeto contou com uma equipe de 20 colaboradores somente na Fábrica, responsável pela fase de implementação. Destes 20 colaboradores, eram 15 desenvolvedores, um coordenador de projeto e quatro revisores técnicos. Havia mais 40 colaboradores no Projeto, entre eles um gerente de projeto, quatro analistas de negócio, dez analistas de sistemas, cinco arquitetos, 15 desenvolvedores e cinco analistas de testes.

Tomando como base a documentação do RUP (IBM, 2006) e analisando as características do projeto em estudo referentes ao seu número de colaboradores, sua estimativa de duração – aproximadamente dois anos – e a distância geográfica de seus colaboradores, pode-se classificar o projeto TV como sendo um projeto de grande porte.

6.2 OPERACIONALIZAÇÃO DA PESQUISA

Foi seguida a diretriz do RUP que define a função de revisor técnico em um projeto de software para se atingir os objetivos traçados. Utilizou-se o método de estudo de caso, de modo a avaliar os problemas e necessidades encontrados em um projeto de software de grande porte aplicado ao método RUP.

Após avaliar os problemas, foi aplicada na prática a diretriz de Revisão Técnica do RUP no processo de software para tentar minimizar e, se possível, solucionar os problemas encontrados. Para descobrir os problemas e necessidades a serem solucionados, foram criados dois documentos: primeiramente, uma lista de itens a serem revisados e, logo na seqüência do trabalho, um arquivo histórico de defeitos.

O propósito deste estudo de caso foi esclarecer e descrever quais os pontos do processo de software deveriam ter seus artefatos revisados mais minuciosamente e quais os resultados da aplicação da diretriz de Revisão Técnica sobre os problemas de implementação.

6.3 DADOS: TIPO, TÉCNICAS DE COLETA E TRATAMENTO

Em um primeiro momento, e somente quando os problemas começaram a aparecer, os dados foram levantados para que se tivesse um mínimo de informações para a criação da lista de itens para revisão e escolha dos artefatos a serem revisados. A partir de um breve questionário, foram realizadas reuniões da equipe de Revisão Técnica para definição dos itens e artefatos a serem avaliadas. Após essas atividades, surgiu um documento de *checklist* para auxiliar nas revisões. Os itens do *checklist* de

revisão e a quantidade de vezes que eles ocorrem foram a principal fonte de dados e informações para esta pesquisa.

6.4 LIMITAÇÕES DA PESQUISA

A principal limitação dessa pesquisa é o fato de que este estudo precisou seguir a interpretação da diretriz de Revisão Técnica feita pelo gerente de projetos do projeto TV, ou seja, o quão a revisão de artefatos é considerada importante dentro do projeto. É possível que os gerentes do projeto TV não considerem necessário aplicar níveis de revisões nos diferentes níveis do processo de software, como, por exemplo, no levantamento e análise de requisitos (LEFFINGWELL, 2003). É provável que as revisões sejam implantadas apenas nos níveis onde a quantidade de problemas esteja impactando o bom andamento das tarefas.

Outro ponto a ser considerado é o impacto que um levantamento de defeitos sobre um produto de trabalho causará nas pessoas que realizaram o trabalho. Ao trazer uma nova cultura para o dia-a-dia em um ambiente onde já existe uma rotina, tem-se o trabalho de adaptar as pessoas.

6.5 APLICANDO A DIRETRIZ DE REVISÃO TÉCNICA DO RUP

Logo no início do projeto TV, o número de problemas com os artefatos de análise foi alarmante. Vários outros problemas, relacionados principalmente à arquitetura, surgiram e iriam ser uma constante durante todo o projeto. Não entrarei no mérito desses problemas nesta pesquisa por dois motivos: desvio do foco estabelecido e o comprometimento, não podendo participar das tarefas que estavam fora do contexto exclusivo da disciplina de Implementação do software.

6.5.1 Planejando as Revisões

O foco das revisões dentro do projeto voltou-se para tentar evitar e solucionar os problemas que impactam na implementação do software na Fábrica. Já o escopo da pesquisa ficou restrito apenas aos problemas que impactam na implementação de software e que sejam provenientes das tarefas de Análise e Design. Apesar de ser provável a identificação de problemas relacionados também a tarefas de arquitetura, testes, qualidade de software, entre outros, tais problemas não foram abordados na pesquisa.

A gerência do projeto logo definiu uma equipe de revisores técnicos com bons domínios sobre as atividades e artefatos que englobam a disciplina de Análise e Design do RUP. A escolha de profissionais com as qualidades certas para a realização das revisões é de grande importância para o sucesso das tarefas de revisão (IBM, 2006). A equipe era formada por quatro integrantes e o integrante mais experiente foi indicado pelo gerente de projetos para ser o líder da equipe.

Para definir o que seria avaliado nas revisões iniciais, um questionário foi aplicado para levantar os problemas que estavam ocorrendo. Por se tratar de um primeiro questionário, as perguntas são mais amplas para a detecção dos problemas e, apenas no decorrer das revisões, foi feito o detalhamento dos problemas e sua classificação. As perguntas feitas foram:

> Das opções abaixo cite dois itens que reflitam os problemas encontrados com mais freqüência:
> - problemas referentes à clareza, coerência e coesão com que os documentos de requisitos funcionais e casos de uso foram escritos.
> - problemas referentes aos artefatos UML. Defeitos na criação do artefato ou falta de sincronismo entre os artefatos de um mesmo requisito funcional.
> - problemas referentes a protótipos de tela.
> - problemas referentes às opções a seguir: integridade, versionamento, duplicidade, ou nomenclatura.
> - outros problemas, descreva estes problemas de forma breve.

6.5.2 Conduzindo as Revisões

Para se adaptar ao projeto TV, as funções da diretriz de revisão foram distribuídas para os integrantes da equipe de revisão da seguinte forma: todos os membros da equipe possuíam a função de revisor e também a função de anotador. Para não sobrecarregar nenhum dos envolvidos com a função de moderador, esta foi dividida entre o líder da equipe de revisores e o representante do Cliente TV na Fábrica. A função desse representante do Cliente TV dentro do projeto resumia-se a ser um ponto focal de comunicação entre a Fábrica e o Cliente TV. Dentro do projeto, ainda existe a função do apresentador, que nada mais é do que o autor do produto de trabalho sob revisão.

O processo de revisão também tem seu ciclo adaptado da diretriz do RUP para a realidade do projeto TV. Cada revisor recebeu um documento de requisito do sistema para avaliar. O número de casos de uso que esse documento de requisito de sistema possui é sempre um ou mais, conforme aumento da complexidade na funcionalidade envolvida. Além dos documentos de caso de uso, o documento de requisito de sistema também possui uma especificação técnica respectiva. Essa especificação técnica engloba os modelos de dados relacionais, os diagramas UML e os protótipos de tela.

Com esses componentes em mãos, o revisor faz o levantamento de todos os prováveis problemas de cada artefato para, em seguida, comunicar os prováveis problemas ao apresentador do produto, que também é o seu autor. Este verifica se os problemas são procedentes ou não, podendo contestá-los ou confirmá-los.

Se o apresentador confirmar um erro, este passa a ser definido como um defeito e precisa ser solucionado, ou seja, um defeito nada mais é do que um erro em um artefato que é confirmado por seu apresentador. Um revisor nunca deve solucionar os defeitos, apenas identificá-los, pois estes serão solucionados pelo apresentador do artefato (IBM, 2006). Esses passos se repetem até que nenhum outro erro seja identificado.

Inicialmente foi criado pela equipe de revisão um questionário inicial para levantar dados sobre os problemas que fossem a base para as primeiras atividades de revisão. O questionário era composto apenas por duas questões de múltipla escolha. Que tinham como objetivo levantar os principais erros, conforme o primeiro objetivo especifico primeiro deste trabalho. Segue o questionário aplicado abaixo:

1- Qual o tipo de problema que você mais encontrou? Marque todas as alternativas que julgar relevante:

() Falta de informações nos artefatos

() Falta de clareza e coesão nos artefatos de Requisitos de Sistema

() Falta de clareza e coesão nos artefatos de casos de uso

() Mal funcionamento da arquitetura do software

() Modelo de dados incorreto ou incompleto

() Diagramas UML incompletos ou divergentes

() Protótipos de tela incompletos ou divergentes

() Outros. Descreva o(s) problema(s):

2- Qual dos problemas listados abaixo gerou atraso no tempo estimado para finalização do trabalho:

() Falta de informações nos artefatos

() Falta de clareza e coesão nos artefatos de Requisitos de Sistema

() Falta de clareza e coesão nos artefatos de casos de uso

() Mal funcionamento da arquitetura do software

() Modelo de dados incorreto ou incompleto

() Diagramas UML incompletos ou divergentes

() Protótipos de tela incompletos ou divergentes

() Outros. Descreva o(s) problema(s):

Após aplicar o questionário inicial aos desenvolvedores o resultado obtido foi utilizado na classificação prévia dos problemas. Está pré-classificação logo deu origem a uma classificação mais específica dos problemas utiliza criação do *checklist* de

revisão e especificada no segundo objetivo do trabalho. Este *checklist* será abordado mais adiante.

a) 40% de todos os problemas encontrados eram referentes aos artefatos UML. Entenda-se por problemas com os artefatos UML defeitos na criação do artefato ou falta de sincronismo entre os artefatos de um mesmo requisito de sistema;

b) 35% dos problemas foram referentes à coerência, clareza e coesão com que os documentos de requisito de sistema e seus casos de uso foram escritos;

c) 25% dos problemas eram referentes a outros fatores variados.

Com os dados iniciais coletados pelo questionário foi possível definir os artefatos que seriam testados e as partes destes que necessitavam de maior atenção. Os documentos de requisito de sistema, casos de uso e as especificações técnicas foram os artefatos que apresentaram necessidade de revisão.

A partir do início das revisões, todos os artefatos citados acima obrigatoriamente seriam avaliados pela equipe de revisão e somente após terem seus problemas solucionados e esclarecidos eram disponibilizados para implementação. Logo no início do trabalho, muitos erros foram sendo encontrados. Após o revisor apurar todos os erros existentes em um artefato, ele os informou aos apresentadores. Desta forma o responsável pela criação dos artefatos era a pessoa responsável pela sua correção.

Com esta dinâmica o processo de revisão faz com que com que os problemas sejam corrigidos mais rapidamente e que o apresentador tenha conhecimento dos erros que cometeu, promovendo uma possível evolução do trabalho individual dos apresentadores.

Por causa da distância geográfica entre as equipes da Fábrica e do Cliente TV, o informe sobre os problemas começou a ser feito por meio de e-mails, como recomenda

a documentação do RUP (IBM, 2006). Assim, foi criado pela equipe de revisão um *template* de documentação de problemas, também chamados de itens da revisão. O único intuito desse *template* é padronizar o envio dos itens de revisão, já que a documentação do RUP (2006) deixa claro que o importante é realizar as revisões e não documentá-las.

Figura 10: *Template* de documentação de problemas.

RF / UC	**NOME**

1.

FÁBRICA	
Artefato do Problema:	
Classe / Tópico / Página:	
Método / Tópico / Página:	
Dúvida:	
Contextualizando a dúvida:	
Observações / Fragmentos de textos de documentações	
CLIENTE	
A dúvida procede?	[ ] SIM [ ] NÃO
Motivo / Detalhes / Reposta:	
FÁBRICA	
A resposta atende?	[ ] SIM [ ] NÃO
Parecer / Contra-resposta	

Fonte: Documentação do projeto TV, fornecida pela Fábrica

Como mostra a Figura 10, o *template* de documentação de problemas contém um cabeçalho com nome dado ao artefato que está sendo avaliado e uma tabela de

dúvida para ser usada em cada item de revisão. O cabeçalho contém o nome do requisito de sistema e/ou o nome do caso de uso avaliado. A tabela de dúvida contém o nome do artefato, o tópico e/ou página, qual o provável problema e a contextualização que o revisor achar necessária. O *template* também possui um campo para que o apresentador, que é chamado de cliente nesse documento, escreva se o problema existe mesmo ou não de acordo com sua visão. O revisor também possui espaço para documentar se a resposta do apresentador atende às suas expectativas.

O *template* foi usado para documentação de problemas da seguinte forma: cada possível erro encontrado era documentado através do preenchimento dos campos do *template.* Em seguida todos os erros eram reunidos e enviados através de um email para correção do apresentador.

De acordo com os problemas que se confirmaram em defeitos foi possível realizar uma classificação mais detalhada destes e constatar os locais onde mais ocorriam. Assim os problemas foram classificados em grupos conforme as suas ocorrências em um mesmo contexto. Por exemplo, quando constatado que estavam ocorrendo muitos problemas com os diagramas UML de seqüência, estes problemas foram classificados como sendo de um mesmo contexto: “Análise crítica dos diagramas de seqüência e sincronismo com os artefatos do mesmo requisito”.

Cada contexto identifica durante a classificação dos problemas deu origem a uma sessão em um novo documento que lista as checagens que um revisor deve realizar. Desta forma foi confeccionada a primeira lista de checagem para auxiliar na avaliação dos artefatos. Mais tarde este documento ficou conhecido como checklist de análise. Com a confecção de um *checklist* de análise se obtêm a vantagem de padronizar e agilizar a aplicação dos procedimentos de avaliação nos artefatos

submetidos. Através do *checklist* é possível repassar conhecimento, já que seguindo a seqüência das sessões do *checklsit* se tem boas chances de uma pessoa com pouca experiência consiga realizar o papel de um revisor.

Figura 11: *Checklist* de análise

Documentações fornecidas
Os documentos e diagramas enviados na tag estão disponíveis?
O Documento de requisitos de sistema está no respectivo diretório?
O Documento de especificação técnica está no respectivo diretório?
O Documento de caso de uso está no respectivo diretório?
O Arquivo de controle de classes está respectivo diretório?
O protótipo de tela está de acordo com a descrição de suas funcionalidades?
Análise dos documentos para cada caso de uso
Leitura e entendimento dos documentos de requisitos de sistema e de caso de uso
Leitura e entendimento da especificação técnica
Se existir protótipo, a narrativa/descrição de tela está coerente?
Integridade dos documentos
Os Diagramas informados na especificação técnica estão legíveis e sincronizados com suas especificações?
Existem diagramas duplicados ou faltantes?
Os métodos informados no diagrama de seqüência estão presentes no controle de classes do pacote?
Se houver tela, o protótipo disponibilizado permite as ações especificadas na especificação técnica?
Análise crítica dos diagramas de seqüência e sincronismo com os artefatos do mesmo requisito
Os relacionamentos entre as classes estão coerentes?

Os loops e condições (If's, etc..) estão sendo representados no diagrama de seqüência quando possuem chamadas a outros métodos?
As classes de domínio estão no diagrama de classes e na planilha de controle de classes?
Os atributos de classes de domínio referenciados na documentação estão presentes nas suas respectivas classes?
As mensagens de erros apresentadas estão com seu código (exceto mensagens de LOG) e presentes na especificação técnica e na planilha de mensagens de erro?
Regras de negócio
Existem trechos (regras de negócio) na documentação que podem gerar incertezas e ambigüidades?
Modelo de dados
As consultas em banco possuem as suas tabelas definidas no Modelo de dados?

Fonte: Documentação do projeto TV, fornecida pela Fábrica

A realização das revisões estava durante muito, em média duração de 16 horas ou 2 dias de trabalho. Foi definido que o processo de revisão iniciava quando os artefatos chegavam na Fábrica e finalizava quando todos os erros encontrados haviam sido solucionados e esclarecidos. A comunicação dos problemas levantados entre os revisores e os apresentadores por meio de e-mails foi desgastante e não eficiente como se imaginava. Muitos prováveis problemas eram contestados pelos apresentadores e algumas destas contestações eram também contrariadas pelos revisores, gerando impasses que só eram resolvidos nas reuniões de revisão com a ajuda dos moderadores. Entretanto, algumas vezes o desgaste entre as pessoas já havia se estabelecido.

Quando se avalia o trabalho feito por pessoas é preciso ser tão cuidadoso com o tratamento dessas pessoas quanto com a busca de problemas. Normalmente todos tem

domínio de seu trabalho, mas é difícil aceitar as críticas mesmo quando são construtivas (IBM, 2006).

Diante dessa demora na conclusão das revisões e das constantes contestações de itens revisados, o trabalho dos revisores da Fábrica chegou a ser alvo de críticas por parte do Cliente TV. Essas críticas alegavam baixo padrão de qualidade do trabalho da equipe de revisão. Para avaliar se a crítica do Cliente TV procedia, a Fábrica colocou um auditor para avaliar a lista de problemas levantados nas revisões antes que estas fossem enviadas aos apresentadores. Para a função de auditor era necessário escolher uma pessoa qualificada e com experiência, então, o líder da equipe de desenvolvimento acabou acumulando a função de auditor temporariamente.

O auditor realizava a conferência da real existência de cada erro levantado pelos revisores. Como já poderia era possível prever, esta atividade agregou mais complexidade ao processo de revisão e, em alguns casos, contribuiu para o aumento do tempo da revisão. Em menos de duas semanas de auditoria, a Fábrica chegou à conclusão de que não havia nada de errado com a qualidade do trabalho dos revisores. Para o gerente de projeto da Fábrica, o problema estava na falta agilidade da comunicação entre os revisores e os apresentadores.

Então após 3 semanas de trabalho, tendo uma amostra de 24 requisitos revisados, foi constada uma alta média de erros por requisito. Cerca de 23 problemas eram encontrados em média nos artefatos que contemplavam cada requisito.

Como manda a diretriz de revisão, foram tomadas algumas atitudes corretivas com base nos resultados obtidos. Com isso foi possível viabilizar a realização do terceiro objetivo específico deste trabalho, buscar uma forma de solucionar os problemas levantados.

Algumas atitudes baseadas nos resultados das revisões já vinham sendo encaminhadas desde o início das atividades de revisão. A classificação dos problemas e a criação de um *checklist* de análise foram dois exemplos. Além disso, outra atitude corretiva foi tomada, a forma de comunicação entre os revisores e os apresentadores foi mudada: o envio de e-mails foi substituído pela teleconferência de revisão. A teleconferência de revisão nada mais é do que uma reunião de revisão realizada entre o revisor, o apresentador e um moderador via telefone.

Logo que o revisor terminava seu levantamento, a teleconferência de revisão era agendada com o moderador e o apresentador. Iniciada a teleconferência de revisão, cada item levantado era relatado pelo revisor e confirmado, ou não, pelo apresentador. Esse era o processo para cada item da lista de problemas. Após as 3 semanas iniciais de trabalho o *checklist* de análise já estava pronto e sua implantação, juntamente com a implantação da teleconferência de análise. A implantação destas atitudes corretivas levaram 32 horas ou 4 dias de trabalhado para serem concluídas pela equipe de revisores.

Com a rápida solução dos problemas por meio das teleconferências de revisão, o número de dificuldades encontradas na implementação dos requisitos de sistema conseguiu ser reduzida. Apesar da melhoria, o cronograma ainda estava sendo impactado. Desde a chegada na Fábrica de novos documentos referentes a um novo requisito de sistema, até o início de seu entendimento pelo desenvolvedor responsável pela sua implementação, eram despendidas, em média, 12 horas de trabalho para liberação desse requisito. Uma diminuição de cerca de 4 horas em média em cada revisão. Mesmo com esta redução, o tempo ainda era considerado alto pela equipe revisão e comprometedor ao cronograma de trabalho pelo gerente de projeto da

Fábrica.

Então após mais 7 semanas de trabalhos e uma amostra de 56 requisitos revisados mais conclusões puderam ser obtidas. Segundo levantamento da equipe de revisão, uma boa solução estaria em tratar a origem dos problemas e não uma rápida solução desses. Em outras palavras, se os problemas mais comuns fossem evitados, muitas teleconferências de revisão poderiam ser evitadas ou, pelo menos, teriam seu tempo de duração reduzido para o projeto TV. Após realização de uma reunião entre o gerente de projetos, a equipe de revisão e o representante do Cliente na Fábrica, chegou-se à conclusão de que seria necessária uma padronização dos artefatos de Análise e Design entre a equipe de revisão da Fábrica e a equipe de Análise e Design do Cliente.

Então, a Fábrica enviou dois representantes ao Cliente: o líder da equipe de revisão e mais um experiente integrante dessa equipe para realizar os trabalhos de padronização junto ao Cliente, também chamados pelas equipes de alinhamento. Pelo lado do Cliente, também foram designados dois representantes: o líder da equipe de análise e mais um experiente integrante dessa equipe.

Os trabalhos foram caracterizados por reuniões de apresentação dos processos e padrões utilizados pelas equipes e por reuniões para confecção de um *checklist* unificado padrão para as equipes. É feita aqui uma ressalva: apesar de todo projeto utilizar o método estruturado RUP, a configuração do processo para a realidade do projeto TV pode gerar confusão. Por este motivo que foram feitas reuniões de apresentação de processo e padrões entre as equipes. As vantagens de um *checklist* padrão são as seguintes:

a) com a aprovação dos itens de checagem pelo Cliente, a aceitação de críticas

em forma de problemas levantados é maior;

b) a equipe de analistas de sistema baseia seu trabalho nos padrões acordados e que serão posteriormente avaliados;

c) a origem dos problemas é tratada.

O trabalho de alinhamento entre a Fábrica e o Cliente teve duração de uma semana exatamente. Após esse período de alinhamento, foi concebido o *checklist* padronizado para ser à base de avaliação da equipe de revisão. É conveniente lembrar que o *checklist* produzido se aplica somente às necessidades do projeto TV e dificilmente poderá ser usado de forma genérica para atender às necessidades de outros projetos.

Então, neste segundo momento das atividades de revisão foram tomadas as seguintes atitudes: o trabalho das equipes de análise e implementação foi padronizado e a lista de checagem foi corrigida (padronizada). Para a aplicação efetiva destas atitudes foram necessárias 80 horas ou 10 dias de trabalho.

O grande diferencial implantado foi à integração completa dos trabalhos e da comunicação das equipes de análise, revisão e implementação de requisitos. O *checklist* padronizado passou a ser seguido como padrão pela equipe de análise fazendo com que os representantes evitassem e corrigissem problemas corriqueiros antes mesmo dos requisitos e seus documentos estarem prontos para serem enviados à fábrica.

Tais medidas surtiram efeito imediato nos trabalhos de revisão. O tempo médio dos primeiros 9 requisitos revisados na primeira semana após as mudanças foi de cerca de 5,2 horas por requisito revisado. Uma diminuição de mais 6,8 horas por requisitos revisado, considerada muito satisfatória.

Com este resultado as revisões entraram em uma fase de estabilidade. Os problemas foram controlados com eficácia e não foram necessárias novas melhorias no processo de revisão até o fim do projeto TV.

7 DISCUSSÃO DOS RESULTADOS

Os principais objetivos deste trabalho foram reduzir problemas provenientes da Análise e Design dentro da implementação do processo de software e buscar meios de encontrar e solucionar problemas em qualquer projeto que utilize o método estruturado RUP. A pesquisa englobou uma revisão da literatura envolvendo o entendimento de conceitos da engenharia de requisitos, fases do desenvolvimento de software e métodos de processo de software estruturado.

Com a pesquisa tendo seu foco no método estruturado RUP, a documentação do mesmo mostrou que uma importante diretriz não estava sendo aplicada pelo projeto TV. Chamada de diretriz de Revisão Técnica, esse conjunto de instruções procura resolver problemas similares aos problemas que originaram os objetivos deste trabalho.

O projeto TV apresentou sérios impactos em suas atividades causados pelo número demasiado de problemas na implementação do processo de software. Logo ao terminar os estudos da documentação do RUP, concluiu-se que a falta de aplicação da diretriz de Revisão Técnica poderia estar provocando o grande número de problemas do projeto TV.

O resultados concretos obtidos com as atitudes corretivas aplicadas após as 3 primeiras semanas de revisões foram as seguintes: conseguiu-se, em poucas semanas de trabalho, diminuir a média de problemas, que eram de 23, para uma média de 16 problemas por artefatos de um requisito de sistema: uma redução de 31,5% no número de problemas encontrados.

Estes resultados não foram suficientes para o sucesso das revisões no projeto TV e após mais sete semanas os resultados obtidos com as atitudes corretivas tomadas

foram os seguintes: a equipe de revisão conseguiu, em apenas três semanas de trabalho, uma redução da média de problemas por artefatos de um requisito de sistema de 16 problemas para três problemas. Uma redução de 33,4% no número de problemas encontrados.

É importante esclarecer que o autor deste trabalho não possuía autonomia para a aplicação do processo dentro do projeto. A aplicação da diretriz de Revisão Técnica no projeto TV só foi possível porque os impactos dos problemas estavam causando preocupação na gerência do projeto, relacionado a um possível fracasso no cumprimento do cronograma. Nesse contexto, a oportunidade de colocação do estudo em prática foi criada e os trabalhos puderam ser iniciados e concluídos com sucesso.

A diretriz de Revisão mostrou-se uma boa forma de solução dos problemas levantados quando aplicada no estudo de caso. O resultado concreto do estudo de caso mostra que os problemas diminuíram consideravelmente atendendo os objetivos estabelecidos.

8 RECOMENDAÇÕES PARA ESTUDOS FUTUROS

Este trabalho mostrou como aplicar a diretiva de Revisão Técnica do RUP para problemas provenientes da disciplina de Análise e Design. Avaliando-se que a diretriz de revisão do RUP abrange todas as fases do processo de software, algumas pesquisas poderiam ser feitas sobre o impacto do uso ou não dessa diretriz em cada uma das fases do processo de software.

Uma sugestão para trabalho futuro seria uma pesquisa sobre a aplicação ou não de revisões na fase de elaboração do processo de software, mais especificamente na disciplina de requisitos do RUP. Isso porque, segundo (LEFFINGWELL, 2003), entre 40% e 60% de todos os problemas encontrados em um projeto de software são causadas por falhas ocorridas na fase de levantamento de requisitos.

Outro trabalho de pesquisa interessante pode ser realizado dentro da própria disciplina de Análise e Design do RUP, mais especificamente na arquitetura do software que esta disciplina monta ou implementa. No estudo de caso desenvolvido neste trabalho, grande número de problemas foi identificado como sendo provenientes da arquitetura do software. Projetos de grande porte em tempo real, com necessidades grandes de tolerância à falhas e necessidades de arquitetura distribuída podem ser seriamente afetados por problemas provenientes da arquitetura definida para o projeto de software.

9. CONCLUSÃO

9.1 CONCLUSÃO GERAL

O método estruturado RUP prega rotinas bem definidas de revisões para cada fase do processo de software. Uma diretriz inteira e várias funções do método são dedicadas às revisões. Com a aplicação das revisões sempre se espera a diminuição dos problemas em um projeto e a prevenção de problemas futuros.

Neste trabalho, foi aplicada a diretriz de Revisão Técnica do RUP. Os impactos no projeto conseguiram ser bem identificados e a importância de se revisar as atividades dentro de cada fase do processo de software, evitando o rápido aumento do impacto das soluções, ficou evidente. Dado o exposto, os objetivos do trabalho foram alcançados e a contribuição foi dada ao se mostrar à importância das revisões e na prevenção de problemas, principalmente nas fases iniciais do processo de software quando os impactos ao projeto são menores.

9.2 CONCLUSÃO ESPECÍFICA

Como descrito na lista de objetivos deste trabalho, o estudo de caso iniciou-se com um levantamento dos principais problemas de implementação por meio de um questionário inicial aplicado aos desenvolvedores responsáveis pelas implementações do software. Por meio do convívio diário com a equipe de desenvolvedores, foi possível criar um ambiente amigável e com boa comunicação, contribuindo para que o resultado da aplicação do questionário correspondesse às expectativas.

Com as informações provenientes do questionário, foi possível classificar os principais problemas e gerar uma base de checagem para as avaliações. Esse objetivo foi conseguido de forma natural, atendendo às expectativas e sem maiores dificuldades.

Encontrar uma forma de solucionar os problemas de implementação foi um dos objetivos traçados inicialmente para este trabalho, mas, com o decorrer da pesquisa e do estudo de caso, percebeu-se que solucionar os problemas não era o mais importante e nem papel de um revisor. A diretriz de revisão do RUP deixa claro que o importante é encontrar e comunicar os prováveis problemas para que os responsáveis os solucionem.

Para que problemas já encontrados não ocorram novamente foram aplicadas ações baseadas na lista de problemas levantados, como sugere o RUP. Nesse momento, o estudo de caso teve suas maiores dificuldades. Uma boa comunicação entre as equipes de colaboradores envolvidas mostrou-se mais importante do que o esperado, visto que à distância das equipes dificultou muitas as melhorias e desgastou as pessoas. O gerente de projeto precisou impor toda a sua experiência e mostrou também ter uma grande importância para o sucesso da aplicação da diretriz de revisão. A padronização e acompanhamento das atividades desenvolvidas em equipes separadas por longas distâncias mostrou-se fundamental para um bom andamento das atividades.

Os resultados finais foram satisfatórios e os objetivos foram atendidos. O trabalho acabou ficando com escopo bem enxuto, deixando de lado problemas relacionados à arquitetura, ao levantamento de requisitos e à integração de software. O projeto TV só desenvolveu as atividades de revisão quando os problemas já estavam muito avançados e já tinham comprometido o cronograma do projeto. Em projetos que

utilizam o método estruturado RUP, as revisões se mostraram fundamentais para o acompanhamento e controle de problemas em um projeto de software.

Como visto durante o estudo de caso, o número de problemas que podem ocorrer em um projeto que utiliza o método estruturado RUP pode crescer a ponto de impactar no cronograma. Portanto, a importância das revisões se justifica pelo fato de que a correção de algum erro na disciplina de Análise e Design possui um custo muito inferior do que a correção de um erro nas fases mais adiantadas do processo de software, até 20 vezes menor (LEFFINGWELL, 2003).

REFERÊNCIAS

BlasChek, José Roberto. *Gerência de Projetos* - o Principal Problema dos Projetos de Software. Disponível em http://www.bfpug.com.br/islig-rio/

Booch G. *Object-Oriented Analysis and Design with Applications*. 2.ed. Addison-Wesley, 1994,

Humphrey, Watts S. *Managing the Software Process*. Hardback, 1990.

IBM(R) Rational Unified Process(R). Rational Method Composer Versão 7.1.1, 2007. OpenUP Version 1.0 2007. Disponível em <http://epf.eclipse.org/wikis/openup>.

IEEE Computer Society 2004 Version, SWEBOK® Professional Practices Committee Guide to the Software Engineering Body of Knowled.

Kroll, P.; Kruchten P. *The Rational Unified Process Made Easy:* A Practitioner's Guide to the RUP. Addison Wesley, 2003.

Larman, Craig (1995) *Utilizando UML e Padrões* – uma introdução à análise e ao projeto orientados a objetos e ao Processo unificado. 2 ed. Porto Alegre: Bookman, 2004.

Leffingwell, Dean. *Calculating your return on investment from more effective requirements management*. Disponível em http://www.ibm.com/developerworks/rational/library/347.html#bib3

Michaelis Moderno Dicionário da Língua Portuguesa. Editora Melhoramentos Ltda, 2007. Disponível em <http://michaelis.uol.com.br/moderno/portugues/index.php>

Naur, P.; Randell, B. *Software Engineering: Report on a Conference sponsored by the NATO Science Commission.* Garmisch, Germany, 1968.

Royce, Walker. *Software Project Management: A Unified Framework.* Addison-Wesley, 1998.

Pressman, Roger S. *Engenharia de Software.* 5. ed. São Paulo: Ed McGrawHill, 2002.

Rational Unified Process Best Practices for Software Development Teams, 1998. Disponível em <http://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf>

Schwaber, K.; Beedle, M. *Agile Software Development with SCRUM*. Prentice Hall, 2002.

Silva, Ricardo Pereira da. *UML2 em Modelagem Orientada a Objetos.* - Florianópolis, Visual Books, 2007

Soares, A. L.. *Introdução, Identificação e Análise em Engenharia de Requisitos.* Local de publicação: editora, 2005.

Sommerville, Ian. *Software Engineering*. Porto Alegre: Addison Wesley, 2003.

Thayer, R.H; Dorfman, M. IEEE/ANSI 830-1993, 1993. *Software Requirements Engineering*, 1997. http://ieeexplore.ieee.org

Wells, Don. *Extreme Programming*: A gentle introduction. Disponível em <http://www.extremeprogramming.org/>. Último acesso em 12/12/2007.

Westfall, Linda. The Westfall Team; *Software Requirements Engineering: What, Why, Who, When, and How*. Disponível em <http://www.westfallteam.com/>.

Wiegers K.E. *Software Requirements*. Microsoft Press, 1999.